PHD DISSERTATION

Realization of Multipoint Communication over the Internet using XCAST

ODIRA Elisha Abade

Nagoya University

Graduate School of Engineering

Department of Computational Science and Engineering

September 2012

Realization of Multipoint Communication over the Internet using XCAST

481067019 ODIRA Elisha Abade

Abstract

Multipoint communication protocols such as multicast can increase efficiency of bandwidth utilization in the Internet. However, conventional multicast requires maintenance of state-information in a router's Multicat Forwarding Table(MFT). This does not scale well for larger groups and has impeded multicast deployment. Hence multicast variants for small groups have been proposed.

This dissertation is motivated by the fact that explicit multiunicast (XCAST) has been proposed as a complementary protocol to conventional multicast. However, deployment of XCAST in the Internet currently is not easy because the current routers cannot process its packet headers correctly. Furthermore, no studies have been conducted on how QoS provisioning can be achieved in an XCAST network.

The first part of this work proposes an out-of-the-box component called *"XCAST Routing Engine"*, that is connected to core routers and helps in processing of XCAST packets. The second part proposes a QoS-aware XCAST (QS-XCAST) and shows its implementation using a network simulator (OMNeT++). We further extend the model to investigate the feasibility of integrating XCAST into the Locator Identifier Separation Protocol (LISP) aimed at deployment in the Future Internet.

The XCAST Routing Engine provides a simple, efficient and cost-effective way for incremental deployment of XCAST in the Internet. Performance evaluation of the XCAST Routing Engine shows that XCAST processing does not incur heavy penalty on routers. Using OMNeT++, empirical results show that XCAST QoS provisioning using DiffServ provides efficient bandwidth utilization, better packet forwarding fairness between XCAST and non-XCAST traffic, lower traffic load on routers and elimination of collusion attack (*"Good Neighbour Effect"*).

Contents

List of Figures

List of Tables

List of Abbreviations

AF Assured Forwarding

ALM Application Layer Multicast

BE Best Effort

DiffServ Differentiated Services

DSCP DiffServ Code Point

ECN Explicit Congestion Notification

EF Expedited Forwarding

HBH Hop By Hop

IAB Internet Architecture Board

ICMP Internet Control Message Protocol

IESG Internet Engineering Steering Group

IETF Internet Engineering Task Force

Intserv Integrated

IPv6 Internet Protocol Version 6

IRTF Internet Research Engineering Task Force

ISM Internet Style Multicast

JUNOS Juniper Network Operating System

LISP Locator Identifier Separation Protocol

MIB Management Information Base

NETCONF Network Configuration Protocol

OID Object Identifier

OMNeT++ Object Modular Network Testbed in C++

OSI Open Systems Interconnection

PIM Protocol Independent Multicast

QoS Quality of Service

QS-XCAST QoS-aware Explicit Multiunicast

RFC Request For Comments

SAMRG Scalable Adaptive Multicast Research Group

SNMP Simple Network Management Protocol

SSH Secure Shell

UDP User Datagram Protocol

VLAN Virtual LAN

VoIP Voice over Internet Protocol

X2U XCAST to Unicast

XCAST Explicit Multiunicast

Chapter 1

Introduction

The last decade has witnessed an explosion in Internet technologies and devices that connect onto the Internet. While the initial form of the Internet was mainly for research in academia and focused majorly on delivery of data packets, the Internet has since evolved into a critical business tool, taking the center stage as a key tool for delivery of multimedia content (audio and video data). This evolution has also come with an expansive growth of the Internet both in terms of the number of users and the number of Internet-based applications, some of which support life-critical operations while others are very sensitive to packet loss and delay.

One of the dominant questions facing the Internet today is how to achieve implementation of applications on the Internet that are capable of meeting the needs of the numerous users and their devices while keeping such implementations scalable to the billions of the current and future users and the devices which are themselves, diverse in their computing power. Several aspects have been proposed. Among them are two approaches. The first is the use of multipoint communication technologies to ensure scalable and efficient utilization of Internet resources. The second is to re-think the architecture of the Internet into a *"Future Internet architecture"*.

Multipoint communication refers to the transfer of information among a set

of participants through distinct one-to-many communication channels. The channels (paths) are from a single location to multiple end points. Multipoint communication approach was popularized by the need for collaboration on the Internet and saw the emergence of technologies such as multicast[1, 2] that could help minimize the number of packets sent from a given source to multiple recipients.

The Future Internet can simply be defined as the Internet of "tomorrow"; a multi-service Internet that will need to be more powerful, more connected, more intuitive and more a part of our everyday lives, at home, at work and even on the move. This Internet of services, things and infrastructure, will include everything from smart appliances that talk to each other to clothes that monitor our health, from cars that can't crash to mobile technologies and cloud platforms that run our businesses. **Future Internet Architecture** therefore refers to the structure and organization of this all-pervasive system of things, services and infrastructure.

Multicast is a form of multipoint communication in which if N members participate in a communication session then the sender sends data to all other remaining members (R) of the session by transmitting only one data packet instead of *(R)* copies of the same packet in any single transmission instance. The packet is then be replicated by the routers to all of the *(R)* receivers.

Explicit multiunicast (XCAST)[3] is a variant of multicast aimed at delivery of data within small groups and can enhance scalability since it can support a very large number of small sized groups unlike the conventional multicast that can support only a small number of large groups. In the conventional multicast[1, 2], a packet carries a multicast address as a logical identifier of all group members. On the other hand, in XCAST, the source embeds the list of destinations within the XCAST packet header and then sends the packets to a router. The destination addresses are explicitly specified as a list of unicast addresses in the packet header. Section 2.3.3 presents a detailed explanation of how these addresses are embedded in an XCAST header,

how XCAST packets are processed by XCAST-aware routers and how the packets are eventually delivered to their respective destination hosts.

Multipoint communication technologies such as multicast and XCAST are important because they not only simplify collaboration for a group of users but also lead to efficient utilization of network resources. They are therefore commonly used in delivery of multimedia data. However, multimedia data are sensitive to factors such as packet loss, jitter and delay. Their delivery on the Internet must be done within acceptable constraints. Packet loss refers to the failure of data packets to arrive at their intended destination in a communication network. Jitter on the other hand refers to the deviation in or displacement of some aspect of the pulses of a digital signal from their assumed true periodicity. These factors affect quality of communication in the network and therefore have direct impact on the user experience during a communication session.

Traditionally, the concept of quality in networks meant that all network traffic was treated equally. The result was that all network traffic received the network's *best effort*, with no guarantees for reliability, delay, variation in delay, or other performance characteristics. With *best-effort* delivery service, a single bandwidth-intensive application can result in poor or unacceptable performance for all applications. The limits of delay, jitter and packet loss constitute factors often referred to as the *Quality of Service (QoS)*. An exhaustive definition of Quality of Service (QoS) is given in section 2.4. **QoS provisioning** refers to the concept of quality in which the requirements of some applications and users are more critical than others hence some traffics receive preferential treatment but others only get normal (*"best-effort"*) treatment. The ones that receive preferential treatment have a guarantee on the degree of availability (uptime), reliability, delay, jitter and other network characteristics required for acceptable user experience.

The challenge with XCAST is that while it can be used to deliver data to multiple

receivers which themselves might exhibit varying QoS requirements, the data to these numerours receivers is usually transported in the same XCAST packet. What makes QoS provisioning in XCAST a non-trivial task is the fact that XCAST QoS provisioning must ensure that numerous QoS levels can be supported within the same data packet. This probably explains why there is no significant previous work on XCAST QoS provisioning.

1.1 Motivation

This dissertation is motivated by two key facts. The first is that, despite XCAST having been proposed for use in delivery of multimedia data in small groups like video conferencing and online gaming, deploying XCAST in the Internet is not easy because the current commercial routers do not have inbuilt XCAST processing capabilities. The second is that no significant previous studies have been conducted on how QoS provisioning can be achieved in an XCAST network. The other motivation is the fact that research into the architecture of the Future Internet has picked up and it is therefore imperative that any QoS provisioning for XCAST transcends beyond the current Internet into the Future Internet architecture.

The work presented in this dissertation focuses on explicit multiunicast on IPv6 (XCAST6)[3, 4]. The dissertation presents a study on how XCAST6 can be incrementally deployed in the Internet using software routers we call *XCAST6 Routing Engine*[5, 6]. It then presents a comprehensive study of QoS provisioning in XCAST6 using Differentiated Services (DiffServ) architecture[7, 8] and further integrates XCAST6 into LISP[9] protocol which is premised on the Future Internet architecture where the Locators and Identifiers are in Separate address spaces.

Differentiated Services (DiffServ) architecture is a QoS provisioning mechanism that was specified by the Internet Engineering Task Force (IETF). DiffServ was

designed to provide the benefits of QoS without the scalability limitations. Instead of dealing with QoS on a per-flow basis, it aggregates traffic with similar QoS requirements into classes of traffic. A comprehensive background of DiffServ architecture is given in the appendices of this dissertation.

Location/Identifier Separation Protocol (LISP)[9] is a new protocol, currently under discussion at the IETF. It is aimed at solving the scalability issues of the current Internet by separation of network location and identity spaces. Therefore LISP is one of the protocols that are laying down the foundation for the Future Internet. More details will be explained in section 6.3.

The first part of this work focuses on deployment of XCAST6 in the real world. It proposes, implements and evaluates an out-of-the-box component referred to as XCAST6 Routing Engine[5, 6] that is connected to core routers and helps in processing of XCAST6 packets since current commercial routers are not XCAST6-aware. When the core router receives an XCAST6 packet, it forwards the packet to the XCAST6 Routing Engine for processing. Once an XCAST6 packet has been processed by the XCAST6 Routing Engine, the replicated packets are sent back to the core router for onward forwarding.

The second part of this work uses a network simulator (OMNeT++)[10, 11] to investigate QoS provisioning in an XCAST6 network. The second part implements both XCAST6 protocol and Differentiated Services architecture in OMNeT++. A simulation model is then built that uses both XCAST and DiffServ. The model is then used to conduct an extensive study on the QoS properties of XCAST6. As empirically illustrated, the approach provides efficient bandwidth utilization, better packet forwarding fairness between XCAST6 and non-XCAST6 traffic, lower traffic load on routers and elimination of *collusion attack(Good Neighbour Effect)*. The study then extends the model to investigate the feasibility of integrating XCAST6 with LISP, aimed at deployment in the Future Internet.

The remaining sections of this chapter discuss the research problem, solution and the contribution of the work presented in this dissertation. The chapter concludes by presenting the organization of the remaining chapters of this dissertation.

1.2 Research Problem

The goal of this dissertation is the *realization of multipoint communication over the Internet using XCAST*. This entails looking for a simple, cost-effective means of achieving the deployment of XCAST in the real world and also the provisioning of QoS to applications running on XCAST. It further aims at ensuring that XCAST can be deployed not only in the current Internet, but also in the Future Internet where the location and identity namespaces are separated. In order to make this goal manageable, we identified a number of sub-problems whose solution would lead to the solution of the main problem. These include:

1. To investigate how XCAST6 can be deployed in the real world in today's Internet. Real world deployment would allow for testing of XCAST6 performance with respect to various performance metrics on real routers and real applications.

2. To develop a platform in which large-scale testing of XCAST6 performance can be done with the least resources available and within a short time.

3. To investigate how Quality of Service mechanism can be integrated into XCAST6 without extensively compromising XCAST6 performance. This mechanism needs to put into consideration the fact that the current specification of XCAST6 header structure is already complex and is not supported by current commercially available routers. Integrating QoS into XCAST6 should therefore be done without further increasing the size of (or complicating) XCAST6 header.

4. To investigate how XCAST can be integrated with the Locator Identifier Separation Protocol (LISP) for both static LISP and LISP mobility using LISP Mobile Node (LISP-MN).

1.3 Solution

First, we study the implementation of the existing versions of XCAST6 and determine how they can be deployed into the real networking environment. This involves understanding why the existing XCAST6 implementations cannot be deployed in the commercially available routers. We then propose how to bridge this deployment gap by proposing a *"soft-router"* called *XCAST6 Routing Engine*.

Secondly, we seek to solve the next sub-problem by integrating XCAST6 into one of the popular network protocol simulators. This is because an XCAST6 simulator would help realize rapid and cost-effective investigation of various aspects of XCAST6. However the current commercial and open source network simulators do not support XCAST6 header structure and routing models. We therefore chose an open source simulator, OMNeT++[11, 10] and implemented XCAST6 protocol into its IPv6 stack. Having done this, we could then achieve a low-cost, rapid quantitative simulation of XCAST6 performance.

Third, we integrate Quality of Service awareness into XCAST6 by integrating XCAST6 protocol into Differentiated Services(DiffServ) Architecture. Integrating XCAST6 into DiffServ is challenging due to inherent architectural differences between XCAST6 and DiffServ when used for QoS provisioning. We therefore define algorithms for overcoming these architectural differences to ensure a smooth integration of the two. Our choice for the use of DiffServ for QoS provisioning in XCAST6 was informed by the realization that there is no significant previous attempt into handling XCAST6 QoS provisioning and also by the fact that XCAST6 header is

complex for the current routers therefore any QoS approach should not add another layer of complexity into XCAST6 processing.

1.4 Contribution

This work makes the following contributions:

1. *A mechanism to achieve gradual deployment of XCAST6 in the real world.* Current commercial routers do not have the inbuilt capabilities for processing of XCAST6 header. This is because XCAST6 header comprises of at least an IPv6 header tunneled within another IPv6 header and also a routing extension header where the destination addresses are embedded. Furthermore the commercial routers do not have inbuilt XCAST6 processing algorithm hence there is a need to have an "out-of-the-box" component that can understand and process the XCAST6 headers effectively before forwarding the processed XCAST6 packets to the conventional routers for forwarding to their respective destinations. To achieve this "out-of-the-box" solution, we implemented a soft-router we call XCAST6 Routing Engine on FreeBSD[12] operating system which is then connected to the core routers. When the core routers receive an XCAST6 packet, they simply forward the XCAST6 packet to the XCAST6 Routing Engine which does the XCAST6 processing of the header. Once the XCAST6 packet has been processed in the Routing Engine, the processed packets are sent back to the core router which then forwards them to their correct destinations.

2. *An analytical understanding of the effect of XCAST6 performance on routers.* We evaluated the performance of the XCAST6 Routing Engine so as to understand the impact of XCAST6 processing on routers. This evaluation was conducted by using DV video[13, 14, 15] traffic in an XCAST6 testbed and then analysing the behaviour of the XCAST6 Routing Engine with respect to

various performance metrics such as average router load, CPU and Memory utilization and also the internal system bevahiours such as the number of context switches that a router's CPU makes while processing XCAST6 packets.

3. *A quantitative simulation of XCAST6 performance characteristics.* In addition to the real world deployment using XCAST6 Routing Engine, we also realized that there is a need to have an environment where large scale testing of XCAST6 performance could be done. Since the existing network simulators do not have XCAST6 implementation, we chose to implement XCAST6 in OMNeT++. Using OMNeT++ simulator, we then conducted quantitative simulation of XCAST6 performance with respect to metrics such as stress, end-to-end delay, efficiency and packet processing overhead rate.

4. *An extensive investigation of XCAST QoS provisioning using Differentiated Services.* By way of simulation using OMNeT++, we have conducted an extensive study on QoS provisioning using Differentiated Services in XCAST6 networks. We've dubbed our implementation as *QS-XCAST6*. Owing to the fact that XCAST6 header is already complex for today's commercial routers, we chose a QoS provisioning method that will not add another layer of complexity onto an XCAST6 header. Hence we used Differentiated Services for QoS provisioning. However, Differentiated Services architecture was originally meant for unicast communication hence its integration into a multipoint communication environment poses a lot of challenges. We proposed changes to XCAST6 processing and also inclusion of DSCP classes within the IPv6 routing extension headers in an XCAST6 packet. We then compared our proposal to both conventional XCAST6 and unicast with respect to throughput, average per-hop delay, link utilization, router traffic load and fairness to other protocols. We further investigated the effect of varying the number of receivers(*group size*) on these pa-

rameters under both QS-XCAST6[16, 17] and XCAST6. Since XCAST6 (and by extension QS-XCAST6) is meant for Internet-wide usage, we also investigated the effect of network scale on using our approach. We found out that our proposal out-performs conventional XCAST6 in key areas such as efficient bandwidth utilization and very fair packet forwarding behaviour to other non-XCAST traffic.

5. *An economical way of utilizing bandwidth resources while provisioning QoS for real-time traffic.* Our work also reveals that using the QoS aware XCAST6 (QS-XCAST6)[16, 17] can be important in ensuring bandwidth economy by taking advantage of the various bandwidth allocation thresholds for the different DSCP classes. This results in a phenomenon we call QS-XCAST6 Gain (QXG) in chapter 5 and the lower the ratio of realtime to non-realtime traffic, the higher this gain for a network service provider.

6. *A simple approach for mitigating collusion attacks that naive deployment of XCAST6 can be vulnerable to.* Networks in which different QoS levels of services are offered and prices vary with the kind of Service Level Agreements(SLAs) are at times called "pay-per-service" networks. Collusion attack is a vulnerability problem in those "pay-per-service" networks whereby a subset of clients collude to exploit vulnerabilities in the network implementation to pay substantially less amounts for the best QoS services offered, leaving other clients and the service provider unaware of the unfolding collusion. The other unaware users actually end up to be the ones soldiering the cost of services enjoyed by the colluding clients hence they are usually referred to as *"Good Neighbours"* and the attack is also at times called *"Good Neighbour Effect (GNE)"*. In DiffServ networks, GNE occurs when clients can exploit *"DSCP confusion"* problem in routers; a phenomenon in which the network routers queue low priority DSCP

packets in the queues of higher priority DSCP classes. QS-XCAST6 solves this problem by dynamically re-writing the DSCP field of all the copies of XCAST6 packets at every replication point and in each copy, it puts the best DSCP value that satisfies all clients whose destinations are embedded in the XCAST6 header without giving the packet unduly higher DSCP value.

7. *Integration of XCAST6 with LISP Protocol that opens up research of XCAST6 for the future Internet.* The final part of this work investigates on how XCAST6 can be used in LISP networks. This is an important aspect because LISP protocol is one of the protocols that is expected to run the future Internet. We looked at three possible approaches that can be used to integrate LISP and XCAST6. We looked at the pros and cons of these approaches and settled on one of them which we implement in OMNeT++ for evaluation purposes. We further work on this using LISP Mobility architecture (through LISP Moblie Node) to integrate XCAST6 Mobility and LISP-MN.

1.5 Dissertation Structure

The dissertation is organized into two parts. Part I deals with the implementation of XCAST6 in the real world while part II deals with study conducted on a simulation environment. The structure of the remaining sections of this dissertation is organized as shown in figure 1.1. The chapters are marked with numbers within the ellipses. Information in the dotted boxes indicate methods that were used to obtain data that is presented in the corresponding section of the dissertation.

Chapter two presents the background study and some of the related work. We elaborate on the concept of Quality of Service and multipoint communication. The chapter further delves into the Differentiated Services architecture and gives a comparative study of previous attempts that have been made to provide Quality of Service

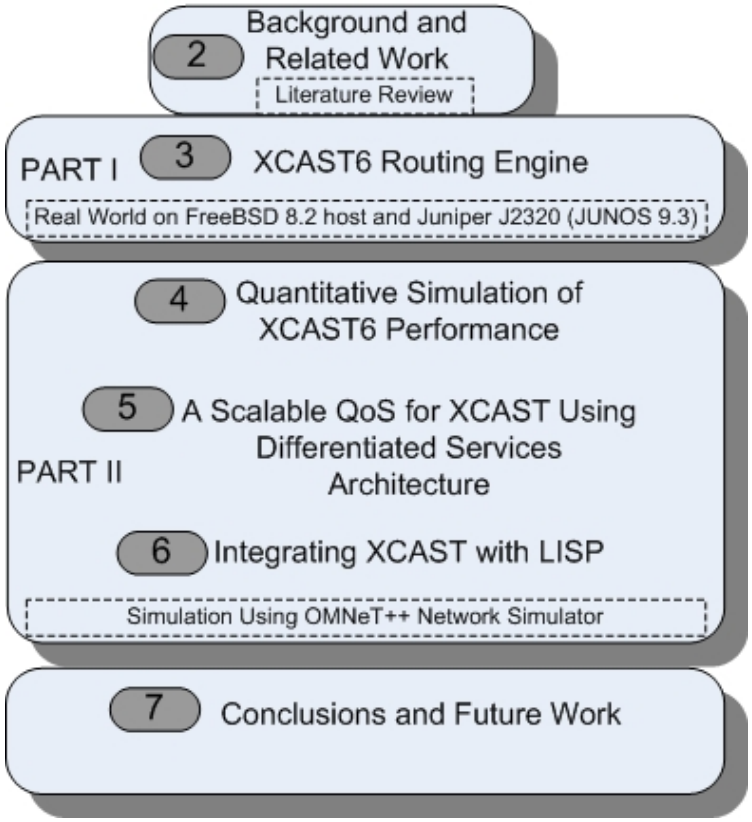


Figure 1.1: Structure of the dissertation

in multipoint communication environment using Differentiated Services. We study up to ten different protocols and give a classification of the various approaches that each of those protocols have attempted and explain why such approaches cannot be feasible for XCAST integration.

Chapter three deals with the issue of deployment of XCAST6 in the real world. We investigate the challenges faced with the current implementation of XCAST6 and propose an XCAST6 Routing Engine to help in real world deployment of XCAST6. XCAST6 Routing Engine is a software router implemented in FreeBSD operating system which we then use together with a Juniper router to explore how this concept can be used for gradual deployment of XCAST6. We also conduct performance evaluation of the XCAST6 Routing Engine with respect to a number of metrics in order to understand how XCAST6 would impact the performance of commercial routers.

Chapter four delves on the Quantitative simulation of XCAST6 performance. First it shows how an XCAST6 protocol is implemented in the open source of network simulator, OMNeT++. Specifically, XCAST6 is implemented in the INET Framework[18] of OMNeT++. Chapter four explains the changes made onto the INET Framework to allow for integration of XCAST6 into the IPv6 stack of INET framework. The implementation is then tested and used to quantitatively investigate performance aspects of XCAST6.

Chapter five focuses on the scalable QoS-aware XCAST6(QS-XCAST6). First it extends the work on chapter four by integrating the Differentiated Services architecture into OMNeT++. Since OMNeT++ currently has only a single classifier, our work introduces numerous classes that specifically handle DiffServ functionality into OMNeT++. We then use the platform to conduct an extensive study on QoS provisioning in XCAST6 using Differentiated Services. We compare the QS-XCAST6 with the typical XCAST6 when used to deliver QoS sensitive traffic such as IPTV traffic. This chapter also focuses on the simplification of integration of XCAST6 with DiffServ and further shows how QS-XCAST6 solves the problem of collusion attack which naive application of XCAST6 could be vulnerable to. We empirically show that QS-XCAST6 is efficient in bandwidth utilization and also offers a very good *"packet forwarding fairness"* to other non-realtime protocols when used in QoS provisioning.

Chapter six on the other hand delves on how XCAST6 can be implemented onto Locator Identifier Separation Protocol (LISP). This chapter also extends the work of chapter three in that we implemented the LISP functionality onto OMNeT++. We then use the simulator to investigate various options of integrating XCAST6 and LISP.

Chapter seven concludes the dissertation and looks at some possible future research directions in this area.

Chapter 2

Background and Related Work

2.1 Overview

Realizing multipoint communication over the Internet involves a number of factors. These include ensuring simple, cost-effective solutions for deploying multipoint communication technologies in the Internet. It also involves identifying how the multipoint communication networks can show predictable behaviour in terms of availability(uptime), bandwidth (throughput) and limited delay; usually known as *"Quality of Service (QoS)"*. A simple definition of multipoint communication is the "transfer of information among a set of receivers using distinct one-to-many channels". There are a lot of technologies that can be applied to achieve multipoint communication. Arguably, the most common multipoint communication technology is *"multicast"*. This chapter defines multicast, shows the various types of multicast and then explains the difference between multicast and XCAST. The chapter further explores some of the challenges that have beleaguered the deployment of both multicast and XCAST. Finally it gives a background on how Quality of Service (QoS) can be guaranteed in both XCAST and multicast networks.

2.2 Introduction

The need for group and collaborative applications on the Internet saw the emergence of multipoint communication technologies such as Internet Standard Multicast[1, 19, 20, 21] to provide efficient packet delivery thereby minimizing the consumption of bandwidth. As the number of services and users on the Internet increases, so is their diversity in terms of bandwidth, delay and jitter requirements. These requirements are dependent on several factors like the media processing capabilities of the user devices, the amount of bandwidth the users are capable of paying for and the contract agreements with the ISPs. Therefore the Internet needs to employ mechanisms of ensuring that each user's requirements are appropriately met. This implies provisioning of a varying levels of Quality of Service (QoS) to the users and applications.

Quality of Service(QoS) refers to a collection of networking technologies and techniques, the goal of which is to provide a guarantee on the ability of a network to deliver predictable results satisfying the users' requirements. Specifically this relates to network performance in terms of availability (or uptime), bandwidth (throughput), delay(latency), packet loss and error rate. QoS provisioning involves not only ensuring that the above performance metrics are within a user's acceptable limits but also prioritization of network traffic to serve users differently depending of their needs.

QoS is especially important for transfer of multimedia data in Internet applications such as IPTV, VoIP, real-time Internet games and even microprocessors[22]. QoS applies to nearly all aspects of data communication including service oriented architectures in IP networks[23, 24, 25, 26], Ad hoc networks[27], optical networks[28], scheduling[29] and routing in general[30, 31].

The tremendous growth of group communications and quality of service (QoS)-aware applications over the Internet has accelerated the need for scalable and efficient network support[32, 2, 33, 34, 35]. Therefore enforcing a single QoS provisioning strategy might not work. This becomes even harder in multipoint communication

where data to multiple recipients is sent out in only a single packet.

In the sections that follow, we give a detailed background of both multicast and XCAST technologies. We then explore the challenges facing bothe their deployment in the Internet and QoS provisioning for application running on top of them. We further enumerate attempts that have been made to overcome these challenges under multicast and then delve on why the multicast-oriented approaches for both deployment and QOS provisioning cannot be applied for XCAST. How these can be realized for XCAST are then discussed in details in later chapters of this dissertation.

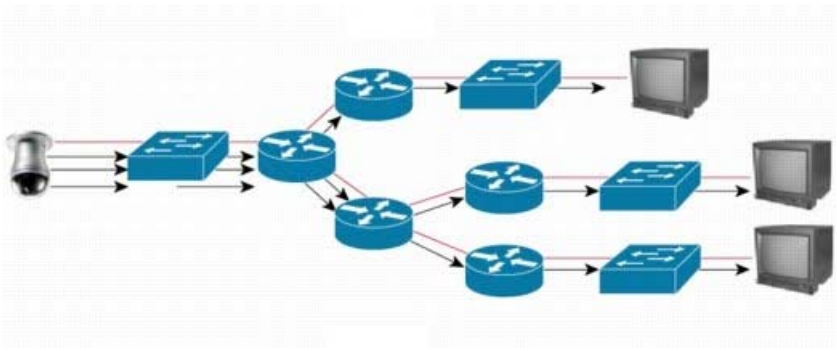
2.3 Multipoint communication

Multipoint communication refers to the transfer of information among a set of participants through distinct one-to-many communication channels. The channels (paths) are from a single source (the sender) to multiple end points (receivers). This kind of communication can be achieved mostly in two ways:

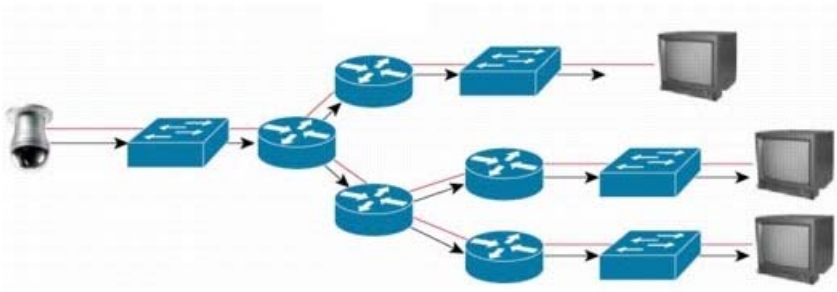
i. **Repeatedly unicasting**: In which the source sends out copies of the same packet repeatedly to all the receivers as shown in figure 2.1(a). This can be done possibly in a round-robin pattern amongst all the participants in the session.

ii. **Through multicast technology**: In this case the source sends out only one packet as shown in figure 2.1(b). However the packet gets replicated within the routers in the network so that each participant receives a copy.

Multicast technology is efficient in terms of bandwidth utilization because only one packet is sent out. Figure 2.2 gives an illustration of multicast-based technologies in multipoint communication. It first classifies the technologies based on the type of group sizes they support and then on how they are implemented; whether at router level or at application level.



(a). Multiple unicasts of the same packet.



(b). Multicast. Only one packet is sent out.

Figure 2.1: Multiple unicast versus multicast.

IP multicast, usually considered the Internet Standard Multicast has several protocol implementations. Application Layer Multicast (ALM) is usually implemented in overlay networks and a number of protocols and approaches have been proposed. XCAST on the other hand is for small groups and has a unique address encoding features that makes it fall in its own class. Figure 2.2 is by no means exhaustive in illustrating the multicast variants. The subsequent sections give an elaborate description of multicast technology. We discuss the key concepts in multicat technology and the various multicast QoS provisioning approaches.

2.3.1 Multicast

Multicast, the ability to efficiently send data to a group of destinations was first proposed by David Cheriton and Stephen Deering in 1985. They proposed what became known as the Host Group Multicast model[2]. Multicast[1, 2], is a form of

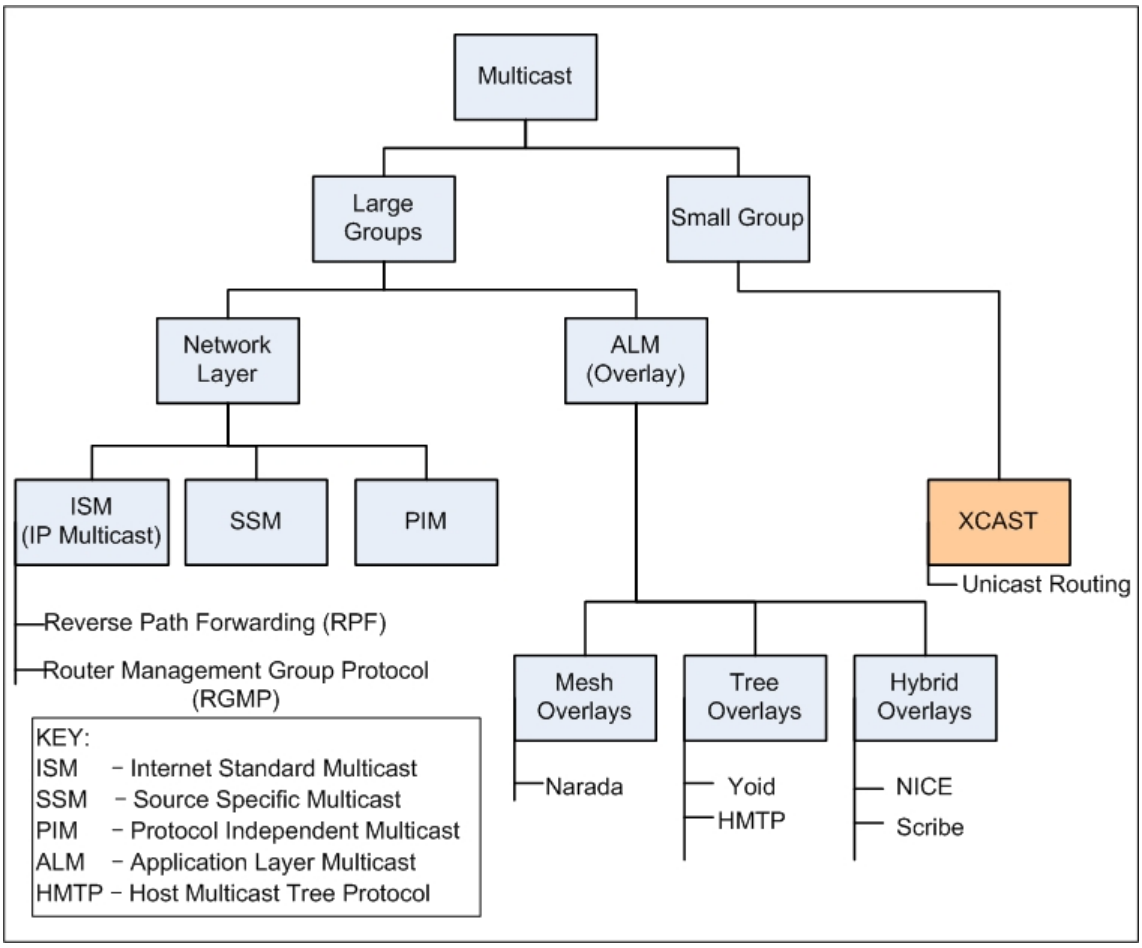


Figure 2.2: Multicast technology map

multipoint communication in which if N members participate in a communication session then the sender sends data to all other remaining members (R) of the session by transmitting only one data packet instead of *(R)* copies of the same packet in any single transmission instance. The packet is then be replicated by the routers to all of the *(R)* receivers. However replication takes place only at points in the network routers where branching occurs in the delivery path. See figure 2.1(b).

Multicast technology is itself divided into several variants. We look at the *"Host Group Model"* of IP multicast in the next section and briefly mention other multicast variants too.

Host Group Model

The difference between multicasting and separately unicasting data to several destinations is best explained using the *"host group model"*[2]. In this model, a host group is a set of network entities sharing a common identifying address, called *a multicast address*. All the entities receive any data packets addressed to this multicast address by the senders (sources). The senders may or may not be members of the same group and might be having no knowledge of the groups's membership. This means that from the sender's point of view, this model reduces the multicast service interface to a unicast one. Thus, the multicast model was proposed to reduce the many unicast connections into a multicast tree for a group of receivers. A *"group"* can therefore be loosely defined as a set of entities participating in communication within a specified session. A *"member"* of a group refers to an individual entity participating in the communication session.

Multicast group life cycle

Usually the lifetime of a multicast group comprises of four main steps as summarized in figure 2.3. These include:

i. **Multicast group (session) creation**: This is the first step in initiation of a multicast session. It involves assigning a unique address to the multicast group such that the data of one group does not conflict with the other groups. Both multicast groups (sessions) and addresses have associated lifetimes.

ii. **Multicast tree construction with resource reservation**: After creating the groups, the next step is the construction of a multicast distribution tree. The tree begins from the root (usually the source node) and spans a number of nodes to the receivers, which mostly form the leaves. This is usually a complex process and involves application of a number of multicast tree construction algorithms.

iii. **Data transmission**: After successfully executing the two previous steps, data can be transmitted in the multicast session. Other activities such as session control, failure handling and tree re-arrangement (for QoS assurance) also occur in this stage.

iv. **Multicast session teardown**: When the session's lifetime elapses, the source begins the teardown process. Session teardown involves releasing the resources reserved for the session along all of the links of the multicast tree and purging all session-specific routing table entries. The multicast group address is then released and this marks the completion of the teardown process.

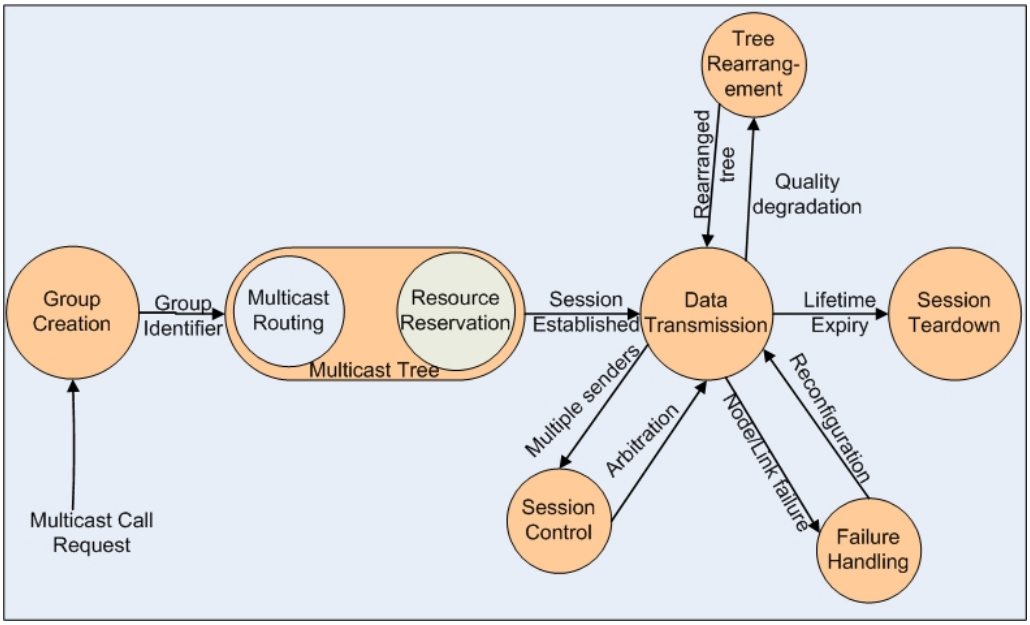


Figure 2.3: The life cycle of a multicast session

Types of Multicast groups

Multicast groups can be classified based on a number of characteristics they exhibit. Usually, the behaviour of a multicast group is unrestricted and can be looked at from multiple perspectives. Groups may have local (e.g. LAN) or global (WAN)

membership. They may be transient or persistent in time and may as well have constant or varying membership. Based on these, multicast groups can be classified as:

- **Dense**: These are groups which have members in most of the links or subnets in the network.
- **Sparse**: Groups which have members only in a small number of widely separated links or subnets.
- **Open**: The sender (source) needs not to be a member of the group before they can send to other group members.
- **closed**: Groups which allow only members to send to the group.
- **Permanent**: Groups which exist for a longer duration of time. Usually assumed to last *"forever"*.
- **Transient**: Groups that exist only for a short period of time.
- **Static**: Groups whose membership remain constant in time;
- **Dynamic**: Groups which allow members to join/leave at anytime in the course of the group's lifetime.

Other variants of multicast protocol include: Internet Standard Multicast (ISM)[1, 19, 20, 21], Application Layer Multicast(ALM)[36, 37, 38], Protocol Independent Multicast(PIM)[39, 40] and Source Specific Multicast(SSM)[41] among others. ISM uses a range of protocols such as Reverse Path Forwarding(RPF)[42, 43], Router Group Management Protocol (RGMP)[44], Bidirectional PIM (Bidir-PIM)[45] and Pragmatic General Multicast (PGM)[46] to control flow of packets within a network.

2.3.2 Multicast Deployment Issues

Deployment of multicast at router level in the Internet has been plagued by numerous challenges among them:

i. **State scalability problem**: Multicast routing state does not scale well. Multicast routing protocols exchange messages that create state for each source, group pair, *"commonly denoted as (S,G) pair"* in all the routers that are part of the point-to-multipoint tree. This can be viewed as "per flow" signaling that creates multicast connection state, possibly yielding huge multicast forwarding tables (MFTs). Routers need to keep states "per-group" and in some multicast variants, "per-group, per-source" information must be maintained. A large number of groups results in a large number of state information to be stored by the routers. This translates to larger memory requirements and a slower packet forwarding rate.

ii. **Per-session signaling**: For every multicast session, a lot of signaling has to be done in order to know members of the group who are still actively participating in the session. For large groups, this can lead to exchange and maintenance of a lot of information that eventually may lead to router overload.

iii. **Complex tree construction algorithms**: Constructing optimal multicast tree is complex and involves application of complex algorithms and tree data structures. For a static multicast group, determining the optimal tree has been modeled as *Steiner tree problem*[47] which is known to be NP-complete.

iv. **Difficulty in QoS provisioning**: QoS guarantee in multicast has been found to be difficult[48]. This is especially true in instances where different members of a group require different levels of QoS. This is called *QoS heterogeneity*. Implying that a single QoS provisioning mechanism can hardly satisfy all members with heterogeneous QoS requirements.

v. **Multicast address scope**: Multicast group address must be unique in the scope where multicast is deployed. In most cases multicast applications are required to be used all over the Internet. This implies the scope is global. It is difficult to guarantee a globally unique multicast address since the multicast address ranges are well known and any organization can pick an address from that range.

vi. **Destination unawareness**: When a multicast packet arrives in a router, the router can determine the next hops for the packet, but knows neither the ultimate destinations of the packet nor how many times the packet will be duplicated later on in the network. This complicates QoS provisioning, especially when used with DiffServ scheme where resource allocation per link is of ultimate importance. Usually this leads to a problem called *Neglected Reservation Subtree ("NRS")*[49]. It also complicates the security, accounting and policy functions.

vii. **Source advertisement**: Multicast routing protocols provide a mechanism by which members get 'connected' to the sources for a certain group without knowing the sources themselves. In sparse-mode protocols[50, 51], this is achieved by having a core node, which needs to be advertised in the complete domain. On the other hand, in dense-mode protocols[52] this is achieved by a "flood and prune" mechanism. Both approaches raise additional scalability issues.

To solve the scalability problems of multicast, a new variant of XCAST targeting groups with very few members was proposed. This is called explicit multiunicast (XCAST).

2.3.3 Explicit Multiunicast (XCAST)

The 1998 IAB Routing Workshop[53] came to the conclusion that "providing for many groups of small conferences (a small number of widely dispersed people) with global topological scope scales badly given the conventional multicast model". This led to

an implication that two kinds of multicast seem to be important; a broadcast-like multicast that sends data to a very large number of destinations and a "narrowcast" multicast that sends data to a fairly small group[54]. An example of the first is the audio and video multicasting of a presentation to all employees in a corporate Intranet. An example of the second is a videoconference involving three or four parties. It seems prudent to use different mechanisms for these two cases because a "one size fits all" protocol will be unable to meet the requirements of all applications.

Explicit Multiunicast (XCAST), is a multicast scheme with complementary scaling properties to those of the conventional IP multicast; XCAST supports a very large number of small multicast sessions while IP multicast only scales well in terms of members in a group but not for a large number of groups. XCAST achieves this by explicitly encoding the list of destinations in the data packets, instead of using a multicast group address.

XCAST takes advantage of one of the fundamental tenets of the Internet "philosophy", namely, that one should move complexity to the edges of the network and keep the core network simple. This is the principle that guided the design of IP and TCP and it is the principle that has made the incredible growth of the Internet possible.

In XCAST, the source embeds the list of destination in the XCAST header and then sends the packets to a router. The destination addresses are explicitly specified as a list of unicast addresses in the packet header. A bitmap is then used to keep track of the set of addresses to which data has been sent and the other set to which data has not been sent. Each router along the way parses the header, partitions the destinations based on each destination's next hop and forwards a packet with an appropriate header to each of the next hops. When there is only one destination left, the XCAST packet can be converted into a normal unicast packet which can be unicasted along the remaining portion of the route. This is called XCAST to Unicast (X2U).

2.3.4 History of XCAST

While active research in XCAST seems to have picked in late 1990s, it is interesting to note that the idea of XCAST had been in existence for sometime. However the three groups[54, 55, 56] that independently re-invented it actually did not know this. The first proposal of the multicast concept in the Internet community, by Lorenzo Aguilar in his 1984 SIGCOMM paper[57] proposed the use of an explicit list of destinations. At about the same time (precisely 1985), David Cheriton and Stephen Deering developed the concept of Host Group model[2]. Since Aguilar's proposal seemed to have serious scaling problems, the Host Group model[2] was adopted. This was even understandable considering the fact that the Internet of 1985 must have been extremely smaller than that of today.

In late 1990s, Rick Boivie and Nancy Weldman of the IBM Watson Research Center began research on *"Small Group Multicast (SGM)"*[54]: a multicast scheme in which the source node keeps track of the destinations that it wants to send packets to and creates packet headers that contain the list of destination addresses. At about the same time, Dirk Ooms and Wim Livens of Alcatel Corporate Research Center in Belgium were working on a similar concept which they called *"Connectionless Multicast"*[55]. Yet again within the same time span, extensive research on IPv6 was going on and Yuji Imai of Fujitsu laboratory in Japan began to work on how to realize multicast on IPv6 for small groups. His work was called *"Multiple Destination option on IPv6(MDO6)"*[56].

These three independent pieces of work were presented to the IRTF and formed part of the Sclable Adaptive Multicast Research Group (SAMRG) working group. The harmonization of these pieces of work resulted in the new name "XCAST". A lot of research was conducted and a series of Internet Drafts written which later in the 2007 resulted into the XCAST RFC document, RFC5058[3]. The research has continued in XCAST since then. XCAST concept and options as described in the

RFC document[3] forms the basis of version 1.0 of XCAST on IPv6. This has been extended by proposals in the draft document[4] to form the basis of XCAST6 version 2.0. To address efficient routing of XCAST6, we have also written an Internet draft document[5] proposing the XCAST6 Routing Engine. This dissertation looks at not only the XCAST6 Routing Engine but also goes further into the QoS provisioning in XCAST and the feasibility of integrating XCAST with LISP for use in the future Internet.

2.3.5 Motivation behind XCAST

XCAST was founded on the philosophy that, routers in the core network should not have to keep track of a large number of individual multicast flows. This is one of the fundamental tenets that have allowed for the expansive growth the Internet has seen since its invention; that is, "one should move complexity to the edges of the network and keep the middle of the network simple". It is the same principle that guided the design of TCP/IP.

One advantage of multicast schemes is that they can be used to minimize bandwidth consumption. XCAST also can be used to minimize bandwidth consumption for "small groups". But it has additional merits. XCAST eliminates the per-session signaling and per-session state information of traditional IP multicast schemes. This allows XCAST to support very large numbers of multicast sessions. This scalability is important since it enables important classes of applications such as IP telephony, videoconferencing, collaborative applications, networked games, etc., where there are typically very large numbers of small multicast groups. Compared to traditional IP multicast, XCAST has the following advantages:

i. **No multicast address allocation required**. XCAST uses unicast routing table for transfer of data packets.

ii. **No state maintenance**. Routers do not have to maintain state per session

(or per channel). This makes XCAST very scalable in terms of the number of sessions that can be supported since the nodes in the network do not need to disseminate or store any multicast routing information for these sessions.

iii. **No single point of failure**: Unlike the shared tree schemes, XCAST does not depend on a "core node" whose failure impacts the entire network. XCAST also minimizes network latency and maximizes network "efficiency".

iv. **No multicast protocols**. XCAST does not need multicast routing protocols (neither intra- nor inter-domain). XCAST packets always take the "right" path as determined by the ordinary unicast routing protocols.

v. **Heterogeneous receivers**. In an XCAST packet, besides the list of destinations, a list of Diffserv Code Points (DSCPs) could also (optionally) be contained. While traditional IP multicast protocols have to create separate groups for each service class, XCAST incorporates the possibility of having receivers with different service requirements within one multicast channel.

vi. **Reliability**: XCAST allows for a simple implementation of reliable protocols on top of it. This is because XCAST can easily address a subset of the original list of destinations to do a retransmission.

vii. **No need for Symmetric paths**. If a path between two nodes A and B is both the shortest path from A to B as well as the shortest path from B to A, the path is said to be symmetric. Traditional IP multicast routing protocols create non-shortest-path trees if paths are not symmetric. In XCAST routing is efficient with or without symmetric paths. Symmetric paths are therefore not required. It is expected that an increasing number of paths in the Internet will be asymmetric in the future due to traffic engineering and policy routing. Therefore the traditional IP multicast schemes will result in an increasing amount of suboptimal routing.

viii. **Automatic re-routes and failure recovery**. Route changes can occur due

to node failure or link congestion among other factors. XCAST reacts immediately to unicast route changes. In traditional IP multicast routing protocols, a communication between the unicast and the multicast routing protocol needs to be established. In many implementations, this is on a polling basis, yielding a slower reaction to, e.g., link failures. It may also take some time for traditional IP multicast routing protocols to recover from failures if there is a large number of groups that need to be fixed.

ix. **Unicast Traffic Engineering**. XCAST packets can make use of traffic-engineered unicast paths.

x. **Easy security and accounting**. In contrast with the Host Group Model, in XCAST all the sources know the members of the multicast channel, which gives the sources the means to, for example, reject certain members or count the traffic going to certain members quite easily. Not only a source, but also a border router is able to determine how many times a packet will be duplicated in its domain. It also becomes easier to restrict the number of senders or the bandwidth per sender.

2.3.6 Challenges in XCAST Deployment

Despite all the benefits of XCAST, its deployment has been faced with challenges stemming mainly from:

i. *Header structure*: XCAST header comprises of tunneled IP headers as shown in figure 2.4. This complicates its processing.

ii. *Processing algorithm*: Currently available commercial routers do not understand XCAST processing algorithm.

Header structure

An ordinary XCAST6 header comprises of two IPv6 headers, an IPv6 routing extension header and a transport header (usually UDP for multimedia applications). The list of destinations is embedded within the IPv6 routing extension header. A simple header of XCAST on IPv6 (XCAST6) that uses UDP as a transport protocol is shown in figure 2.4.

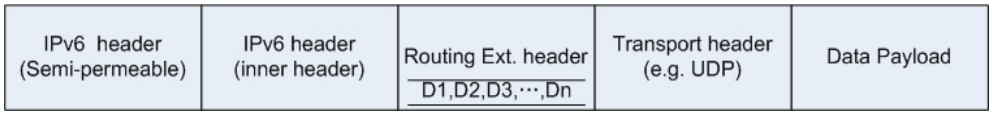
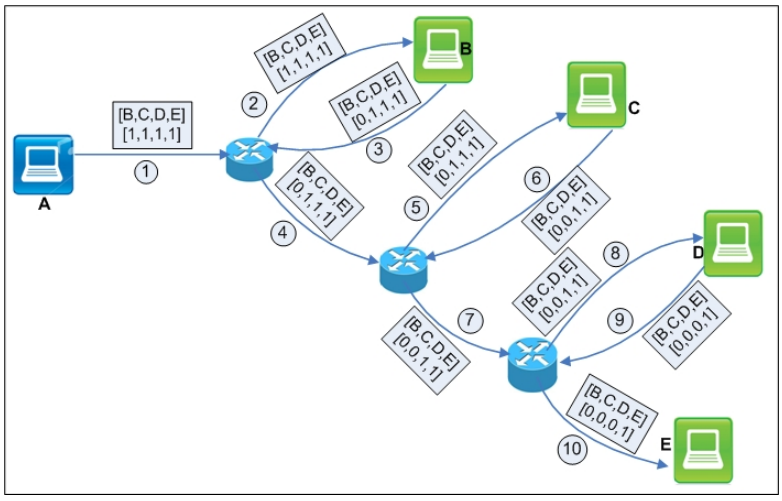


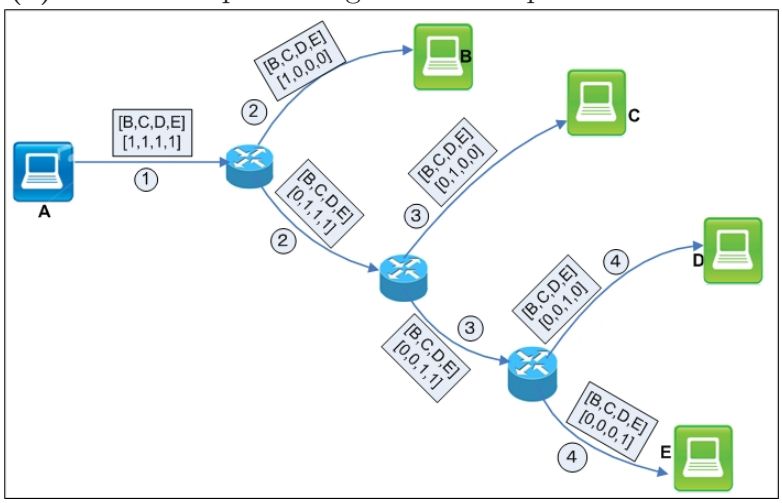
Figure 2.4: XCAST6 version 2.0 header structure

Therefore XCAST6 version 2.0 uses an *"IP within IP tunneling"* to enable routing of XCAST6 packets within the network. Tunneling is used so that nodes that are not XCAST6 capable can simply forward the packet as if it were an ordinary IPv6 packet based on the information they read on the outer IPv6 (semi-permeable tunneling) header.

Since the existing commercial routers are not XCAST-aware, they will simply pass the XCAST6 packets without proper processing. This leads to XCAST6 packets being processed only at the end hosts which are XCAST aware. Such a scenario leads to inefficient XCAST processing as shown in figure 2.5(a). In figure 2.5(a), XCAST6 packets are delivered in a cascaded manner from one end host to another instead of being partitioned in the branching routers as required. This leads to several hops (10 hops for node E)and the resulting delay is likely to affect communication quality considerably. Proper processing in routers is shown in figure 2.5(b) in which packets are replicated and forwarded almost concurrently from each branching router.



(a). Inefficient processing of XCAST packets at end-hosts.



(b). Proper processing of XCAST packets in routers.

Figure 2.5: Cascaded delivery of XCAST6 packets.

Processing algorithm

XCAST aware routers and hosts know that XCAST destinations are embedded within the IP packet and there are bitmaps that determine how copies of the packet are to be delivered. This implies that XCAST processing is usually different from that of ordinary packets. When an XCAST packet is received by an XCAST aware node, the node applies the XCAST processing algorithm, whereby the node:

i. Performs a route table lookup to find the next hops for each of the destinations listed in the XCAST packet

ii. Partitions the set of destinations based on their next hops

iii. Replicates the packet so that there is only one copy of the packet for each of the next hops that had been determined in the previous steps.

iv. Modifies the list of destinations in each of the copies so that the list in the copy for a given next hop includes just the destinations that ought to be routed through that next hop.

v. Sends the modified copies of the packet to each of the next hops.

vi. If there is only one destination left for a particular next hop, the router can further take an optimization step in which the XCAST packet is sent as a standard unicast packet in a process called XCAST to Unicast (X2U).

XCAST deployment in the Internet is therefore seriously jolted by the fact that the routers currently installed in the Internet are not XCAST capable and cannot follow the XCAST processing algorithm. Therefore a need arises for finding mechanisms on how this efficient XCAST6 processing can be achieved so as to realize effective deployment of XCAST6 on the Internet. This forms the basis of reseearch in chapter 3 of this dissertation.

2.4 QoS: Quality of Service

Quality of Service (QoS) is a set of technologies for managing network traffic in a cost effective manner to enhance user experience. QoS technologies enable the measurement of bandwidth utilization, detection of changing network conditions (such as congestion or availability of bandwidth), and prioritization or throttling of network traffic. For example, QoS technologies can be applied to prioritize traffic for latency-sensitive applications (such as voice or video) and to control the impact of latency-insensitive traffic (such as bulk data transfers). The goal of QoS is therefore to provide

preferential delivery service for the applications that need it by ensuring sufficient bandwidth, controlling latency and jitter, and reducing data loss.

The Internet Engineering Task Force (IETF) defines two major models for QoS on IP-based networks: Integrated Services (Intserv)[58] and Differentiated Services (Diffserv)[7, 8]. These models encompass several categories of mechanisms that provide preferential treatment to specified traffic including:

i. *Admission control*: Determines which applications and users are entitled to network resources. These mechanisms specify how, when, and by whom network resources on a network segment (subnet) can be used.

ii. *Traffic control*: Regulates data flows by classifying, scheduling, and marking packets based on priority and by shaping traffic (smoothing bursts of traffic by limiting the rate of flow). Traffic control mechanisms segregate traffic into service classes and control delivery to the network. The service class assigned to a traffic flow determines the QoS treatment the traffic receives.

The Intserv model integrates resource reservation and traffic control mechanisms to support special handling of individual traffic flows. The Diffserv model on the other hand uses traffic control to support special handling of aggregated traffic flows.

2.4.1 Why Qos?

QoS provides the following benefits:

i. Gives administrators control over network resources and allows them to manage the network from a business, rather than a technical, perspective.

ii. Ensures that time-sensitive and mission-critical applications have the resources they require, while allowing other applications access to the network.

iii. Improves user experience.

iv. Reduces costs by using existing resources efficiently, thereby delaying or reducing the need for expansion or upgrades.

2.5 QoS in Conventional Multicast and XCAST

Group based applications not only require scalable and efficient network support but also have stringent QoS requirements in terms of throughput, end-to-end delay, delay jitter, and error rate. QoS provisioning therefore plays an important role in realizing an acceptable multipoint communication scheme. There have been approaches applied to QoS provisioning in multicast but due to the inherent differences between multicast and XCAST in how members of a session are handled, these multicast-oriented approaches cannot fit well for XCAST. In the sections that follow, we look at how QoS is handled in both multicast and XCAST.

2.5.1 QoS in Conventional Multicast

Multicast technologies can reserve resources along the multicast tree as a way of ensuring that QoS requirements are met. Resource reservation can thus be achieved by protocols such as RSVP[59]. However, such protocols do not determine the path hence a poor path choice will still impact negatively on the QoS of the multicast group. It is the responsibility of the multicast routing protocol to determine that path. QoS provisioning in multicast can therefore be achieved through the following approaches:

i. QoS-aware routing.

ii. Tree rearrangement.

iii. Core/tree migration.

iv. DiffServ-aware multicasting.

QoS-aware routing

Multicast routing protocols can be classified as either *source-based protocols* or *center-based protocols*. The source-based approach uses a shortest path tree (SPT) rooted at the sender/source. In SPT, each branch of the tree is the shortest path from the sender to each group member. The shortest path (in hops) is usually the shortest delay path. Hence the receivers in the multicast tree typically receive excellent QoS. In the converse, source-based trees introduce scalability problems for large networks. This is because each individual receiver must have a shortest path from source to receiver. The shortest path provides additional performance *(better QoS)* at the cost of network resources.

The center-based protocols are also known as shared-tree protocols. They construct a multicast tree spanning the members whose root is the center or core node. These protocols are highly suitable for sparse groups and scalable for large networks. However they provide excellent bandwidth conservation at the cost of QoS to the receivers.

Tree rearrangement

The tree rearrangement mechanism is a means to achieve balance between the goal of reduction of the cost of QoS provisioning and the disruption of a multicast session. The group dynamic events of join and leave can disrupt an ongoing multicast session. However it is important to ensure that the multicast tree after member join/leave will still remain near optimal and satisfy the QoS requirements of all on-tree receivers. One way to handle dynamic member join/leave is by reconstructing the tree every time a member joins or leaves the session. This involves migration of on-tree nodes to the new tree, which may result in a large service disruption that may not be tolerable, especially by QoS multicast sessions. Another approach is by incrementally changing the multicast tree through the graft/prune mechanism. This incremental

change approach suffers because the quality (e.g., tree cost) of the tree maintained may deteriorate over time. Tree rearrangement takes into account two important and possibly contradicting goals of: cost reduction and minimization of service disruption.

Core/tree migration

In center-based multicast routing protocols, core selection is important because the location of the core influences the tree cost and delay. The quality of the tree based on the current core may deteriorate over time due to dynamic join and leave of members. The maintenance of a good-quality multicast tree requires online selection of a new core, online construction of a multicast tree based on the new core, and migration of the members from the old multicast tree to the new one. This way the quality of the session is maintained.

DiffServ-aware multicasting

Another approach to multicast QoS provisioning is to integrate multicast with the Differentiated Services (DiffServ) Architecture. This is usually not an easy task since DiffServ specifications were aimed at unicast communication. Additionally, DiffServ specification conflict with multicast QoS in a number of ways especially, the DiffServ requirement of a stateless core network yet multicast involves maintenance of state-information in routers. In section 2.6 we shall introduce key components of the DiffServ architecture. A detailed discussion of multicast-DiffServ integration challenges and approaches aimed at solving the challenges is presented in the appendices section of this dissertation.

2.5.2 QoS in XCAST

QoS research in XCAST is generally an unexplored field; characterized by lack of significant previous work. Through literature review, we have been able to find only

two pieces of work in XCAST QoS[60, 61]; both of which have been published by the same research team. In[60], Siregar et al, also acknowledge the lack of previous work in XCAST QoS research.

Nonetheless, the work by Siregar et al, published in[60] seems to only enumerate the available QoS routing techniques for unicast and multicast. They simply suggest that per-packet dynamic routing for unicast can be used in XCAST but they fail to show how this dynamic routing should be adopted for XCAST. On the other hand, in [61], they propose a new modified IPv6 extension header which they call *"IPv6 QoS header"* which should be used in holding a QoS value for an XCAST packet. The QoS value is to be calculated by routers based on the number of users underneath a router, total users requesting the video streams and the available priority levels. These values further depend on some probability assignment and prioritization based on the QoS levels. However, the work is not clear on how the priority and probability values are to be calculated and allocated.

The approach proposed in[61], therefore has two glaring challenges:

i. **Processing overhead costs**. Calculation of the probability and priority values incur additional processing cost. Moreover, the router also has to calculate the users beneath it who are participating in the current session and also the total number of users in the session. This increases computational complexity.

ii. **Longer packet header**. Adding a new extension header implies that the XCAST packet will have two IPv6 headers (inner and outer), two routing extension headers (for list of destinations and for QoS), transport header and a payload header as shown in figure 2.6. Compare this to the header shown in figure 2.4. The implication here is that even if all these headers are fit into one XCAST packet, the space left for payload data is too small. The packet will only carry a small amount of data which in itself might impact the QoS. Additionally, a longer header might be too large for the usual MTU in routers which leads to

IP Fragmentation problems.

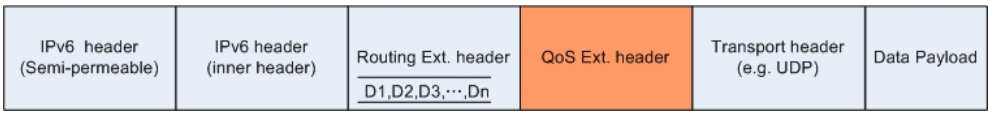


Figure 2.6: XCAST packet with "QoS extension header".

In chapter 5, we propose to leave the XCAST header as specified in the XCAST RFC[3] but use DiffServ architecture for QoS provisioning. This proves to be simpler and adds no severe processing overheads to the routers.

2.6 DiffServ: Differentiated Services Architecture

DiffServ architecture[7, 8, 62, 63] was specified by the IETF and defines an architecture for implementing scalable service differentiation in the Internet. A comprehensive background of DiffServ architecture is given in the appendices of this dissertation. The primary goal of DiffServ was to provide the benefits of QoS without the scalability limitations of IntServ[58]. Rather than attempting to deal with QoS on a per-flow basis, the DiffServ model aggregates traffic with similar QoS requirements into classes of traffic. DiffServ does not maintain per-flow information, thus eliminating the two key weaknesses of the IntServ model, the setup and the maintenance of per-flow state information. In addition, DiffServ focuses on domain-wise (AS) behavior rather than end-to-end behavior in order to expedite the deployment of QoS services.

A *"service"* defines some significant characteristics of packet transmission across a set of one or more paths within a network. These characteristics may be specified in terms of *throughput, delay, jitter, and loss* or may otherwise be specified in terms of some *relative priority of access* to network resources. Service differentiation is desired to accommodate heterogeneous application requirements and user expectations and also to permit differentiated pricing of Internet services.

DiffServ architecture is composed of a number of functional elements implemented in network nodes including:

i. A set of per-hop behaviors (PHB)[64, 65, 66].

ii. Packet classification functions

iii. Traffic conditioning functions such as metering, marking, shaping, and policing.

DiffServ architecture achieves scalability by implementing complex classification and conditioning functions[67, 68] only at network boundary nodes, and by applying per-hop behaviors to aggregates of traffic which have been appropriately marked[69, 70, 71] using the DS field in the IP headers (*Traffic Class in IPv6*). Per-hop behaviors are defined to permit a reasonably granular means of allocating buffer and bandwidth resources at each node among competing traffic streams.

A key requirement of the DiffServ architecture is that *"per-application flow"* or *"per-customer forwarding state"* need not be maintained within the core of the network. In the DiffServ model, traffic entering a network is classified and possibly conditioned at the boundaries of the network and assigned to different behavior aggregates. Each behavior aggregate is identified by a single DiffServ Code Point(DSCP). Within the core of the network, packets are forwarded according to the per-hop behavior associated with the DSCP.

Routers in a DiffServ domain are therefore categorized into two classes, namely *core routers* and *edge routers*. In contrast to core-routers, edge routers maintain *"per-application"* or *"per-customer forwarding state"* information. This is a fundamental conflict with most multipoint communication protocols in which the forwarding-state information is maintained in routers including the core routers. Key application areas of the DiffServ architecture include SLA-based Internet service pricing[72], traffic engineering[73, 74, 75] and queueing[76, 77, 78]. Differentiated Services have also been used in optical networking[79, 80, 81], general dynamic QoS adaptation[82, 83]

and in routing domains[84, 85].

2.7 LISP: Locator/Identifier Separation Protocol

The Location/Identifier Separation Protocol[9] has been recently proposed to provide an incrementally deployable solution to separation of network location and identity spaces anticipated in the future Internet. In this section we give a brief overview of LISP protocol while more details will be explained in section 6.3. LISP considers two different types of addresses: *Endpoint Identifiers (EIDs)* and *Routing Locators (RLOCs)*. EIDs identify hosts, and are assigned independently of the network topology while RLOCs identify network attachment points, and are used for routing. This allows for EIDs to remain unchanged even if a topological change occurs, such as a handover in a mobile network. In LISP, packets in transit are encapsulated; the outer header contains RLOCs while the inner contains EIDs. LISP also introduces a *Mapping System (MS)*[86], a distributed database that maps EIDs to RLOCs.

LISP, as an architecture, provides two important features to the Internet:

i. First it truly splits location from identity, which is a requirement to provide native mobility and multihoming. With LISP, mobile clients can be seamlessly equipped with multiple wireless interfaces, and handover from different points of attachment, or among interfaces.

ii. Secondly, it provides a new level of indirection. A hostname lookup in DNS returns an EID, a second lookup is required to the *Mapping System* to find the associated RLOC. With LISP, this MS acts as a location management system. But unlike in traditional mobility protocols, such as Mobile IP[87, 88], LISP's MS is distributed and federated. Mobile IP's location management system (the Home Agent) is deployed at the mobile client's service provider. The other advantage is that LISP's MS avoids mobile service provider lock-in. Within the

LISP architecture[9], LISP-MN [89] specifies the mobility functionality

2.8 Related Work

The research presented in this dissertation focuses on XCAST deployment and QoS provisioning. The work also aims at positioning XCAST for utilization in the Future Internet using LISP architecture. So far no previous work in XCAST have delved in the areas forming the key pillars of this research. However, XCAST is a form of multicast. So the possible related work mostly occur in multicast-centric researches.

2.8.1 XCAST deployment

While there is no previous work targeting efficient XCAST deployment and no previous work was found for a Routing Engine concept, similar or related to the one presented in this dissertation, our own laboratory had previously run a related project. This investigated a "NAT free" routing using a component called ""Application Layer Router (ALR)"[90] and a middleware called Scalable Adaptive Multicast Toolkit (SAMTK)[91].

2.8.2 XCAST QoS Provisioning using DiffServ

As mentioned in section 2.5.2, no significant research exists in the area of XCAST QoS. However some pieces of work aimed at utilizing DiffServ for QoS provisioning in multicast do exist. While they might be treated as related pieces of work, they cannot be applied to XCAST because they are either based on the "host-group" model or require multicast tree operations that are dependent on Multicast Forwarding Tables (MFTs). XCAST on the other hand delivers multipoint communication functionality using unicast routing tables and protocols. As detailed in the appendices section of this dissertation, we extensively studied these multicast-centric approaches and

classified them into five classes. We then showed two examples of each class and theif strengths and weaknesses. See the appendices for more details.

2.9 Summary

This dissertation gives a detailed work aimed at realizing multipoint communication over the Internet using XCAST. This entails deployment of XCAST, quality assurance for applications running on XCAST and strategically positioning XCAST for use in the future Internet. The remaining chapters of this dissertation discusses each of the three areas in details and in the appendices, we give a survey of the extensive study we conducted on the challenges of QoS provisioning for multipoint communication using Differentiated Services.

2.10 Conclusion

This chapter gives a background information of the various multipoint communication protocols discussed in this dissertation. We also describe in details, the Differentiated Services (DiffServ) architecture and how it fits in the QoS provisioning aspect of this research. The main purpose of this chapter is to prepare our readers so that they can be better placed to understand the subsequent chapters that focus on the various technical details of which, without a background information, would be so hard to understand. To this effect, this chapter focuses not only on the definition of the various terminologies used in this dissertation but also on the historical information behind the two key technologies of multicast and XCAST that form the pillar of this research.

Chapter 3

XCAST6 Routing Engine

3.1 Overview

Even though several multicast variants exist, multicast deployment has been a challenge. In multicast address allocation, a multicast group address must be unique in its scope. However, on the Internet, this scope will often be global. Therefore implementing multicast at router-level still faces scalability problems especially in the number of groups that can be supported. Explicit multiunicast(XCAST) solves this scalability problem by using unicast routes thereby eliminating multicast state information maintenance in routers and complex distribution tree construction algorithms. However the custom header structure of XCAST has also created obstacles in its deployment in the real-world. This chapter discusses our proposal, called an "XCAST6 Routing Engine" which is an out-of-the-box solution that simplifies gradual deployment of XCAST in the real-world. With the XCAST6 Routing Engine, we not only provide a simple solution that can hasten deployment of XCAST on the real Internet but we also exemplify experimentally using a number of performance metrics, that contrary to other perceptions, XCAST does not actually add an extra-ordinary load to the routing resources.

3.2 Introduction

Multipoint communication has moved from research to deployment then back to research issues again. For instance, deployment of new services such as IPTV coupled with the increased use of collaborative applications and the emerging future multi-service Internet have reignited interest in research in multicast for both fixed and mobile networks. Multicast has been researched extensively over the nearly 30 years of the Internet. However challenges still persist regarding its deployment at network router-levels.

In multicast address allocation, a multicast group address must be unique in its scope. On the Internet, this scope will often be global. Additionally, most multicast routing protocols exchange messages that create state for each (source, multicast group) pair in all the routers that are part of the point-to-multipoint tree. This per-flow signaling can possibly create huge multicast forwarding tables on the Internet routers[92]. Therefore different multicast variants exist but most of multicast applications implement multicast at the application level; commonly known as Application Layer Multicast[36, 37, 38].

Explicit multiunicast on IPv6 (XCAST6) solves multicast's group scalability problem by using unicast routes to deliver point-to-multipoint packets. It thus eliminates multicast routing tables, per-flow signaling and complex distribution tree construction algorithms. XCAST6 can also simplify migration problems in multipoint communication when combined with mobile IPv6[88]. Its efficiency can also be enhanced using Sender Initiated Congestion Control protocol[93]. However, deployment of XCAST6 has had a few challenges with XCAST6 version 1.0 having been designed to utilize hop-by-hop options header for deeper packet inspection. Hop-by-hop options header has inherent characteristics that increase a router's susceptibility to denial of service attacks[94] hence its use in XCAST6 was a drawback which has since been resolved by its total elimination in XCAST6 version 2.0[4] all but with new deployment

challenges.

This chapter is organized into nine sections. In the next section, we discuss the header structure of XCAST6 version 2.0 while in the fourth section we define and show the need for the XCAST6 Routing Engine. The fifth and sixth sections address various aspects of the engine design while in the seventh section, we show how the XCAST6 Routing Engine can be implemented using FreeBSD operating system. In the eight section, we show performance evaluation of the Routing Engine. Finally in the last section we mention some research work related to XCAST6 Routing Engine.

3.3 XCAST6 Header Structure

Section 2.3.3 gave the background information of XCAST6 protocol. This section will be examining XCAST6 in details, beginning by a look at how the XCAST6 header structure is organized. As of the time of this writing, the latest version of XCAST6 is XCAST6 version 2.0 in which the hop-by-hop extension header has been deprecated. A simple XCAST6 version 2.0 packet comprises of two IPv6 headers, a routing extension header, a transport header and the payload as illustrated in figure 3.1.

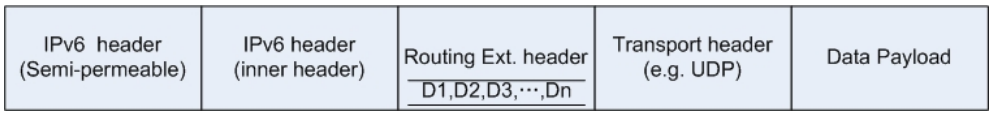


Figure 3.1: XCAST6 version 2.0 header summary

The outer IPv6 header is used to prepare a semi-permeable tunnel[3]. Semi-permeable tunneling is a trick like IP over IP tunneling that XCAST6 uses to make the XCAST6 datagram pass over non XCAST-aware nodes. The traffic class of the outer IPv6 header is “010111XX”. The first four bits of the traffic class are the “experimentally-assigned bits for XCAST6 by IRTF SAM RG”, while the fifth

and sixth bits are for experimental or local use as described in RFC2474[8] and RFC4727[95]. The remaining two bits, "XX" are Explicit Congestion Notification (ECN) bits as specified in RFC3168[96]. The Flow label comprises of three parts namely: "01010111" which is the ASCII code of 'X' (0x58), reserved bits ('00000' by default) and the offset of ICMP target that specifies one of the destinations in the address list for which ICMP reflection, echo reply or errors, is not ignored. The 'NextHeader' points at the inner IPv6 header of an XCAST6 packet. The source address field contains either the address of the source node or that of the latest branching router while the destination address field is usually set to the first address listed in the destination bitmap. Figure 3.2 shows a detailed view of the outer IPv6 header.

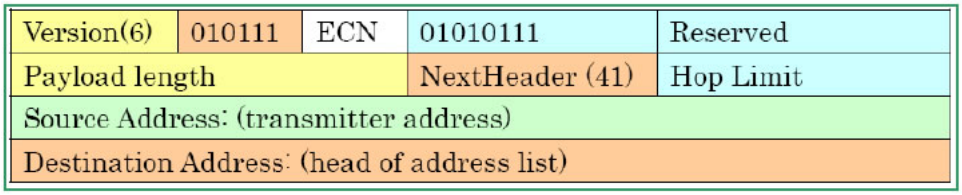


Figure 3.2: XCAST6 version 2.0 Outer IPv6 header

The inner IPv6 header shown in figure 3.3 is processed by the node or the router specified by the destination address of the semi-permeable header. Its source address is set to the unicast address of the original XCAST sender and its destination address set to ALL_XCAST_NODES. If a node is XCAST-aware, it will know how to process this header. However, for non XCAST-aware nodes, they simply drop the packet since ALL_XCAST_NODES is in the range of multicast addresses and is required to be dropped without any ICMP notification by any node that cannot process it.

The routing extension header in XCAST6 is used by the sending node to embed the list of destinations into XCAST6 header and also to maintain a bitmap for tracking XCAST packet delivery. The Nextheader and the Header extension length are filled with the type of the next header and the length of the routing header respectively. The type value in the routing header is 253, for "XCAST route", from the experimental

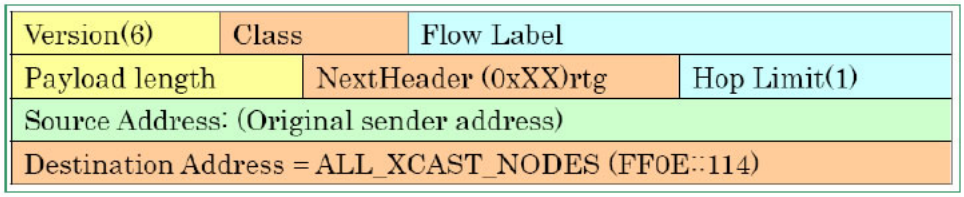


Figure 3.3: XCAST6 version 2.0 Inner IPv6 header

values defined in RFC4727. To guarantee that non XCAST-capable routers discard the packets without replying with an ICMP error message, it is recommended that the fourth octet of the routing extension header be filled with zeros.

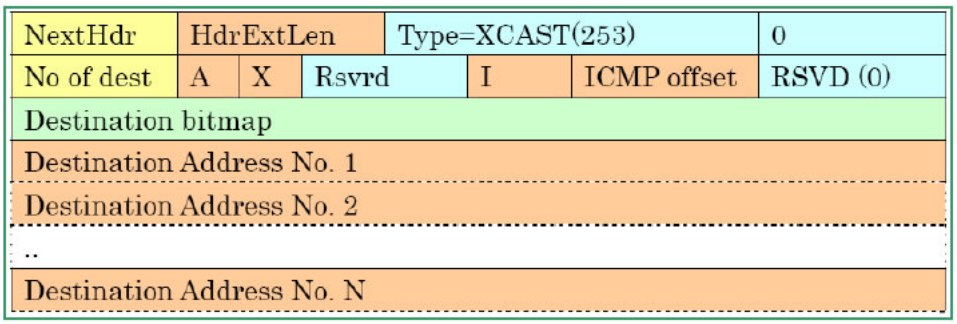


Figure 3.4: XCAST6 version 2.0 Routing extension header

The number of destinations is contained in the fifth octet of the routing header. Due to the length limitations of the IPv6 routing header itself, the maximum number of destinations for XCAST6 is 126. To keep track on which hosts, the packets are to be delivered at each branching point, a bitmap is maintained in the routing header such that when a given field of the bitmap is set to 1, then a packet needs to be delivered to the corresponding destination, otherwise if a bitmap is not set, there is no need to deliver a packet to the destination address corresponding to the bit in the bitmap. The transport header in XCAST6 header defines the transport protocol family that needs to be used. XCAST has been tested with multimedia applications hence the transport headers of choice have been UDP and RTP due to their preference in transmission of multimedia content.

3.3.1 XCAST6 Processing in routers

When an XCAST packet is received by an XCAST-aware router, the router:

i. Performs a route table lookup to find the next hops for each of the destinations listed in the XCAST packet

ii. Partitions the set of destinations based on their next hops

iii. Replicates the packet so that there is only one copy of the packet for each of the next hops that had been determined in the previous steps.

iv. Modifies the list of destinations in each of the copies so that the list in the copy for a given next hop includes just the destinations that ought to be routed through that next hop.

v. Sends the modified copies of the packet to each of the next hops.

vi. If there is only one destination left for a particular next hop, the router can further take an optimization step in which the XCAST packet is sent as a standard unicast packet in a process called XCAST to Unicast (X2U).

3.4 XCAST6 Deployment in the Internet

Experiments and small scale video conferencing have been used to prove the advantages of XCAST6 especially in terms of group scalability[97]. Nonetheless deployment in the real world has not been easy. This is because XCAST protocol has a custom header structure with a new processing algorithm that is not understood by the commercial routers in the market today. However it is impractical to replace the existing routers with new XCAST-aware routers. Moreover, the huge capital investments already put into the existing infrastructure on the Internet must be protected. Therefore two options for deployment of XCAST in the Internet can be postulated

as follows: either to deploy XCAST within the existing infrastructure of commercial routers or come up with XCAST-aware routers.

3.4.1 Using existing commercial routers

As explained earlier, the existing commercial routers are not XCAST aware. They cannot be expected to processes XCAST packets correctly. Instead, they will simply read the semi-permeable tunneling header (outer IPv6 header) and route the packet as if it were an ordinary packet. By deploying XCAST in the current Internet therefore, it means that XCAST can only be processed at end hosts that are XCAST aware. This will often imply an inefficient approach to XCAST processing due to successive (cascaded) delivery of data among XCAST receivers. This scenarion is shown in figure 3.5.

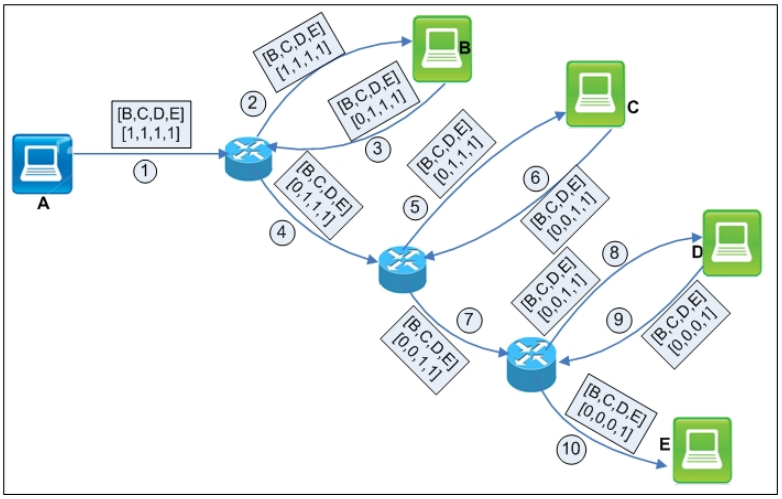


Figure 3.5: Inefficient XCAST processing in conventional network

To solve this problem, routers in an XCAST network need to either be XCAST-aware or have an out-of-the box XCAST processing mechanisms.

3.4.2 Using XCAST-aware routers

This is the ideal situation but the problem is that this ability is currently not available among commercial routers. The router vendors need to be shown the benefits of XCAST and also a confirmation that XCAST processing cannot incur severe penalties in the routers workload so as to motivate them to build XCAST-aware routers. To achieve this, we need to investigate the impact of XCAST processing in routers. The empirical evidence of the impact of XCAST processing in routers will play a key role in motivating the vendors to add XCAST stack into their forwarding engines. It is on this premise that we propose an out-of-the-box solution we call an "XCAST6 Routing Engine" that can be used to realize gradual deployment of XCAST6 in the real-world and then investigate further optimizations that could be done to help embed XCAST into the future commercial routers.

3.5 XCAST6 Routing Engine

The XCAST6 Routing Engine is an XCAST6-aware node connected to the core router. Its purpose is to process XCAST6 packets as had been outlined earlier then send back the processed XCAST6 packets to the core router for further onward delivery.

3.5.1 Implementation options

The XCAST Routing engine functionality can be achieved using three main approaches:

i. Commercially supplied SDKs

ii. Network processors

iii. External Software routers

Each of the above approaches have strengths and weaknesses which we assessed in our decision making process before we settled on the third option.

3.5.2 Commercially supplied SDKs

Until recently, it was not possible to add custom code into the commercially available routers. However, the leading commercial router vendors have recently opened up this space by the introduction of their legacy Software Development Kits (SDKs). The SDKs can be used to extend the functionalities of those vendors' routing devices. Juniper Networks Inc. introduced the JUNOS SDK[98] while Cisco Systems Inc. has also introduced the Cisco Application Extension Platform (Cisco AXP)[99].

While this is a commendable effort and a truly plausible option toward integrating XCAST into the commercial routers, we did not choose it because of the following possible drawbacks:

i. *Impact on production environment*: Since the source code of the new application is added into the routers already deployed in production environments, bugs in the newly added source code can impact negatively on the router, stalling the entire existing routing infrastructure. This means that separate routers are required for testing. Nonetheless ensuring a 100% bug-free software cannot be guaranteed and the impact of such bugs can be catastrophic. Production routers can be completely crippled leading to losses in orders or magnitude and even law suites from disgruntled clients.

ii. *Required skill set*: Since the vendors used legacy codes, this approach will require understanding the the vendor-specific coding conventions which might take time to be mastered by the application programmers.

iii. *Licensing fees*: Acquiring the SDKs attract licensing fees which definitely cannot be affordable to everyone.

iv. *Time constrains*: Programming new applications on top of the existing router codes can be time consuming.

Hence the need for a simple, cost effective XCAST routing engine was realized in this research. To implement this, we used FreeBSD operating system, which is open source and requires no vendor specific skill set.

3.5.3 Network processors

A network processor is an integrated circuit which has a feature set specifically targeted at the networking application domain[100, 101]. Figure 3.6 shows Intel IXP420 network processor. Network processors are typically software programmable devices and would have generic characteristics similar to general purpose central processing units that are commonly used in many different types of equipment and products. Therefore XCAST6 processing algorithm can be implemented in a network processor and the processor then plugged into the available slots in the circuit boards of commercial routers.

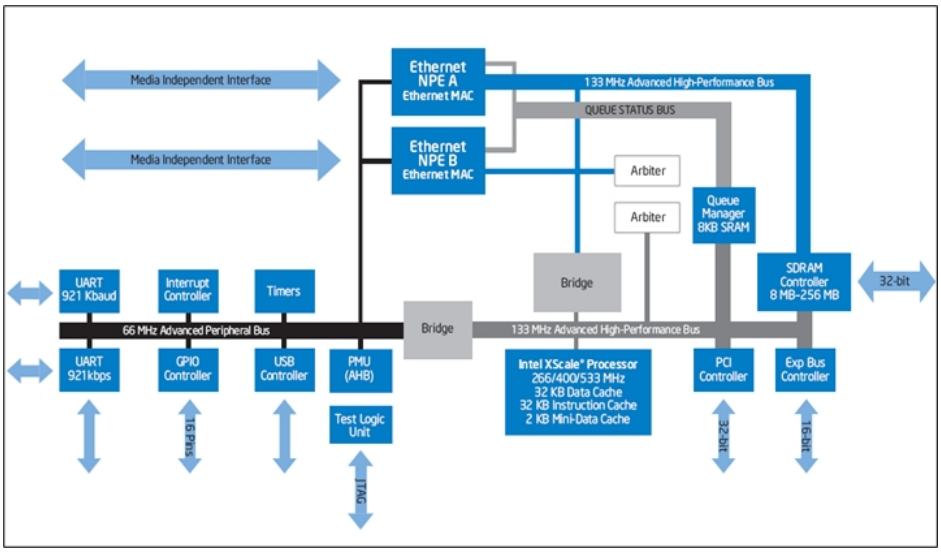


Figure 3.6: Intel IXP420 Network Processor

This technology is similar to the special routing boards and technologies used in

systems such as Open flow[102]. However this approach is likely to face two major challenges:

i. *Special programming skills*: Working with network processors require special programming skills especially a good experience with Hardware Description Language (HDL) such as Verilog. However HDL experts are few compared to those of other high level programming languages such as C, C++ or Java.

ii. *Time constraints*: Programming, testing and installation of network processors takes a longer time and would not be simple and cost-effective compared to other approaches.

3.5.4 External Software routers

This approach uses a PC in which the XCAST protocol has been implemented. We implemented XCAST in the FreeBSD operating system kernel and used the XCAST aware PC as the routing engine. It is connected side-by-side to the core router as shown in figure 3.7 and acts as a "software-router" for XCAST6 packets. As shown in figure 3.7, inbound packets in step 1 are examined by the core router and non-XCAST traffic is handled by the core-router's forwarding engine while XCAST6 traffic is deflected to the XCAST6 Routing Engine in step 2 for processing. The XCAST6 packet is partitioned accordingly and sent back to the core router in step 3 where they are delivered to their final destinations as shown in steps 4 and 5.

3.5.5 Factors to consider in the design

In order to realize this design, we investigate factors that need to be considered namely:

i. How to identify and filter XCAST6 packets inbound to the core router.

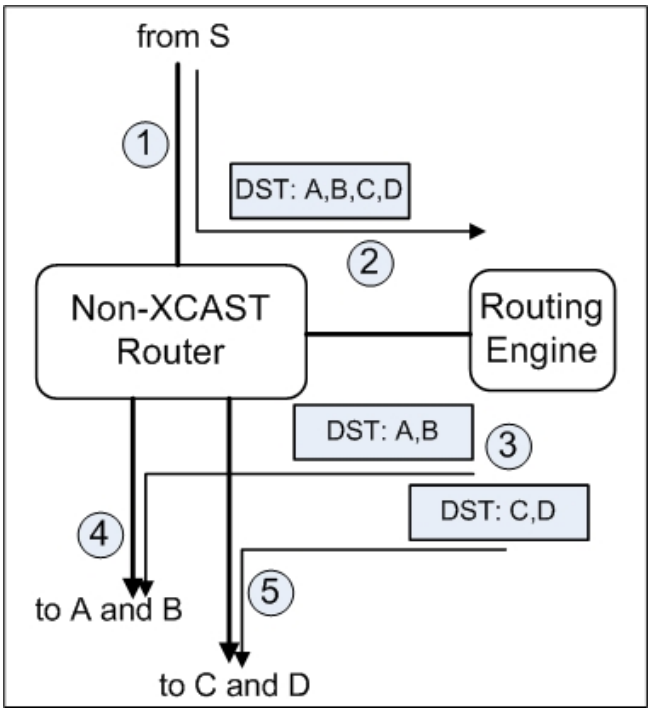


Figure 3.7: XCAST6 Routing Engine

ii. How to process the XCAST6 packets in the routing engine and still realize the same next hops as if processing was done in the core router.

iii. How to forward XCAST6 packets correctly from the XCAST6 Routing Engine.

3.6 Identifying and filtering XCAST6 packets

At the core router, only XCAST6 traffic is re-directed to the XCAST6 Routing Engine. The usual traffic remains to be processed within the core router itself. This requirement can be realized using policy routing framework. With this framework, we can implement a set of rules defining the relationship between the router and the external world in terms of the route information exchange and protocol interaction. We can define the list of routes that the router will accept from its peers, the list of routes the router can propagate to its peers and also determine the redistribution of routes between protocols and interfaces defined in the router. To identify XCAST6

packets, we implemented a policy-based bit-matching utilizing the traffic class of IPv6 packets. On the core router, the policy matches the traffic class "010111" to XCAST6 and all IPv6 stream with that traffic class are forwarded to the XCAST6 Routing Engine. The policy, which can be implemented as a filter in the core router is associated with all inbound interfaces except the one onto which the XCAST6 Routing Engine is connected. This ensures that traffic inbound to router from all segments are handled appropriately. Below is an excerpt using Juniper's JUNOS syntax to show how we implemented the XCAST6 packet filter on a Juniper router.

```
firewall {
    family inet6 {
        filter FBF-nxt-hdr {
            term 3 {
                from {
                /*(DSCP 6bit part)*/
                    traffic-class 23;
                }
                then {
                /*Not a must*/
                 count traffic-class-010111;
                /*Assign lookup table*/
                 routing-instance FBF-nxt-hdr;
                }
            }
            term all {
            /*NB: Default is 'discard'*/
                 then accept;
             }
        }
    }
}
```

Figure 3.8: XCAST6 Policy routing on Juniper JUNOS

For incoming packets, the policy, implemented as a filter-based firewall, "FBF-Nxt-hdr" on IPv6 traffic (inet6) matches the 6 bit part of the IPv6 traffic class used for Differentiated Service Code Point. It counts the matching packets and assigns them to a specific routing table also called "FBF-Nxt-hdr" for the purpose of simplicity.

In the listing above, the routing instance "FBFNxt-Hdr", specifies the option type as 'forwarding' and the option is associated with the routing information base

```
firewall {
    family inet6 {
        filter FBF-nxt-hdr {
            term 3 {
                from {
                /*(DSCP 6bit part)*/
                    traffic-class 23;
                }
                then {
                /*Not a must*/
                 count traffic-class-010111;
                /*Assign lookup table*/
                 routing-instance FBF-nxt-hdr;
                }
            }
            term all {
            /*NB: Default is 'discard'*/
                  then accept;
             }
        }
    }
}
```

Figure 3.9: Routing instance for XCAST6 packets

assigned a static route specifying XCAST6 Routing Engine as the next-hop. All matching packets are therefore forwarded to the XCAST6 Routing Engine. Once in the XCAST6 Routing Engine, the packets are processed then sent back to the core router for effective onward delivery to their respective destinations.

3.7 Synchronizing Routing Tables

In this architecture, both the core router and the XCAST6 Routing Engine are network nodes, each with its own distinct routing table. However the existence of an XCAST6 Routing Engine is transparent to all other nodes in the network. Therefore XCAST6 packets need to be processed as if the processing was done by the core router performing a lookup on its own routing table. We thus seek to have a mechanism by which the routing table of the XCAST6 Routing Engine and that of the core router can be synchronized. We identified two methods by which this synchronization can

be realized:

i. Using Simple Network Management Protocol

ii. Using Network Configuration Protocol.

3.7.1 Routing Table Synchronization using SNMP

SNMP is configured on both XCAST6 Routing Engine and the core router. A program running on the XCAST6 Engine then invokes SNMP commands to get the core router's routing table. In order to parse the IPv6 routing table in IPv6 MIB tree, we need to know the IPv6 routing table's Object Identifier (OID). The OID is used to invoke either GetNextRequest or Get BulkRequest commands of SNMPv1 and SNMPv2 respectively[103]. The program then parses the dumped routing table to extract each "destination" and their corresponding "next hops" which together form a single route entry in the routing table. The new routes are compared against the route entries in the local routing table of the XCAST6 Routing Engine and any new route identified is updated on the local routing table. The program on the XCAST6 Routing Engine polls the core router to ensure the changes if exist, are updated on a regular interval.

The challenge is that using GetNextRequest in SNMPv1 to traverse the MIB can require a large sequence of request-response exchanges between the core router and the XCAST6 Engine especially in the real-world where core network routers usually have huge routing tables. This can introduce unwanted latencies or CPU load owing that most routers use simple processors. GetBulkRequest in SNMPv2 solves this problem since it reduces the number of protocol exchanges required to retrieve a large amount of MIB data by returning a series of variable bindings in a single response. However, the command generator (XCAST6 Routing Engine in this case) is required to specify a "max-repetitions" count so that the responder(core router) can fill in as many variable bindings as it can without exceeding either this count, or the maximum message size.

The challenge however is that it is not possible to know the number of rows in the routing table before-hand. Therefore we cannot possibly set the 'max-repetitions' to an optimum value. With these limitations, SNMP operations in fetching huge data like the routing table of a core router can be highly processor intensive hence it is not a favoured approach.

3.7.2 Synchronization using NETCONF

NETCONF protocol is enabled in both the core router and the XCAST6 Routing Engine. Additionally, SSH is required by the two nodes[104]. A client application running on the XCAST6 Routing Engine (we implemented a Perl program for this) embeds Remote Procedure Calls in XML (XML-RPC) and issues them to the core router over a secure channel via SSH. The XML embedded RPC request can be customized to request for information relating only to a specified table in the IPv6 Routing table hierarchy.

The advantage of NETCONF over SNMP is that NETCONF operates in a transactional manner thereby manipulating semantically related data efficiently. Whereas SNMP modifies or retrieves the value of a single data at a time, NETCONF modifies or retrieves all or selected parameters in a single primitive operation. This ensures it does not incur load on CPU usage. In Juniper routers, the command to get the routing table data via NETCONF is `<get-route-information>`. The router's response, also in XML-RPC, is processed using a custom XSLT template that extracts the various elements and zeros in on `"destination"` and `"next hop"` items for every single route entry in the table. The Perl program is set to poll the core router periodically to check if new routes have been defined in the core router. If a new route entry is found, the local routing table of the XCAST6 Routing Engine is updated accordingly. Otherwise no operation takes place if the two routing tables are in synchrony.

3.8 Forwarding of processed XCAST6 packets

Routers usually have multiple interfaces, each connecting to a different network segment thereby ensuring that traffic meant for each subnet is routed properly. Considering that XCAST6 packets are processed within the XCAST6 Routing Engine, with a routing table that nearly mirrors that of the core router, we need an interface corresponding to each of the interfaces on the core router. To implement multiple interfaces on the XCAST6 Routing Engine, we use a simple approach for cloning interface as shown in figure 3.10. For example, the core router in our testbed has 4 Gigabit interfaces as shown below.

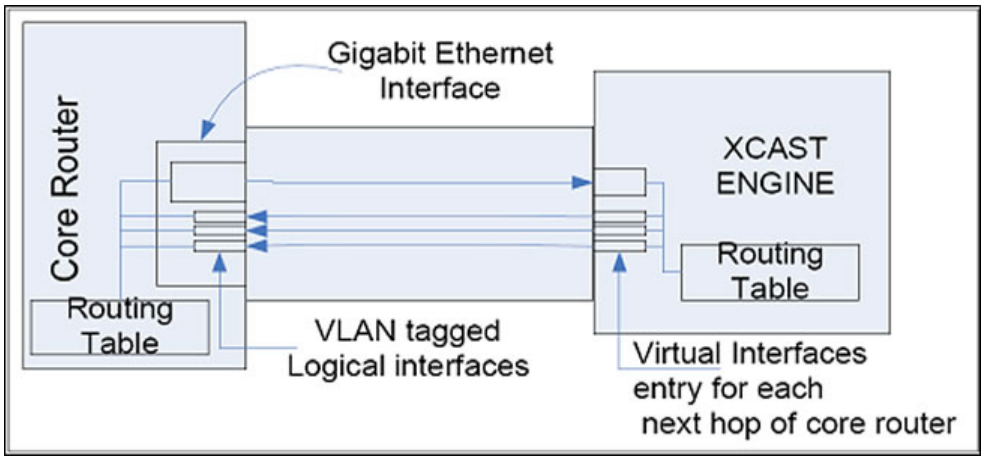


Figure 3.10: XCAST6 Engine Virtual Interfaces

To have a matching number of interfaces on the XCAST6 Routing Engine, we clone a number of virtual interfaces on it using VLAN tagging (IEEE 802.1Q) techniques. Each of these interfaces is configured in its own subnet and ultimately ensures that the processed XCAST6 packets are correctly forwarded to their next hops.

3.9 Implementation

We setup the testbed comprising of a Juniper router, the XCAST6 Routing Engine and other XCAST-aware nodes. The XCAST6 Routing Engine was implemented on a PC running FreeBSD7.2 with Pentium (M), 1.60GHz processor, 760 MB of RAM and 40GB hard disk. The low specifications of this PC is conducive for testing since most routers deployed today also run simpler CPUs and have small capacity memory modules installed.

We installed our latest version of XCAST6 (version 2.0) onto this PC while the other test nodes also had XCAST6 version 2.0 for their respective FreeBSD versions installed in them. The core router in the testbed runs on Juniper J2320 running JUNOS 9.3. It has 4 built-in Gigabit Ethernet ports, 3 modular interface slots, 512 MB DRAM, 512MB compact flash and supports hardware encryption, Unified Access Control and content filtering. All the required configuration were done and the system confirmed to be running well through a process we term as *"testbed characterization"*. Any unneeded protocols was disabled both on the XCAST6 Routing Engine and the core router to ensure that testing is not compromised so much by other external factors. The Juniper router is also additionally deployed onto the WIDE[105] network using the WIDE connection at Nagoya University so as to ensure it operates in a real Internet setting.

3.10 Performance Evaluation

Our aim was to evaluate the routing engine in a real Internet environment. We therefore deployed the routing engine in a real network and used a DV format[13, 14, 15] based videoconferencing application to evaluate its packet processing capability. Digital Video (DV) format is a packet based video/audio encoding in which each encoding scheme specifies a standard digital interface media to exchange the digital stream

Table 3.1: Bandwidth utilization per frame rate

DV Frame Rate	Bandwidth IN (Mbps)	Bandwidth OUT(Mbps)	Expected Bandwidth OUT (Mbps)
1/1	36.024	144.098	144.096
1/2	19.181	76.720	76.724
1/3	13.516	54.062	54.063
1/4	10.683	42.731	42.732
1/5	8.984	35.940	35.937

data. In our experiment, we receive the DV video from the firewire (IEEE1394) interface connected to a camera on one of the PCs in our testbed and receive the video transported across the network using our application on the other hosts in the network. Group management functionality is handled by the Scalable Adaptive Multicast Toolkit (SAMTK)[91]. SAMTK is a middleware for multipoint communication we have developed in our laboratory. DV format was chosen for this test because of the large size of the DV Frames (120Kbytes and 144Kbytes for NTSC and PAL respectively) and the high frame rate required for DV frame transmission (29.97 frames per second).

3.10.1 Bandwidth Utilization

Bandwidth refers to the bit-rate measurements representing the available or consumed data communication resources. It is usually expressed in bits per second or multiples of it (e.g bps, kbps, Mbps, Gbps). Bandwidth is at times used to define the net bit rate, channel capacity, or the maximum throughput of a logical or physical communication path in a digital communication system.

Full DV stream consumes over 30Mbps when using standard NTSC quality video at 525 lines and 29.97 picture frames per second. With multiple receivers placed in different IPv6 network segments in the testbed, we could therefore measure the inbound and outbound packets at the XCAST6 routing engine to determine its pro-

cessing and also calculate the bandwidth utilization for both inbound and outbound XCAST6 traffics. Varying DV frame rates on the sender side also allowed us to determine the processing capabilities of the routing engine with varying number of packets transmitted per unit time. Table 3.1 shows the varying results at each transmission rate.

The XCAST6 Engine is observed to partition packets appropriately and only an infinitesimal variance is noted between the output and the expected values. We attribute this to possible inclusion of the control and session management packets between the nodes and the Group Server in the network.

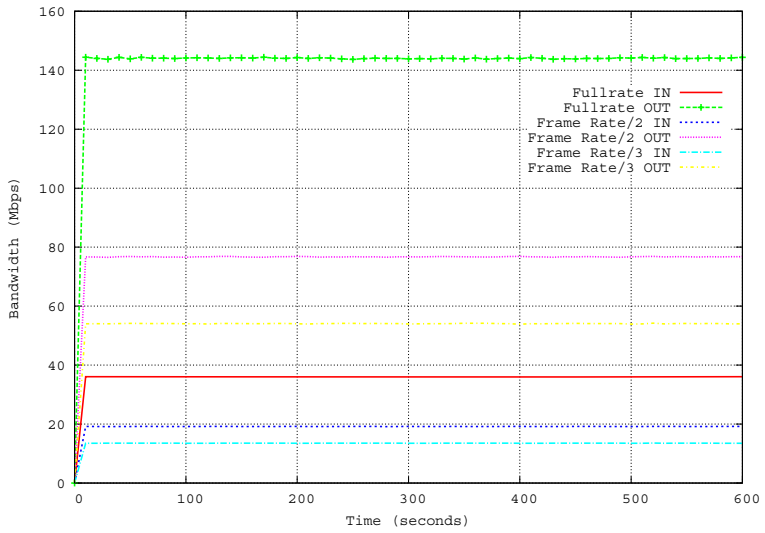


Figure 3.11: Throughput for Full frame rate to Frame rate/3

The graphs in figure 3.11 and figure 3.12 show the variation of the packet processing and bandwidth utilization with time as observed during the processing. The graphs show bandwidth utilization as reported on every 10 second interval. While notable variations were observed in the number of packets processed, the differences were very minimal and a consistent packet processing and partitioning is observed throughout the experiment period. For visibility purposes, we have plotted the observations in two graphs as shown in figure 3.11 and figure 3.12 below.

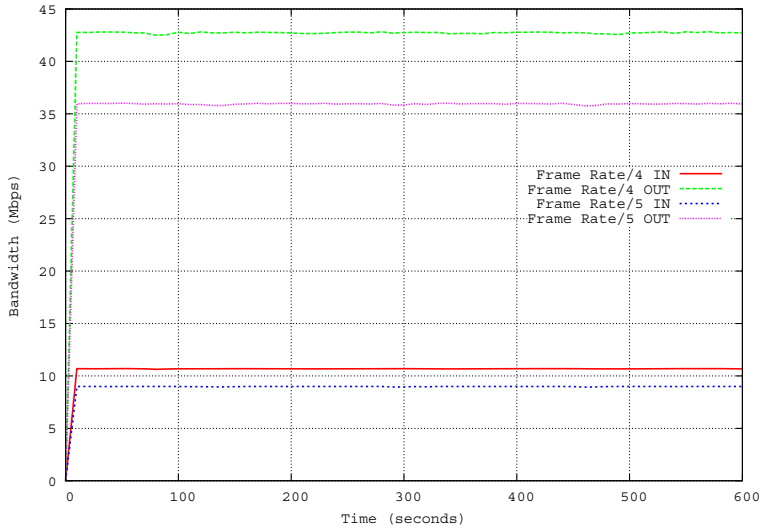


Figure 3.12: Throughput for Frame rate/4 to Frame rate/5

3.10.2 Latency and Latency Distribution

In computer and telecommunication networks, *latency* is defined as a measure of the time delay experienced in the network. Latency can be measured either *"one-way"* (the time from the source sending a packet to the destination receiving it) or *"round-trip"* (the one-way latency from source to destination plus the one-way latency from the destination back to the source). Latency distribution on the other hand refers to the measure of *"how the latency values spread between the lowest and highest"* values recorded.

To effectively measure latency and latency distribution, we used the network performance analysis tool, Spirent SmartBits 600B. The SmartBits 600B had 1 Gigabit card (smartMetrics XD 2 port Gigabit Ethernet, LAN-3320A). We defined five streams on each port based on custom XCAST6 packet with two destinations embedded in the header then measured the percentage utilization of the bandwidth as packet transmission rate (packetss per second) is varied. A constant 136 bytes long payload was used for all the streams. Each stream had an effective length of 286 bytes excluding the signature bits added by SmartBits.

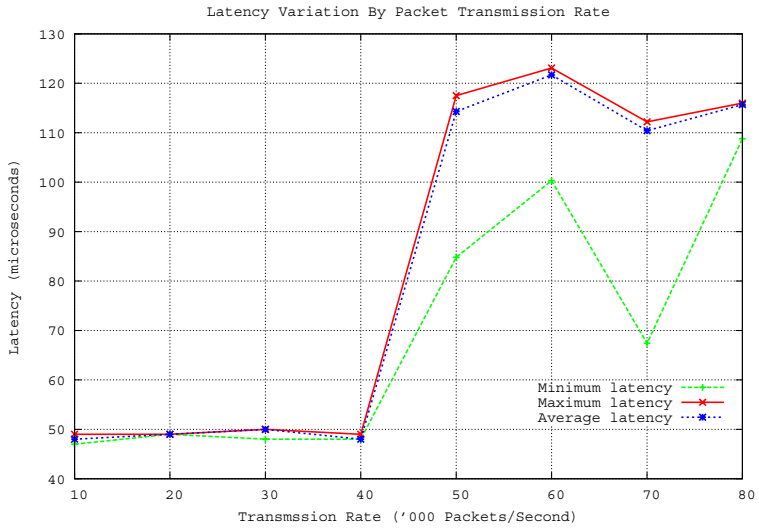


Figure 3.13: Latency variation by packet transmission rate

For all packet rates, the average latency approaches the maximum values observed. We attribute this to the distribution of latencies which tend to be skewed towards the maximum value recorded in each case. At 50,000 packets per second the latency values rise above 110 microseconds which we attribute to the bandwidth utilization which was observed to hit above 50% at this frame rate. From this rate, the Engine starts dropping XCAST6 packets as observed from the log files. This is attributed to replicated XCAST6 packets being too many than the 1 Gigabit Ethernet adapter can transfer. Figure 3.13 shows the observations.

The large variation on the latency values observed motivated us to investigate the latency distribution of these values over the observed range. Since the observations were made at different packet transmission rates and at varying frequencies within each transmission rate, plotting them on histograms would not only be cumbersome but might also not be easy to interpret. We therefore applied the Kernel density estimation techniques in order to observe the percentage distribution of these values.

In statistics, kernel density estimation (KDE) is a non-parametric way to estimate the probability density function of a random variable. Kernel density estimation is

a fundamental data smoothing problem where inferences about the population are made, based on a finite data sample. Kernel density estimation plots provide the probability density estimates for a population using the measured data. This is done by replacing each measurement with a location (mean) of the measurement and a spread (deviation) selected by a free *"spread"* parameter. Then the entire data set is summed and normalized by the number of measurements included. Figure 3.14 shows this estimation. We note that over 35% of the latency values observed are slightly above 110 microseconds, approximately 12% of the values are at 110 microseconds and another 20% fall just slightly below 110 microseconds.

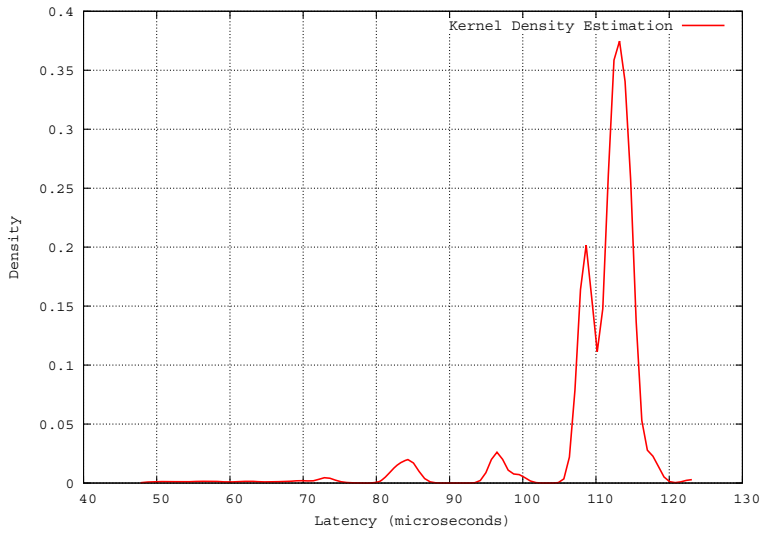


Figure 3.14: Kernel Density Estimate for latency distribution

3.10.3 Packet Loss

Packet loss is defined as the failure of one or more packets to arrive at their destination. Packet loss measurement in this setup was done not at the Interfaces of the routing engine but at the applications on the receiving hosts. Only infinitesimal packet loss of 0.08% percent was detected at the hosts hence we deduce that the XCAST routing engine did not incur a significant packet loss while doing XCAST6

processing. This situation however might change when there are many receivers in the network transmitting many huge packets because of the likelihood of the XCAST6 routing engine replicating several packets when the next hop routers are significantly different for the many receiving nodes. However such situations in the real Internet are demmed to be rare hence, in as much as the packet loss ratio is expected to rise with the increase in the receivers, we still expect an efficient performance by the XCAST6 routing Engine..

3.10.4 Internal System Behaviour

The aim of implementing XCAST6 Routing Engine is not only to simplify the deployment of XCAST6 in the real world but also to help in understanding the impact of XCAST6 protocol processing on the internal behavior of the routers especially with regards to system load level if XCAST6 were deployed in commercial routers. With the XCAST6 Routing Engine, we achieve this objective by actively profiling the FreeBSD system onto which the engine has been implemented when processing XCAST6 packets. One key internal behaviour is the "context switch". A context switch is the computing process of storing and restoring the state (context) of a CPU so that execution can be resumed from the same point at a later time. This enables multiple processes to share a single CPU. Context switches are usually computationally intensive and much of the design of operating systems is to optimize the use of context switches. Therefore the number of XCAST-related context switches can be used to infer the load XCAST processing incurs on routers.

CPU Context Switch Counts due to bus I/O

XCAST6 runs at the kernel level therefore, the reported observation is for "System level" and not "User level" CPU utilization. We used FreeBSD's PMC tools[106], to investigate various internal activities of the system when processing XCAST6 packets.

Using the PMC tools, we registered the counts of context switches related to data read and data writes that the CPU makes when processing XCAST6 packets. This was compared with the observations made when the routing engine is not actively engaged in XCAST6 processing and also to when the engine is processing ICMPv6 packets. Specifically the PMC tool was run by monitoring the changes in the behaviour of process that monitors software interrupts made by the network process (swinet) in FreeBSD. The results are presented as shown in figure 3.15.

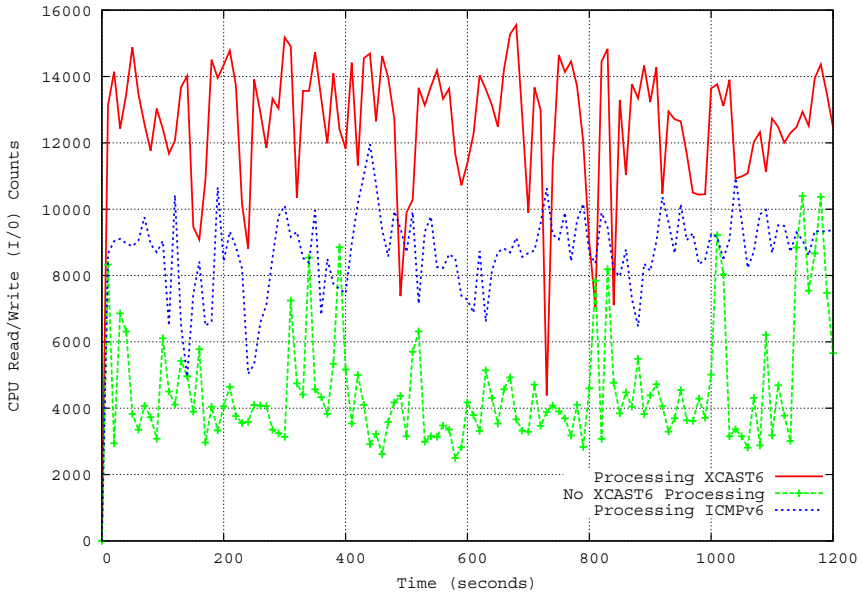


Figure 3.15: Count of context switches due to I/O requests on CPU bus

We observe that processing of XCAST6 incurs a higher number of context switches due to CPU bus I/O activities than ICMPv6. However if we consider this together with the values observed when the XCAST Routing Engine is not processing any XCAST6 packets then the difference while significant, is not too big to cause an alarm. This dispels the general feeling that owing to its complex header structure, XCAST6 could impact heavily on the router's CPU load. However we also note that ICMPv6 registers fluctuations that on the lower bound are almost equivalent to peak observations made when the routing engine is not actively processing data packets.

XCAST6 related observations on the other hand do not show this great fluctuation. This is expected considering the structure of the XCAST6 header which embeds two IPv6 packets.

CPU Context Switch Counts due to memory access

We also investigated the number of context switches made due to memory access requests while processing XCAST6 packets. Hyok Kim et al[107] have shown that these counts are correlated with the number of completed memory transactions and are important because they help in determining the system level memory bandwidth requirements. We therefore use them to help understand such memory bandwidth requirements for processing XCAST6 packets. The observations are shown in figure 3.16. The counts of "system" memory (not userland memory) transactions for XCAST6 and ICMPv6 processing and also when no active packet processing is done show clearly that XCAST6 processing registers higher counts while ICMPv6 maps nearly equally to the system idle state. This is possibly attributed to the XCAST6 packet length which is certainly longer than that of ICMPv6.

Impact of the embedded destinations on CPU and Memory utilization

We further investigated the impact of the embedded destinations in the XCAST6 header on CPU and Memory utilization. We used Spirent SmartBits 600B performance analysis tool in this measurement. Five streams were defined on each port based on custom XCAST6 packet, varying the number of destinations each time and subjecting the XCAST6 routing engine to the huge traffic generated by SmartBits but maintaining the bandwidth utilization at not more than 60% because the resulting XCAST6 outbound traffic are also sent through the same network interface and Vlan tagged interfaces that still depend on the same physical interface. It is observed that XCAST6 packets with fewer destinations consume more CPU resources than those

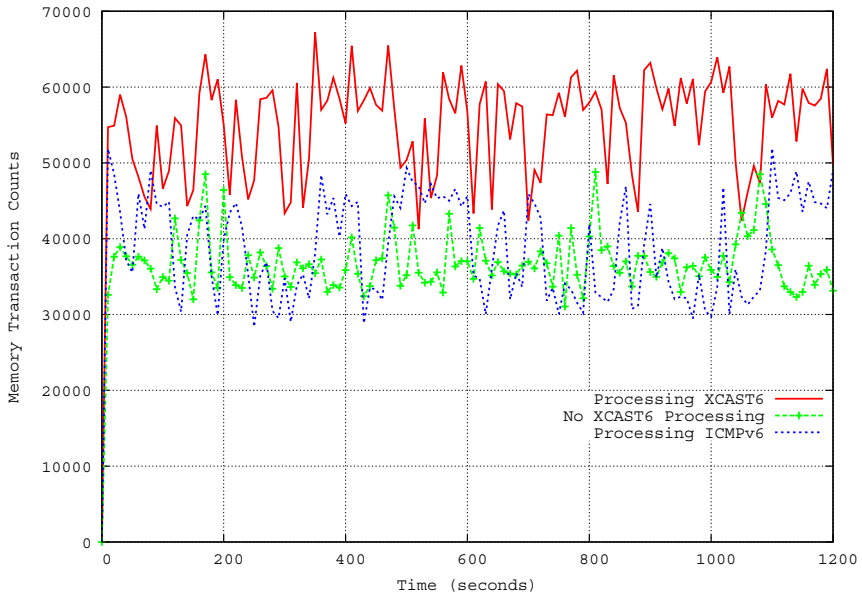


Figure 3.16: Comparative count of context switches due to memory transaction requests

with several XCAST destinations in the header as shown

Packets with fewer XCAST6 destinations are shorter than those with many destinations embedded in the header. Therefore several shorter packets are required to maintain a constant bandwidth compared to those needed for longer packets. With fewer destinations in the XCAST6 header, the CPU processes more XCAST6 packets than it does when the packet embeds several destinations hence the near-inverse proportionality depicted by the graph above.

Despite the higher number of context switches related to System level memory read observed earlier, we also note that once a stable level is realized, XCAST6 does not consume a lot of memory resources even if the number of the headers in the XCAST6 packet is increased. Maximum memory utilization varies only slightly irrespective of the number of destinations as depicted in the figure below. We attribute this to the fact that XCAST6 runs at the kernel level therefore it launches no additional application that requires any huge memory resources.

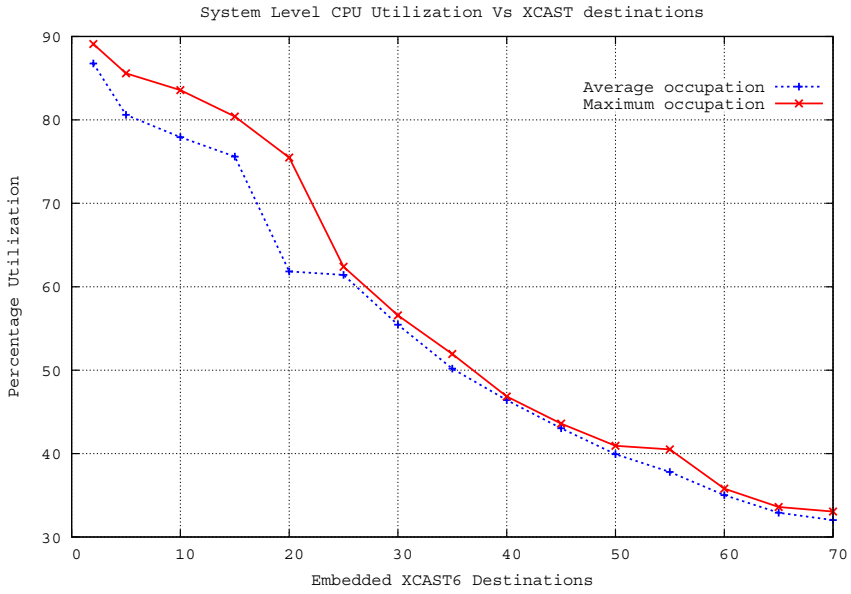


Figure 3.17: XCAST6 CPU Utilization

XCAST6 packet fragmentation

We observed IP fragmentation when more than 70 XCAST6 destinations are embedded in an XCAST6 header. The MTU of the XCAST6 routing engine therefore needs to be set to a value that accommodates longer packets especially in deployment scenarios where the group membership is envisaged to approach 100 nodes. Enabling jumbo packets processing in the engine is recommended. Overall, the XCAST6 Routing Engine showed a very good performance within the acceptable limits. While CPU utilization is higher with fewer XCAST6 destinations, we note that XCAST6 is a group communication protocol hence scenarios like two destination can best work with peer-to-peer. Moreover, it is highly unlikely that a communication scenario can arise that imposes a requirement on maintaining a threshold on bandwidth utilization that we did in our test, for stress testing purposes only. Therefore when used with the right number of destinations that warrant group communication, XCAST6 is not CPU intensive. Should such a need a rise then XCAST6 can be complemented with SICC[93].

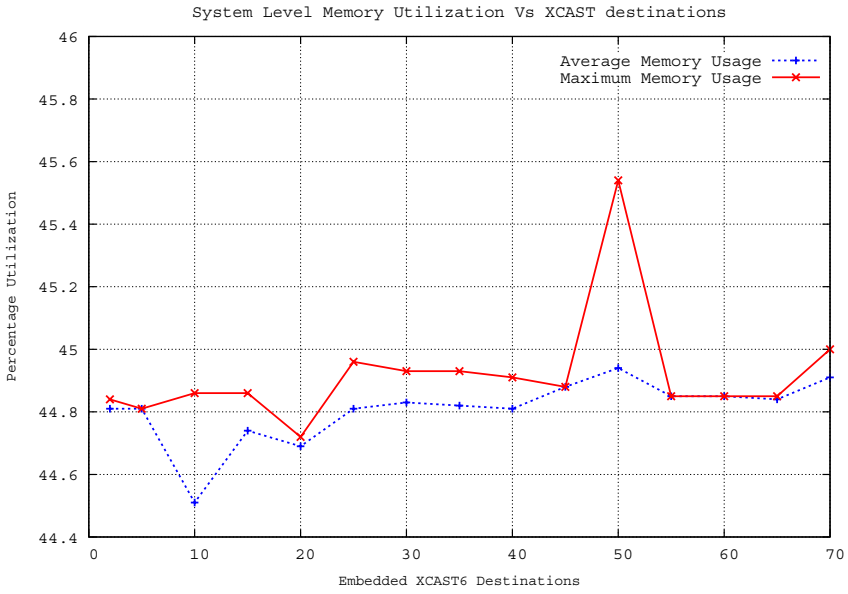


Figure 3.18: XCAST6 system memory utilization

3.11 Related Works

The SAMTK project[91, 90] provides interfaces for XCAST, ALM and ALR network plugins. ALR (Application Level Router) has some components that overlap with the functions of our routing engine. It parses UDP packets and does a lookup in its internal forwarding table to duplicate and deliver the packets to multiple destinations. In addition, it provides NAT traversal function by using a singe UDP port both for session registration and packet delivery. Many recent projects also attempt to define architectures for large scale Internet group services that can operate in the future Internet. OASIS[108] proposes a generic approach to inter-domain multicast, guided by an abstract, DHT-inspired overlay that may operate on a future Internet architecture. It is aimed at facilitating multipath multicast transport, offering fault-tolerant routing, and arbitrary redundancy for packets and paths. However, it is based on the assumptions of a globally available end-to-end unicast routing. Simple technologies such as the one we have presented here can help ensure the end-to-end unicast routing that OASIS assumes to always exist. Multipoint communication

research is also ongoing in structured overlay and hybrid networks. In the HAMCast project[109], Xuemin Shen et al review the key concepts of multicast and broadcast data distribution. They further perform an analysis examining different distribution trees constructed on top of the key-based routing layer. Finally, they compare the performance characteristics of the various multicast approaches and identify major internal differences. This way, they seek to identify key areas that need optimization for application in the future Internet.

3.12 Conclusion

In this chapter, we have described the XCAST6 Routing Engine, a software routing component that simplifies gradual deployment of XCAST6 in the real-world even if the router is not XCAST-aware. We used a low cost PC with modest specification to implement the XCAST6 Routing Engine on both FreeBSD 6.2 and FreeBSD7.2, connecting the PC to a Juniper J2320 router as the core router in the testbed. To handle routing table synchronization issues, we proposed and implemented two different approaches using either SNMP or NETCONF. XCAST Routing Engine provides a simple, efficient and cost-effective way for incremental deployment of XCAST in the Internet. We have evaluated its performance and empirically shown that XCAST processing does not incur heavy penalty on routers.

Chapter 4

Quantitative Simulation of XCAST6 Performance

4.1 Overview

Sometimes real world implementation of XCAST6 can take a long time to deploy and resource constraints can limit the scale of testing that can be done. We have therefore integrated XCAST into the IPv6 stack of INET Framework for OMNeT++ to allow for large-scale, exhaustive and realistic investigation of XCAST6 characteristics and to complement our current research into real world deployment of XCAST6. The contribution of this chapter is two fold. First, it describes our implementation of XCAST6 in OMNeT++'s INET framework. Secondly, it experimentally shows the effectiveness of XCAST6 with regards to various multicast performance metrics such as stress, end-to-end delay, efficiency and packet processing overhead rate.

4.2 Introduction

Numerous multicast technologies have been proposed to ensure efficient utilization of Internet bandwidth for group communication. Even though IP Multicast is conceptu-

ally able to provide efficient group communication and utilize the available bandwidth efficiently[110], its global deployment has not been effectively realized. This has majorly been due to security issues, lack of efficient congestion control scheme and need for special support in the network devices[111]. However, there exist local networks that are multicast capable[112] and are used to deliver TV and Telephony services.

Multicast was originally targeted at very large groups but there also exist group communication applications such as video conferencing, IP telephony and online multiplayer games which only require a large number of distinct, small groups and cannot be served well with the traditional IP multicast. Small group, multi-destination multicast variants have therefore been proposed specifically to serve this class of applications. Explicit multiunicast (XCAST)[3] is a form of small group multicast in which the sender embeds IP addresses of all receiving nodes in a special header which it then sends with every data packet.

Deployment of XCAST in the real world has been a subject of research[97, 5]. In chapter 3 of this dissertation and in other publications[5, 6], we proposed a simple routing engine for realizing incremental XCAST deployment in the Internet. However, we are also cognizant of the fact that sometimes real world implementations can take a very long time to realize. Resource constraints can also limit the scale of testing that can be done. A simulation environment for XCAST6 is therefore necessary. However most of the existing simulators do not have multicast routing models required by small group multicast protocols such as XCAST.

Kolberg and Burford[113], in their project of extending OMNeT++ for Scalable Adaptive Multicast simulations, have come up with XCAST and AMT implementations for the Oversim simulator[114]. Their work however currently supports only simulation of XCAST on the IPv4 stack. On the other hand, our current research on XCAST6 deployment in the real world is based on XCAST6 version 2.0[4] hence we found it necessary to extend the IPv6 stack of the INET Framework[18] in

OMNeT++[11, 10] to allow for simulation of our current research.

This chapter is organized as follows: in the next section we briefly describe OMNeT++ and the INET Framework; the simulation environment in which we implement the XCAST6 Simulator. In section four we describe the implementation of XCAST6 on OMNeT++ while in section five we describe our simulation of XCAST6 using OMNeT++ and discuss the results in section six.

4.3 Simulation tool for XCAST6

4.3.1 OMNeT++

Object **M**odular **Ne**twork **T**estbed in C++ (OMNeT++)[10, 11] is an open source, extensible, modular, component-based simulation library and framework. Because of its generic architecture, OMNeT++ is used in simulating a wide range of networks. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects such as the INET Framework, INETMANET, xMIPV6 and MiXiM among others.

OMNeT++ is a collection of hierarchically nested modules which communicate with each other using message passing and messages may carry arbitrary data structures. The depth of module nesting is unlimited. Modules can be connected with each other via gates (other systems would call them ports) and channels. Channels can posses certain bandwidth, delay and loss characteristics. The modules can also be combined to form compound modules. Modules at the lowest level of the hierarchy are called simple modules and they encapsulate the model behavior. Simple modules are programmed in C++ and make use of the simulation library. In the hierarchy, the top-most module is called the System Module or Network and contains one or more sub-modules each of which could contain other sub-modules.

The structure of the model and that of the modules are defined using a special language called NED (Network Description) language. Each module is usually defined in a separate NED file. Among the features of NED that makes it scale well to large projects are its component-based approach and hierarchical nature. NED supports inheritance therefore modules and channels can be easily sub-classed. Derived modules and channels can add new parameters, gates and even new submodules. NED also uses a Java-like package structure to reduce the risk of name clashes between different models.

OMNeT++ offers an Eclipse-based IDE, a graphical runtime environment, and a host of other tools which support features for development, debugging, running simulations and for visualization and analyses of simulation results. However, for larger networks and more complex simulations, OMNeT++ also has a command line simulation environment that allows for dedication of more computing resources to simulation rather than being consumed in supporting the GUI functionalities. There are extensions for real-time simulation, network emulation, alternative programming languages (Java, C#), database integration, System C integration, and several other functions.

4.3.2 The INET Framework

The INET Framework[18] builds upon OMNeT++ and uses the same concept of modules and messages whereby modules communicate with each other by message passing. It is organized into protocol layers that nearly mirror the OSI reference model. Specifically, the INET Framework contains models for several wired and wireless networking protocols in the Link, Network and Transport layers of the protocol stack. Support for mobility and wireless communication has been derived from the Mobility Framework project[115]. Some of the implemented protocols include: UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, and many others. It

implements the MPLS model with RSVP-TE and LDP signaling.

The INET Framework represents protocols by simple modules whose behaviours are defined in a C++ class of the same name. However, a simple module's external interface such as gates (connectors) and parameters is described in an NED file. To describe protocol headers and packet formats, the INET framework uses message definition files (msg files) which are translated into C++ classes by OMNeT++'s opp_msgc tool. The generated message classes subclass from OMNeT++'s cMessage class.

However, not all modules in the INET Framework implement protocols. Some are used to execute specific non-protocol related tasks. There are modules which hold data (for example RoutingTable), facilitate communication between modules (NotificationBoard), perform autoconfiguration of a network (FlatNetworkConfigurator), move a mobile node around (for example ConstSpeedMobility) and perform housekeeping associated with radio channels in wireless simulations (ChannelControl).

Modules in the INET Framework can be freely combined to form hosts and other network devices with the NED language without C++ code or the need for recompilation whatsoever. Therefore, there are some common modules that appear in all (or many) host, router and device models such as InterfaceTable, RoutingTable and NotificationBoard modules. Modules communicate by message passing and so are the protocol layers. When an upper-layer protocol wants to send a data packet over a lower-layer protocol, the upper-layer module sends the message object representing the packet to the lower-layer module. The lower layer protocol module then encapsulates the message and sends it out. When a lower layer protocol wants to send a packet to an upper layer protocol, it removes lower layer information from the packet (decapsulation) then sends the message to an upper protocol layer module.

Extra information such as connection identifiers, destination addresses, and parameters like the packet TTL are often needed to be transmitted together with the

packets between the protocol layers. This extra information is attached to the message object as Controlinfo. Controlinfo are small value objects, which are attached to packets (message objects) with its setControlInfo() member function. Controlinfo only holds auxiliary information for the next protocol layer, and is not supposed to be sent over the network to other hosts and routers.

4.4 Implementing XCAST6 in OMNeT++

XCAST and AMT are some of the Scalable Adaptive Multicast protocols currently not supported by most of the existing network simulators, especially when analyzing hybrid multicast protocols[113]. In [113], Kolberg et al have proposed an implementation of XCAST for Oversim and OMNeT++ which in the current state, is only on the IPv4 protocol stack of INET Framework. Additionally, their implementation is an XCAST model whereby an XCAST packet gets duplicated at branching nodes and fewer addresses are attached to each copy of the packet until each copy reverts back to a unicast IP/UDP message as the packet traverses the network.

This complex header reconstruction at every branch can be reduced to a simple operation of updating a bitmap[3, 4] in the XCAST6 header and leaving the destinations intact in all copies of the XCAST6 packets. Furthermore, the use of bitmaps instead of the header reconstruction has the advantage that it allows for the possibility of using XCAST with IPSec since only the bitmap is changed and not the entire IPv6 header.

Since our main research on XCAST is in the area of real world deployment of XCAST6 version 2.0[4], our motivation is to realize an XCAST6 simulation environment which can complement our real world XCAST6 deployment when faced with time and resource constraints. Therefore, our target in OMNeT++ was the implementation of XCAST simulation on the IPv6 stack. In our work, in addition

to targetting the IPv6 stack of OMNeT++'s INET Framework, we also implement the XCAST6 concept where the bitmaps are used instead of the complex header reconstruction[3, 4]process.

Our approach also utilizes much of the functionalities in the INET Framework's network layer modules with significant changes done only in the network layer modules. Since the key nodes such as *Router6* and *StandardHost6* modules inherit from most of the network layer modules like *IPv6* and *IPv6ControlInfo*, our implementation does not require any adaptation whatsoever to the existing *StandardHost6* and *Router6* modules. Therefore sample networks can be generated using any existing tools such as OMNeT++ IDE or ReaSE[116] without any other need for adaptation. The subsequent subsections describe our implmentation in each of the layers of INET Framework.

4.4.1 Network Layer

The greatest impact of XCAST6 is in the network layer of the INET Framework. We derived a number of classes to implement XCAST6 functionality while some other classes were only slightly altered by adding a few lines for XCAST6 specific functionality. As specified in [3, 4], XCAST6 destinations and the bitmap are contained in the IPv6 routing extension headers. We modified the *IPv6ExtensionHeader.msg* file to include these XCAST6 specific attributes in the INET Framework's IPv6 routing extension header. This in effect adds these two data containers into the generated *IPv6ExtensionHeader* class.

To facilitate inter-protocol layer transmission of XCAST6 controls, we added the XCAST6 traffic class, destination addresses list and the bitmap in the *IPv6ControlInfo* Module. Traffic class of XCAST6 header is *"010111XX"*, which is *"23"* in decimal notation, so any IPv6 traffic in the simulation with a traffic class of 23 ("010111XX") is recorgnized as XCAST6 traffic and this information is shared between protocol layers

using the *IPv6ControlInfo* Modulue. The *IPv6Datagram* module was also amended to be able to handle the new XCAST6-capable *extension header* modules.

In INET Framework, routing decisions are made in the IP layer. We amended the *IPv6 class* to allow for XCAST6-aware routing. We added a method called *routeXcast6Packet()* in the *IPv6* class and also amended its message handling method to be able to identify XCAST6 messages and handle them by invoking our new XCAST6 routing method.

The *IPv6FlatNetworkConfigurator* that comes with OMNeT++'s INET Framework assumes that all nodes are in the same subnetwork. To work with numerous subnets, we used a *NETCONF-style* XML file detailing routing tables of the nodes within the network. We therefore modified the *RoutingTable6* class such that at stage3 of the initialization process it invoked our newly defined method called *parseXMLConfigFileForStaticRoutes()*. In this method we parse the XML routing file and use the route information to either invoke the *addDefaultRoute()* or *addStaticRoute()* methods of the *RoutingTable6* class.

4.4.2 Transport Layer

In INET, the source and destination addresses of a UDP datagram are defined in the *UDPControlInfo* module. We therefore modified this module to be able to handle multiple destinations as required by XCAST6. In [3, 4], the XCAST6 header is defined such that the destination address of the outer IPv6 header is that of *ALL_XCAST_NODES*, given in the range of IPv6 multicast addresses as *"ff0e::114"*. The destination address used by the *UDPControlInfo* module in our implementation also uses the *ALL_XCAST_NODES* while the list of unicast destinations and the bitmap are contained in our newly defined containers in this module.

There is no further modifications and derivations needed in the implementation of UDP protocol since we use *UDPControlInfo* and *IPv6ControlInfo* to exchange the

XCAST6 information between protocol layers in the stack and just before transmitting the message across the network, we add the XCAST6 information into the *IPv6Datagram* message.

4.4.3 Application Layer

We developed a simple UDP application which can be used to test XCAST6 in OMNeT++. It is based on the *UDPBasicApp* that comes with INET Framework. However we implemented in it the ability to specify a parameter with the list of IP addresses to be used as XCAST6 receivers from the omnet.ini file. In simulation models with several nodes and only a set of them need to form an XCAST6 group, it is possible to specify a number of groups and the IP addresses of the members of the group. The application then delivers data to all members of the specified group.

In the INET's *UDPBasicApp* example the application randomly picks one address and sends data to it. We improved on this such that when simulating on XCAST6, it is possible to specify a list of groups, of course each group also has a list of its members. Our application can randomly select a group and deliver a UDP packet to all members of that group.

4.4.4 Statistics collection

Our implementation also collects various information about XCAST6 packets on each of the nodes in the model. For example we collect information on how many XCAST6 packets have been delivered locally, forwarded by a router, processed by a given node, sent out by the sending node and the number of replications that a router peforms during each simulation run. We shall add more features to this functionality.

4.5 Simulations

We used OMNeT++ IDE to define a model network with a topology similar to the one shown in figure 4.1 comprising of hosts, routers and switches. The model network had seven subnetworks with six routers connected to each subnetwork by an Ethernet switch. In each subnet, we defined ten IPv6 hosts connected to the Ethernet switches by bidirectional links. We considered a simple case where there is only one XCAST6 sender, the host connected to router R1, while all the remaining 70 hosts were receivers. All the links were assumed to have the same bandwidth and equal chances of packet loss. The sender's message frequency was assigned to 0.9s in the omnet.ini file.

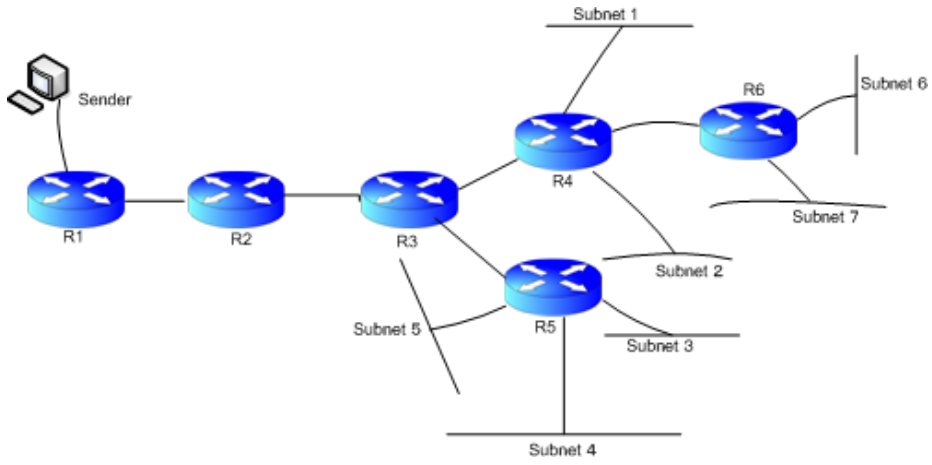


Figure 4.1: Simulation Model Network

4.5.1 Simulation scenario

We are interested in investigating XCAST behavior at different points in the network, namely the sender side, the core and the edge of the network. We therefore investigate the number of XCAST6 packets circulating in the network at each of these 3 distinct points in different simulation scenarios. We defined test scenarios by changing the number of receivers in each case. This changes the length of XCAST6 header and

Table 4.1: Summary of simulation scenarios

Scenario	**Number of Receivers**	**Receivers distribution per subnet**
1	70	All subnets: 10
2	60	Subnets 1 and 2: 5, Other subnets: 10
3	60	Subnets 6 and 7: 5, Other subnets: 10
4	50	Subnets 6 and 7: 5, Subnets 3,4,5: 7, Subnet 1: 9
5	40	Subnets 3,4,5,6,7: 5, Subnet 1: 7, Subnet 2: 8

the number of locally attached hosts for the routers each time. We then monitored the impact of the number of receivers on XCAST6 on each of the routers at various points in the network. Table 4.1 summarizes each of these test scenarios.

4.6 Performance Evaluation

Several performance metrics have been defined to characterize the multicast communication service and its impact on the network[117, 118]. The most important ones being stress (both link and node stress), link stretch, time to first packet, control overheads and efficiency. We measured some of these metrics for XCAST6 as discussed in the subsections that follow.

4.6.1 Node Stress

This is defined in terms of the number of identical packets a physical link (link stress) or a node (node stress) carries in a network. Clearly for XCAST6, the link stress is 1 because no packet is sent repeatedly over the same link. However we measured the node stress since at each branching point, the XCAST algorithm states that the

router duplicates the packet according to the number of the next hops. For our model network routers R1 and R2 have no branches hence the stress is minimal and incurred only in destination lookups for the embedded XCAST6 destinations.

$$\lambda_h = \frac{1}{T} \sum_{i=0}^{T} K \left(L_h(i) + N_h(i) \right) \tag{4.1}$$

To calculate the average stress at the branching routers, we define a number of variables. We define K to be the number of packets sent by the sender, L as the number of locally attached receivers of the packet, N the number of unique next hops for each of the destinations in the XCAST6 header and T the time interval for each measurement split into (i) time slots. For each of the routing node (h), we used the variables K, L, N and T to formulate the simple equation(4.1) which we used in calculating the average stress (λ).

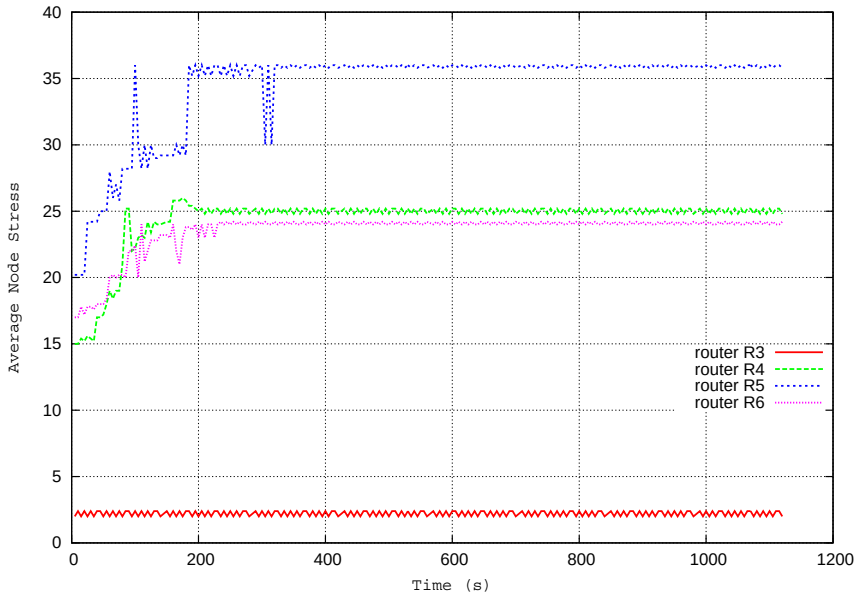


Figure 4.2: Average stress per router for a group of 70 nodes

In all scenarios, we observe that it takes sometime before the stress level stabilizes. We observed that a lot of Neighbor Discovery and Router Advertisement Packets are exchanged in the network over the same period of time so we attribute the shape

of graph at those points to these kind of exchange messages. Using equation (1) we calculated the average node stress at intervals of 50 seconds based on emitted XCAST6 statistical signals. Figure 4.2 shows the results for the first scenario with 70 receivers. Router R3 registers a constant low average stress of approximately 2 which corresponds to the expected unique next hops while router R5 registers the highest stress level averaging to approximately 36. This is because of the several receivers that are locally attached to it. Routers R4 and R6 with the same number of locally attached receivers register nearly similar level of stress but that of router R4 is slightly higher because router R4 also has router R6 as the next hop for all XCAST6 packets destined to subnetworks 6 and 7.

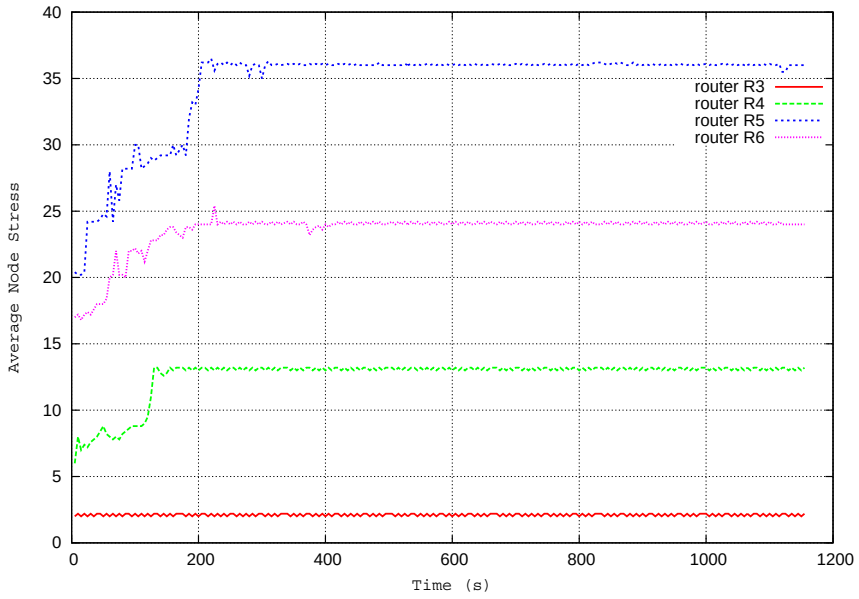


Figure 4.3: Average stress per router for a group of 60 nodes (scenario one)

We reduced the receivers to 60 but conducted two tests by changing the number of locally attached receivers on routers R4 and R6. Figure 4.3 shows the results when the number of receivers attached to router R4 were reduced to 10 (5 in each subnet for subnets 1 and 2) while retaining locally attached receivers on router R6 at 20, (10 in each subnet for subnets 6 and 7).

The stress on R4 reduces while that on R6 remains at the same level registered in the first simulation scenario. The reduction of the number of receivers from 70 to 60 only impacts on router R4 in the entire model.

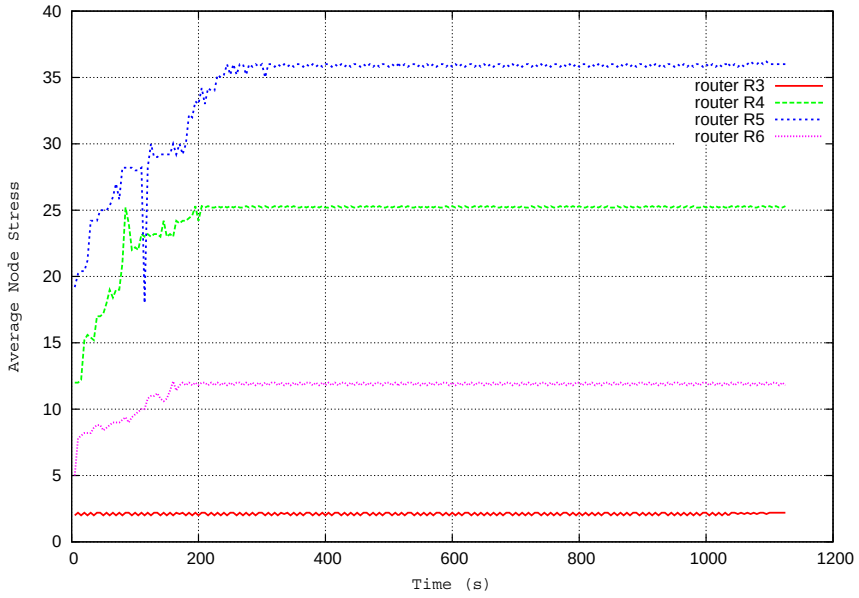


Figure 4.4: Average stress per router for a group of 60 nodes (scenario two)

Figure 4.4 shows that the impact of changing locally attached hosts is noted on routers R6 and R4 while the rest of the routers in the model are unaffected despite the fact that the total number of receivers has been reduced to 60. The fourth scenario had a group size of 50 receivers only and host distributed as summarized in table 4.1. A considerable reduction in stress on router R5 is seen as indicated in Figure 4.5.

In our final scenario, we reduced the number of receivers to 40. For comparison we had 15 locally attached receivers on router R4 and R5 but router R4 also has a next hop for all packets destined to subnetworks 6 and 7. Figure 4.6 shows that router R4 registered the highest stress level since it had more hosts than all the others. In all scenarios, router R3 recorded the least nodal stress.

We therefore note that an important point on effectiveness of XCAST6 protocol is that it enhances the Internet philosophy of pushing all the loads to the edge networks,

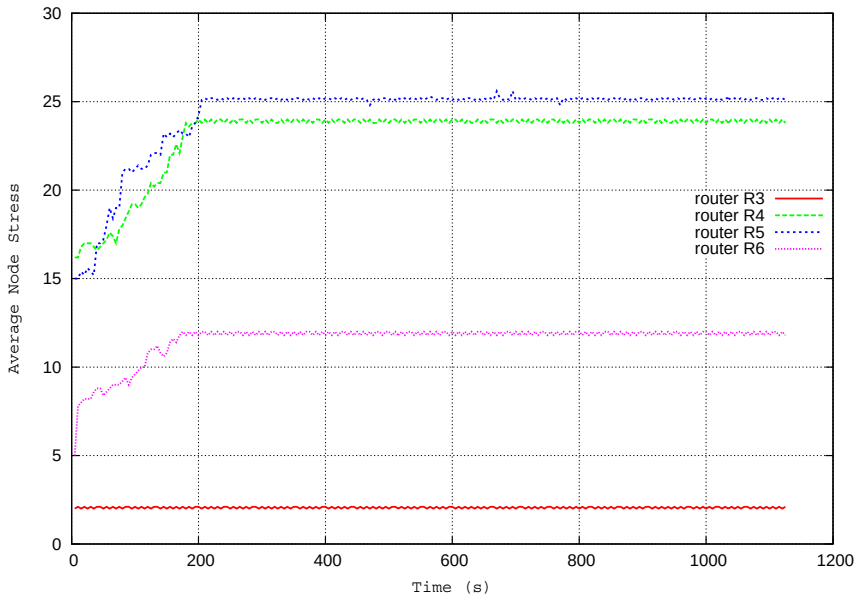


Figure 4.5: Average stress per router for a group of 50 nodes

away from the core. We observe that only the edge routers registered higher stress levels depending on the number of locally attached receivers and the next hops of the packets being transmitted.

4.6.2 End-to-End Delay

We also measured the average end-to-end delay during each of the above scenarios. Using the NICE overlay protocol implemented in OverSim[114], we used the same network model to compare the time delays in XCAST6 and Application Layer Multicast (ALM) and results are shown in figure 4.7. This also acts as an indicator of the processing overhead incurred by XCAST6 due to the increased packet header complexity as compared to ALM. However, XCAST6 is targeting small groups and the difference in end-to-end delay noted is also very small considering it is in the range of very minor fractions of a second.

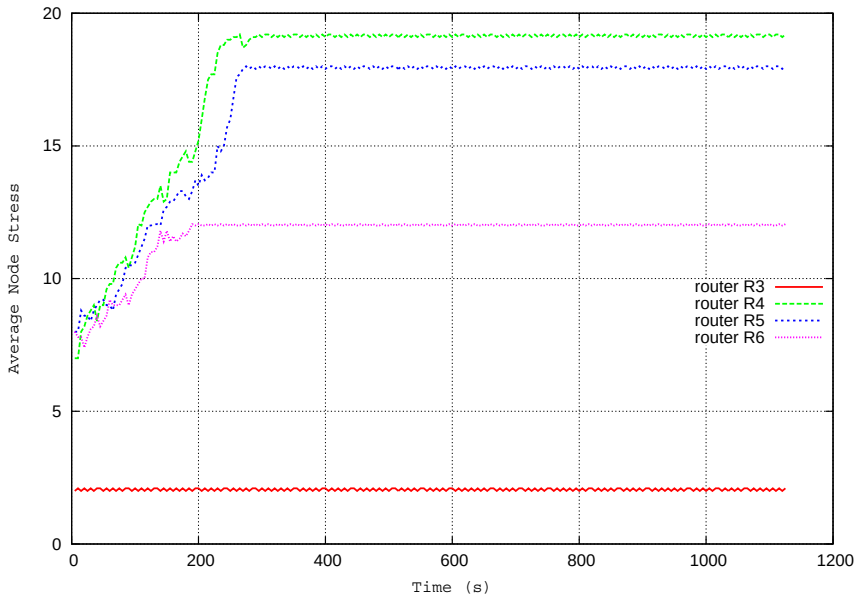


Figure 4.6: Average stress per router for a group of 40 nodes

4.6.3 Cost overhead rate

Considering that the result of section 4.6.2 shows a likely impact of the XCAST6 header complexity as a processing overhead, we compare XCAT6 and multicast based on a factor we define as the cost overhead rate. This is calculated using the number of packets that XCAST6 can generate if used in networks of hosts with varying MTUs such as the Internet. We define the following variables for this comparison: HDR to be the size of IPv6 header, ADR to be the size of an IPv6 address, N to the number of XCAST6 receivers. THDR to be the length of the transport protocol header and P to be the size of the payload data to be delivered in an XCAST6 message. We assume that each bit of the bitmap requires only 1 unit of data to store.

$$f_x(N,P) = \frac{HDR + N*ADR + N*1 + 2 + THDR + P}{MTU} \tag{4.2}$$

Where 2 is added to represent the bytes storing the size of the bitmap and also the length of the header in the IPv6 routing extension header. For the same data,

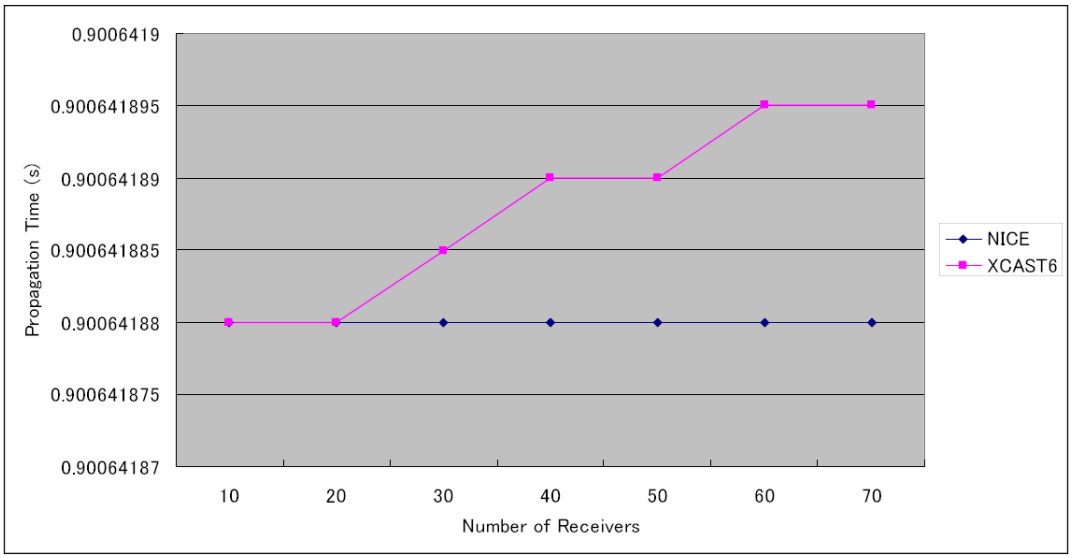


Figure 4.7: End-to-End Delay

the number of packets required by Multicast will be:

$$f_m(N, P) = \frac{HDR + THDR + P}{MTU} \tag{4.3}$$

Now we let a value M be defined as follows:

$$M = HDR + THDR + P \tag{4.4}$$

The ratio of equation 4.2 and 4.3 and substituting for M, can be defined as the XCAST6 gain on Multicast. This comes to:

$$\frac{f_x(N, P)}{f_m(N, P)} = \frac{M + N\ (ADR\ +1)+2}{M} \tag{4.5}$$

We note that ADR and THDR are constant. For IPv6, the ADR is 16 bytes and THDR for UDP is 8 bytes. Therefore the value of (M) in equation 4.4 only varies with the length of the payload data, (P). When we plot equation 4.5 above for upto 75 destinations against the payload size of between 250 bytes and 2000 bytes we find

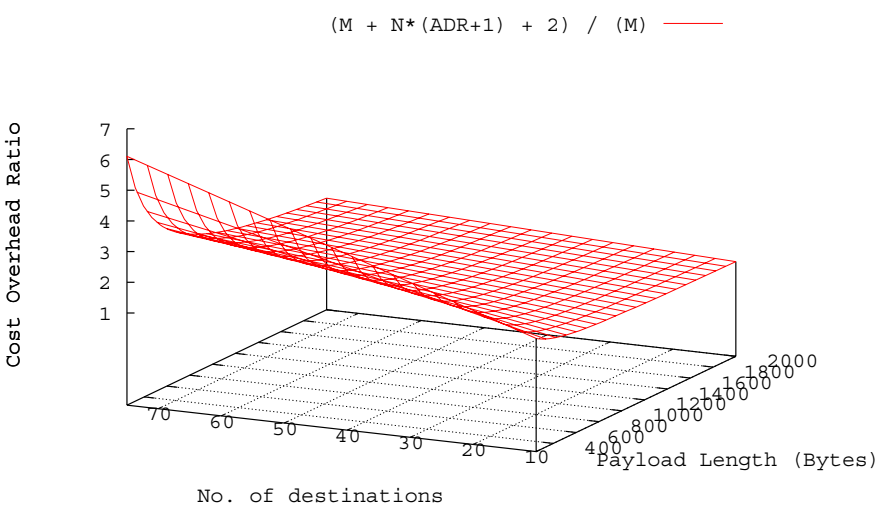


Figure 4.8: XCAST6/Multicast cost overhead ratio

the resulting comparison as shown in figure 4.8. The value of the cost overhead rate goes as high 6.1 when there are many destinations and the payload length is shorter. However, for a majority of cases the rate is between 2 and 2.5 especially for fewer destinations and payload length of between 600 bytes and 1400 bytes. Considering that the MTU for common Network Interface cards is at 1500 bytes, it implies that XCAST6 bears a low overhead for the typical communication scenarios.

4.7 Conclusion

In this chapter, we have shown how to implement XCAST6 on OMNeT++. We chose an approach that would ensure that all existing host modules of OMNeT++ can work without any need for alteration. We have also implemented various signaling information for statistical data collection in the model and used these to measure the stress node due to XCAST packet replication. We have confirmed that the simulation works correctly by analyzing various XCAST performance characteristics.

Chapter 5

Scalable QoS for XCAST using DiffServ Architecture

5.1 Overview

This chapter deals with QoS provisioning in XCAST networks using Differentiated Services (DiffServ). We show that integration of DiffServ in XCAST is a non-trivial problem due to inherent architectural differences between XCAST and DiffServ. We then propose a scheme called QS-XCAST that uses dynamic DSCPs to adapt to the heterogeneity of receivers in an XCAST network. We also provide an algorithm for harmonizing the *receiver-driven* and *sender-driven* QoS approaches between XCAST and DiffServ thereby determining the correct DSCP-PHB for all links in an XCAST network. By simulating using OMNeT++ we evaluate QS-XCAST using four metrics: throughput, average per-hop-delay, link utilization and forwarding fairness to other traffic in the network. Our solution eliminates *DSCP confusion* and *collusion attack* problems to which naive XCAST QoS provisioning is vulnerable. It also offers a more efficient bandwidth utilization, better forwarding fairness and less traffic load compared to the existing XCAST.

5.2 Introduction

The expansive growth of the Internet in the past decades has resulted in merging of both data and real-time multimedia traffic. Availability of adequate bandwidth at low cost however continues to be one of the challenges facing Internet users today. While the available bandwidth to the end users has continuously increased over the same period, the increase is always outdone by an increase in the number of Internet users and the emergence of new applications some of which have intensive bandwidth consumption.

As the number of services and users on the Internet increases, so is their diversity in terms of bandwidth, delay and jitter requirements. These requirements are dependent on several factors like the media processing capabilities of the user devices, the amount of bandwidth the users are capable of paying for and the contract agreements with the ISPs. Quality of Service (QoS) provisioning on the other hand is a complex mix of factors such as bandwidth availability, criticality of applications, pricing and the nature of the underlying networks. Therefore enforcing a single QoS provisioning strategy might not work in this scenario hence the need for heterogeneous QoS provisioning for each of the users and services running on the Internet. This becomes even harder in multicast where data to multiple recipients is sent out in only a single packet.

The Differentiated Services (DiffServ) architecture[7, 8] is one of the proposals made by the IETF for provisioning of QoS in the Internet. In this architecture, the network routers are classified into two main groups of *core routers* and *edge routers*. The *edge routers* are further categorized as either *ingress edge-routers* or *egress edge-routers*, depending on their locations relative to the source of the packets transmitted in the network. These routers form a domain in which the ingress edge-routers' principal task is to mark the packets with specific codes called DiffServ Code Point (DSCP) and each DSCP is expected to allow the flow in the network to be shaped

according to a specific defined behaviour called Per-Hop-Behaviour (PHB)[7, 8, 64, 65, 66].

Explicit multiunicast (XCAST)[3], a promising technology for IPTV, videoconferencing and multiplayer online games, has been proposed as a multicasting scheme with complementary scaling properties which can solve the scalability problems of conventional IP Multicast. Most research in XCAST however have focused on its performance[97, 6, 119, 120, 121, 4, 122] and implementation[5, 3] in the Internet leaving out its Quality of Service.

In addition to difficulties facing muticast QoS provisioning[123], QoS provisioning in XCAST is further complicated by the fact that while XCAST is a form of multicast, routing of XCAST packets does not use the multicast tree delivery paths but instead it uses a unicast routing table[3]. Therefore most of the current solutions on integrating DiffServ with Multicast which are dependent on construction and aggregation of multicast trees cannot be applicable in XCAST networks. QoS provisioning in XCAST using Differentiated Services should therefore be cognizant of the need for heterogeneous QoS owing to the inherent heterogeneity of the receivers and at the same time avoid reliance on multicast forwarding tables. This calls for the need for dynamic DSCP assignment based on unicast routing.

In this chapter we highlight the architectural conflicts between XCAST and DiffServ that make their integration a non-trivial problem and then propose a scheme called QS-XCAST that uses dynamic DSCPs to cater for the heterogeneity of receivers in an XCAST network. We also provide an algorithm for harmonizing the receiver-driven and sender-driven QoS issues between XCAST and DiffServ thereby determining the appropriate DSCP-PHB for all links in the XCAST network. Through simulation in OMNeT++[11, 10] we evaluate our solution using the following metrics: throughput, average per-hop delay, link utilization, traffic load and forwarding fairness. The latter two are obtained from the router's buffer evolution pattern. We

show that our solution eliminates the problems of *DSCP confusion* and *collusion attack* that impede integration of DiffServ in multicast networks. It also gives better performance in terms of bandwidth utilization, forwarding fairness to other protocols and less traffic load on the router.

The rest of this chapter is organized as follows. In the next section we highlight the issues that complicate XCAST-DiffServ integration. We also briefly introduce previous attempts at XCAST QoS provisioning. We then look at currently existing DiffServ-Multicast integration approaches and explain why they are not applicable for XCAST-DiffServ integration. In section four we describe our proposed solution and in section five we implement the solution in OMNeT++ simulation environment and then discuss the simulation results and other possible effects of our proposal in an XCAST-DiffServ network.

5.3 XCAST6 QoS Provisioning with DiffServ

The background of XCAST was introduced in section 2.3.3. Further explanation of the structure of an XCAST packet header was then done in section 3.3. XCAST protocol concept and options have been defined by the IRTF in RFC 5058[3] for both IPv4 and IPv6. The implementation of XCAST on IPv6 is usually referred to as XCAST6[3, 4], which is the focus of our work.

Figure 5.1 illustrates packet processing in a sample XCAST network. The XCAST processing algorithm was described in sections 2.3.6 and 3.3.1. Section C.1 also uses a similar diagram to explain how XCAST packet replication is done. In this section we shall be looking at previous attempts at QoS provisioning in XCAST. We shall also be highlighting the the challenges faced while attempting to use DiffServ architecture in XCAST QoS provisioning.

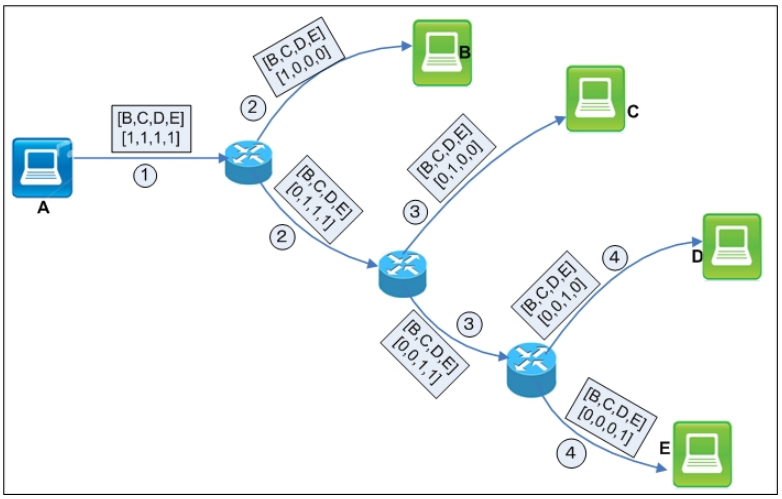


Figure 5.1: XCAST6 Packet delivery.

5.3.1 Problems in XCAST QoS Provisioning with DiffServ

A group communication system using XCAST needs to consider group, network and traffic dynamics that will affect the quality of communication. Group and network dynamics refer to factors such as member join or leave events and changes in the network topology possibly due to node or link failures or the addition of new nodes or links in the network. Traffic dynamics however relate to data flow, link congestion and error control in the network.

To take care of these dynamics, an XCAST network should have some form of QoS provisioning such as integration with DiffServ. Considering the heterogeneity of the receivers, such an XCAST-DiffServ network would have to guarantee dynamic QoS appropriate to the demand of each receiver. This calls for *QoS Precedence* in which each link in the delivery path guarantees data to be delivered at a QoS level not lower than the highest level issued by any of its downstream links. Guaranteeing QoS precedence using DiffServ in an XCAST network is however challenging due to the following reasons:

1. *Multipoint data delivery using unicast routing tables*: XCAST delivers data to

multiple receivers but it does not rely on multicast forwarding tables (MFT) in its routing of data packets. Instead, XCAST routers lookup next hops in unicast routing tables and depend on unicast routing protocols. This means that QoS multicasting techniques that rely on either aggregation or multiplexing of the multicast trees within a DiffServ domain or between DiffServ domains cannot solve the XCAST-DiffServ integration problem on the one hand while on the other hand, unicast based DiffServ cannot provide a solution since XCAST delivers data to multiple recipients.

2. *QoS heterogeneity for a single data packet*: XCAST achieves its multipoint delivery capability through the use of encapsulation techniques whereby multiple destination addresses are embedded in an IP packet header. This means that one packet contains data to be delivered to multiple receivers each of which is likely to have different QoS requirements. The packet's QoS should therefore be such that all the embedded receivers' QoS requirements are met. Boivie et al, in [3] recommend setting the packet's QoS to that of the most demanding receiver. However this approach is insufficient because it leaves serious gaps in management of resources in an XCAST network as we shall be explaining in "*resource management*" section.

3. *Sender-driven QoS versus receiver-driven QoS*: This is fundamentally an architectural conflict between XCAST and DiffServ that complicates XCAST-DiffServ integration. XCAST allows members to join a group at the QoS level that meets their requirements. Furthermore in multicast of which XCAST is a variant, the receivers can request for different levels of QoS depending on the changes in the resources available to them or other dynamics in the network. This is a receiver-driven approach to QoS control which is inherent in multicast (*and by extension in XCAST*) architecture. In DiffServ on the other hand, the QoS provisioning starts from the sender side. The ingress routers insert the

appropriate Differentiated Service Code Point (DSCP) for the receiver in the packet header. The core routers on the other hand simply check the packet's DSCP and then determine the appropriate forwarding queue for the packet. This is contrary to the architectural design in multicast (*and XCAST too*) since the receiver does not mark the packets to determine their QoS level.

4. *Resource management*:

   (a) *DSCP Confusion Problem*: We consider an XCAST network of hosts with heterogeneous QoS requirements as shown in figure 5.2. Assuming this is an IPTV service provider network offering IPTV services at various Service Level Agreements (SLAs) which are classified into various plans dubbed *Premium*, *Gold*, *Silver*, *Bronze* and *Normal*. These plans are then mapped onto *Expedited Forwarding (EF)*[64], *AF41*, *AF31*, *AF21*[65] and *Best Effort (BE)[66]* DSCP-PHB classes respectively. We note that host (*H1*) is a *premium* customer, (*H2*) is a *Gold* customer and host (*H3*) is a *Bronze* customer. Each of them pay different prices for the services and require different treatment but their data is delivered in the same XCAST packet.

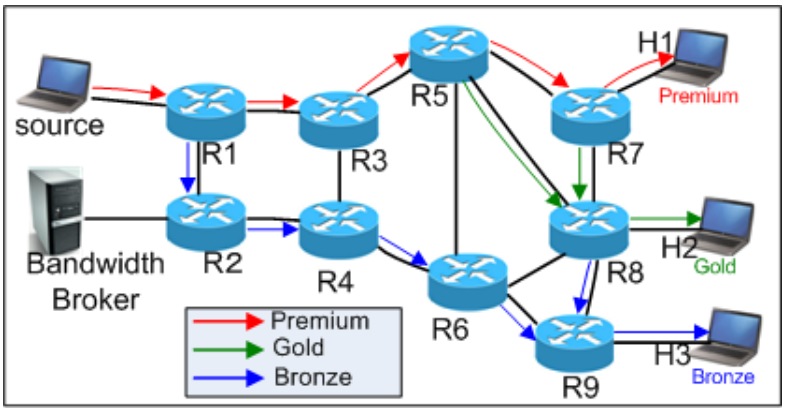


Figure 5.2: XCAST6 Network with heterogeneous QoS requirements

If an XCAST packet is sent from the source to all three customers, then

depending on the routing table entries and the underlying routing protocol in the network, potentially there are three branching routers where the first replication of the XCAST packet can occur; on router R1, router R5 or router R7. Let us assume that replication happens at router R5. Since all the links from the source to the host *H1* are premium (*have EF DSCP*), the DSCP class of the packet from the source will be *EF*. On replication, two copies of the XCAST packet emerge, each of which should now be handled at different QoS levels. The copy bound for *H2* and *H3* through router *R8* should be handled at *AF41* while that bound for *H1* should receive *EF* DSCP treatment .

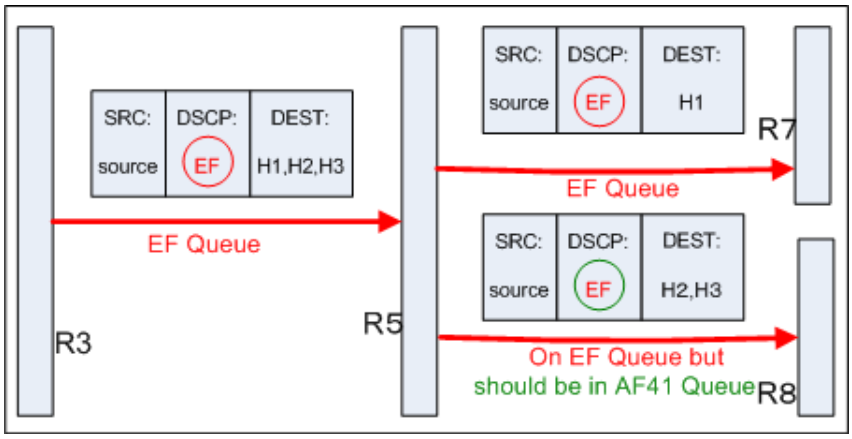


Figure 5.3: XCAST6 DSCP Confusion problem

Since XCAST currently does not support dynamic DSCPs, router R5 will not queue the two XCAST packet copies in their correct DSCP queues as illustrated by figure 5.3. Router R5 will most likely queue the copy bound for *R8* on an *EF DSCP queue* yet it should be queued on *AF41 queue*. This is referred to as *DSCP confusion problem*. While implementing QoS provisioning in XCAST using DSCP, the algorithm should solve such likely DSCP confusion problem.

(b) *Collusion Attack Problem*:

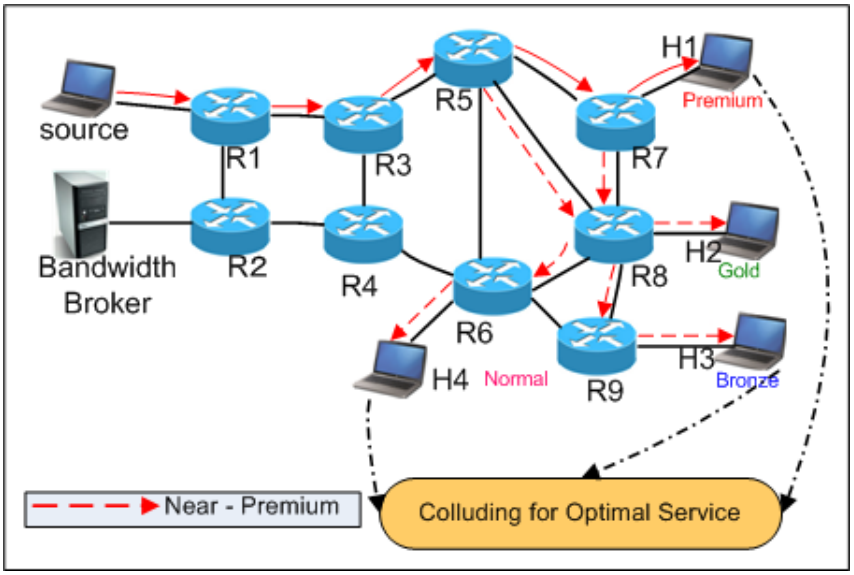


Figure 5.4: XCAST6 Collusion Attack problem

In figure 5.4 we consider a scenario where a new client is enrolled in the IPTV service under the "*Normal*" plan. The new client *H4* joined the network through router R6 and based on the SLA mapping, his/her packets receive *BE* DSCP class. If the source sends a packet to all the hosts (*H1...H4*), then depending on the underlying routing protocol in the network and the entries in the routing tables of each of the routers, such a packet might have three possible branching points; at routers R3 or R5 or even on R7. In such a scenario, in addition to the DSCP confusion problem mentioned above, a new problem arises. Since host *H1* is paying a *premium* rate for the connections from the source all the way (*we assume the path to be R1→ R3→ R5→ R7→ H1*), the resource allocation for this path will be highly prioritized. The IPTV provider will not incur any additional cost in delivering the XCAST packet up to router R7 (*host H1 has paid for it*). If XCAST packet replication occurs at router R7, all the other hosts will be getting the same treatment as *H1* in terms of bandwidth (*and other resources*) allocation as we later show in our simulation. Even (H4) who

is paying for only a *"Normal" (BE)* service will in the real sense be getting near *premium* services. This opens the network to a kind of attack where a subset of clients collude to pay substantially less amount for the best QoS services available, leaving other unaware clients (*Good Neighbors*) to be taking care of the other costs and both the *"Good Neighbors"* and the *IPTV Service Provider* remain oblivious of the unfolding scenario. This is called a *collusion attack* (*in this case all the other hosts H2...H4 can do that. Hosts H2 and H3 can even reduce their SLAs with the source to lower classes while still getting near premium TV services*).

An XCAST QoS provisioning scheme should therefore be more efficient and ensure that at each given moment, receivers actually get only what is acceptable within their respective SLAs irrespective of their heterogeneity. Such a scheme would ensure that the links of the sample network exhibit DSCP-PHBs as shown in figure 5.5.

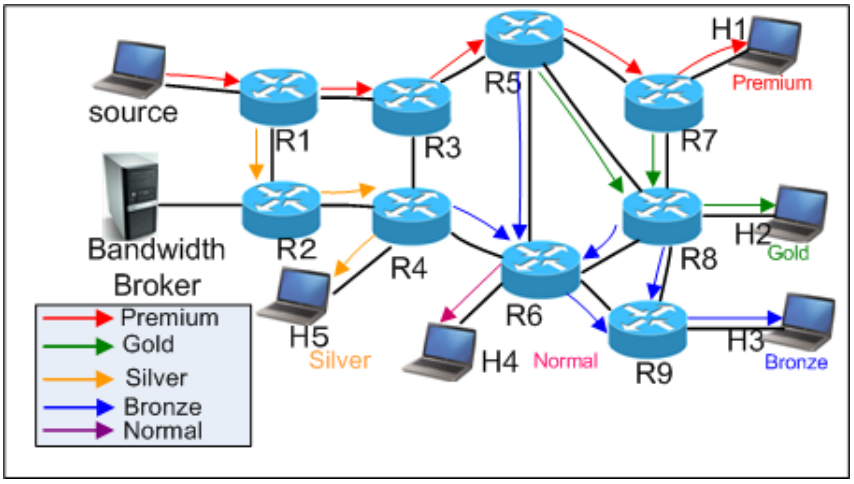


Figure 5.5: XCAST links exhibiting the allowable DSCPs

5.3.2 Previous work on XCAST QoS provisioning

While a lot of research in XCAST has been focusing on its design, implementation and performance, not so much has been done on QoS provisioning in XCAST. In[60],

Siregar et al, also acknowledge the lack of previous work in XCAST QoS research. However their work as well seems to only enumerate the available QoS routing techniques for unicast and multicast and then suggests that per-packet dynamic routing for unicast can be used in XCAST but they do not show how this dynamic routing should be adopted for XCAST. Nonetheless in [61], they propose a new modified IPv6 extension header they call *"IPv6 QoS header"* which contains a QoS value. The QoS value is to be calculated by routers based on the number of users underneath a router, total users requesting the video streams and the available priority levels. These values further depend on some probability assignment. The prioritization based on the QoS levels and how the probability values are calculated and allocated are however not explained.

Since the XCAST header is already complex for existing routers, adding another extension header for the purpose of QoS provisioning is likely to impact negatively on XCAST performance. We therefore propose to leave the XCAST header as specified in the XCAST RFC[3] but use DiffServ architecture for QoS provisioning.

5.3.3 Existing Multipoint DiffServ Solutions

In section 8.3 of RFC 5058 document[3], it is specified that an XCAST packet may contain a list of DSCPs so that the DSCP of the packet is assigned to the most demanding DSCP value from the list. However this specification does not take care of what happens when the XCAST packet is replicated at a branching router and the embedded list of destinations for a given set of next-hops no longer require the higher level of QoS that was embedded in the original IP header. The specification also does not put in place mechanisms to ensure that nodes receive the exact QoS which is entitled to their *"last-mile"* link. It is unlikely that all the receivers in the group will be willing to pay for the highest level QoS which they receive when the DSCP of the XCAST packet is assigned to be that of the most demanding from the list.

This leaves the current DSCP usage specification in XCAST susceptible to *collusion attack* and *DSCP confusion* problems explained above.

There are other research activities aimed at integrating DiffServ into multipoint communication environments. An example of the encapsulation based approach is DSMCast [124, 125] which like XCAST eliminates the maintenance of per-session state information in the routers. However DSMCast works by constructing a *Tree Encapsulation Header (TEH)* from the networks' multicast tree information and then encapsulates the *TEH* into the IP header. Additionally, in DSMCast, the TEH is limited to information on the edge-routers alone and not the actual receivers. This coupled with the fact that it relies on a multicast tree construction makes it not an applicable approach in integrating XCAST in DiffServ networks. Furthermore, DSMCast's approach is susceptible to collusion attack.

Other approaches still exist that aim at integrating DiffServ into multipoint communication by maintaining the state-information of the multicast trees in the core routers. This means that they have a major limitation in terms of scalability since maintenance of per-session state information increases the routers' load proportionately to the size of the multicast group. This also contravenes the DiffServ design philosophy of removing complexity from the core of the network and pushing all such loads to the edge network.

Jun-Hong Cui et al, proposed AQoSM [126] that uses the concept of aggregated multicast but decouples group and distribution tree concepts. AQoSM also proposes that admission control can be carried out on the level of aggregated trees instead of being done at individual links, thereby increasing efficiency due to the statistical multiplexing of multiple groups on a single tree. Being based on tree-group matching, AQoSM cannot be easily applicable in an XCAST scenario that depends on a unicast routing mechanism. Moreover, AQoSM depends on link-state collection which undermines the DiffServ design principle of statelessness at the core. Its dependence on a

single tree also opens it up to the collusion attack and DSCP confusion at the routers. AQoSM also introduces the installation of a new component, the *Tree Manager*, in a DiffServ network whose management and installation process is likely to add some overheads in the DiffServ network.

Other solutions such as Harmonic DiffServ[127], DAM (DiffServ Aware Multicasting)[128] and QMD (QoS-aware Multicasting in DiffServ domains) [129] have been proposed but each of them suffers from both the DSCP problems described above and the complexity involved in state-information maintenance. These approaches are therefore not easily adaptable for integrating XCAST in DiffServ networks.

We therefore propose a model for XCAST-DiffServ integration that not only seeks to solve the problems described above but also relies on the existing XCAST concepts and options with an enhanced resource management and admission control.

5.4 Scalable QoS-aware XCAST (QS-XCAST)

In order to integrate XCAST and DiffServ and to avoid the problems mentioned above, we propose a new approach called Scalable QoS-aware XCAST (*QS-XCAST*) and its implementation in IPv6 is codenamed *QS-XCAST6*. This approach is cognizant of receivers' QoS heterogeneity. It is based on dynamic DSCPs and an algorithm for "request-grant" QoS control between the source and the receivers.

Our approach is based on a typical DiffServ network (*like an example in figure 5.5*) comprising of all the essential DiffServ nodes such as:

1. A Bandwidth Broker
2. Core routers
3. Edge routers
4. End hosts (*senders and receivers*)

Using this approach and XCAST specifications in[3], the XCAST6 header will be as shown in figure 5.6. The DSCP class of the XCAST packet is therefore determined by the DSCP value in the outer IPv6 header.

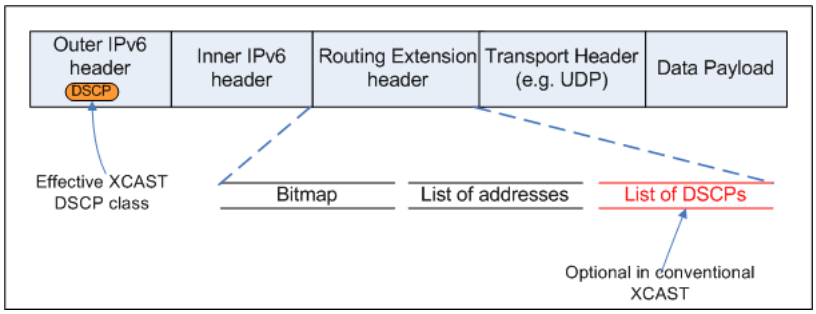


Figure 5.6: A block summary of QoS-aware XCAST header

5.4.1 Proposed Solution

1. *Dynamic DSCPs in the XCAST packet header*: We propose to extend the packet header processing in XCAST to allow for adaptive re-writing of the DSCP field at the branching routers. Therefore during packet replication in a branching router, the bitmap is used to determine which corresponding addresses in each copy of the packet the data is to be delivered and the respective DSCPs of these destinations are evaluated to obtain a new QoS precedence order. The DSCP fields of the packet copies where the new DSCP precedence differs from the original DSCP value are then updated with the new most demanding DSCP for each copy before the copies are transmitted to their next-hop routers. This mitigates the *DSCP confusion* problem and eliminates the vulnerability to *collusion attack*. Since the DSCP check is done at every branching router, we do not create two copies of a packet unnecessarily. Therefore if the current branching point is not the last branch on a path, the bandwidth is still saved.

2. *Receiver initiated QoS Requests*: We propose an approach where a receiver

can initiate the changing of its current QoS level by sending a *"QoS_change"* request to the Bandwidth Broker[130]. In a commercial network this can simply be implemented on a portal where customers choose various SLA levels which are eventually communicated to the Bandwidth Broker for consideration. The Bandwidth Broker knows the topology and capacities of the links within its domain. The Bandwidth Broker can therefore easily determine whether to grant a receiver the requested QoS level or reject the request but provide an acceptable alternative QoS level. The Bandwidth Broker (BB) maps the requested QoS level to its corresponding DSCP-PHB class and keeps a record of each receiver and its latest DSCP class assignment. This record is updated regularly within a definite control period (T) within which the receivers send feedback messages to the Bandwidth Broker in order to allow the Bandwidth Broker to keep-alive the receiver's record. When a receiver leaves an XCAST session, a timeout occurs i.e. the control period *(T)* expires before the receiver sends the feedback message to the Bandwidth Broker. The Bandwidth Broker then deletes the record of the receiver's current DSCP-PHB association.

Using this approach where the Bandwidth Broker is modified to also maintain a record of receivers and their latest DSCP class assignments (QoS level) has the advantage that when initiating traffic shaping and policing in an XCAST session, the ingress edge-routers find it easy to know the QoS requirements of all receivers of a given packet. They can therefore determine the appropriate DSCP precedence and mark the XCAST packet with the correct DSCP. This way not only do we solve the architectural conflict between DiffServ's *sender-driven QoS* and XCAST's *receiver-driven* QoS control approaches but also ensure that by maintaining the latest QoS requirements, we mitigate collusion attack problems. This algorithm is summarized in table 5.2

5.4.2 Algorithms for the proposed solutions

The extended XCAST processing algorithm

For solution (1) in section 5.4.1 above, we modify the XCAST processing algorithm at the routers according to *algorithm 5.1.* When used with the QoS network in figure 5.5, each of the hosts *(H1...H4)* end up receiving the QoS level requested as shown by the colour of their corresponding arrows in the figure. Even if a client attempts to downgrade their current QoS level, they cannot end up paying less for a higher QoS since the algorithm adapts to the latest QoS level as obtained from the embedded DSCP corresponding to each receiver at any branching point in the XCAST network.

Receiver initiated QoS level assignment

RFC2638 [130] defines the Bandwidth Broker (BB) as an agent in a DiffServ network that has some knowledge of an organization's priorities and policies and allocates QoS resources with respect to those policies. Admission control is therefore one of the key roles of the BB in a DiffServ network. The BB acts as a Policy Decision Point (PDP) in deciding whether to allow or reject a flow, whilst the edge routers act as Policy Enforcement Points (PEPs) for policing the traffic (allowing and marking packets, or simply dropping them). Therefore in solution (2) of section 5.4.1 above, we propose an algorithm that controls monitoring of changes in a receivers' QoS requirements thereby letting the receiver to request the BB to appropriately allocate QoS resources dynamically in a given DiffServ domain. The proposed algorithm is summarized in algorithm 5.2

5.5 Simulations and Results

We implemented our proposal in a simulation environment and used the simulation model to evaluate the proposal. The metrics on the receivers are throughput, average

Table 5.1: Dynamic DSCP assignment algorithm

On receiving an XCAST packet:

1. Obtain the current DSCP class of the packet. We call it the ***"original DSCP" class.***
2. Do a route table lookup and determine the next-hop for each of the embedded destination addresses.
3. Partition the set of destinations based on their next-hops.
4. Replicate the packet so that there is only one copy of the packet for each of the next-hops found in step 2 above.
5. Modify the bitmap for the list of destinations in each of the packet copies so that the bitmap in a copy to any particular next-hop is set only for the destinations that ought to be routed through that next-hop.
6. If the ***"original DSCP"*** class was found to be "Best Effort" (*DSCP value of "00000"*) then go to step 8, otherwise process the next step.
7. For each packet copy obtained from step 4 above:
   (a) Obtain the highest DSCP class from the list of embedded DSCPs for which delivery is to be done so as to determine *QoS Precedence*. This is the ***"new DSCP"*** class.
   (b) Update the DSCP field of the current packet copy to the *"new DSCP"* obtained in (a) above.
   (c) Continue to the next unprocessed packet copy.
8. Send the modified copies of the packet on to the next-hops.
9. If there is only one destination for a particular next-hop, the packet can be sent as a standard unicast packet to the destination (X2U).

Table 5.2: Receiver initiated QoS level assignment algorithm

If a receiver wants to change its current QoS level:

1. The receiver selects a preferred level (*higher or lower than its current QoS assignment*) from the list of QoS offered in the network.
2. The receiver communicates the new QoS level to the Bandwidth Broker(BB).
3. The BB verifies if there are adequate resources (e.g. bandwidth) along the receiver's path that can serve the requested QoS level.
4. The BB does a lookup in its map table for the entry of the receiver:
   (a) If the receiver's entry exists in BB's record and the resources are adequate for the requested QoS, the BB checks the DSCP mapping table for the requested QoS level and updates the receiver's DSCP class in the table.
   (b) If the receiver's entry exists in BB's record but resources are inadequate, the BB determines the acceptable QoS level and updates the receiver's table with the DSCP class matching this new acceptable QoS level.
   (c) If the receiver's entry does not exist in the BB's table, the BB creates a new entry for the receiver with an appropriate QoS level.
   (d) The BB notifies the receiver of the assigned QoS level.
5. The receiver sends an acknowledgment to the BB.
6. The BB notifies the edge routers of the latest policy changes.
7. Edge routers update their traffic shaping and policy rules to reflect the latest policies from the BB.

Table 5.3: Simulation Parameters

Simulation parameter	**Value**
Message size	1450 Bytes
Message frequency	50 ms
Queueing scheme	Shown in tables 5.4 and 5.5
Max Queue capacity	20 MB
MAC Tx rate	100 Mbps
Background Traffic frequency	50 ms

per-hop delay and link utilization. To investigate the impact of this model on DiffServ routers, buffer evolution was also investigated and used to infer the router's *traffic load* and *forwarding fairness* to other protocols. We also evaluate the model to verify that our approach eliminates the *collusion attack* problem.

5.5.1 Simulation Model

The proposed QoS aware XCAST is tested using OMNeT++ simulation tool[10, 11, 18]. OMNeT++ currently does not have inbuilt XCAST header encapsulation and routing models. However, previously[119], we gave a detailed description on how to integrate XCAST6 into OMNeT++. Additionally the basic DiffServ in OMNeT++ only has a simple classifier that classifies packets into only two classes but several other DiffServ components are missing. We therefore implemented our own full DiffServ architecture for OMNeT++. Using OMNeT++, we model an IP Television (IPTV) service provider network organized hierarchically such that the core routers form the provider's backbone network while the edge routers form the points where IPTV clients are hooked onto the network similar to the illustration in figure 5.7. Our model network comprises of 29 routers divided into 13 core routers and 16 edge routers (each edge router in its own subnet). Each edge router is connected to 5 hosts. One host is the source that sends data to all other remaining hosts in the entire network. The basic parameters are summarized in table 5.3.

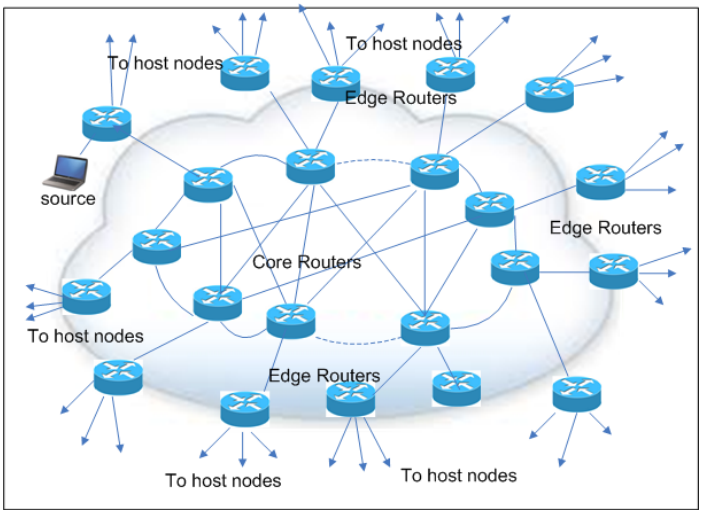


Figure 5.7: Model network for IPTV Service

In addition to IPTV (UDP) messages, a background non-XCAST TCP traffic is also run in the network and processed by all nodes. Interconnections between core routers are restricted to a degree of not more than five per router. For pricing, bandwidth allocation and Service Level Agreements (SLAs) purposes, the IPTV services are offered in six plans namely: *Super-platinum, Platinum, Gold, Silver, Bronze and Normal*. The service plans are mapped onto the DiffServ architecture's DSCP Per-Hop-Behaviours as shown in table 5.4. Bandwidth allocation threshold is 35% for EF class (Super-platinum), 55% for all AF traffic (*Platinum, Gold, Silver and Bronze*) since they use the same buffer model and 10% for Normal.

All receivers are assigned various DSCP classes selected from a pool of six DSCPs explained above. This is to conform with a typical IPTV subscription service in which each client is provided with the services at an agreed SLA. The DSCP is assigned statically at load time for each receiver. In section 5.5.1 we elaborate on the implementation of DiffServ metering and buffering. The receivers are distributed in different subnetworks. During the experiment, for each metric, we varied the number of receivers (*"group size"*) ranging from 10 to 75 hosts and ten simulation runs were conducted for each group size. Thereafter average values from all the runs were

Table 5.4: DSCP allocation and Buffering schemes

IPTV plan	**DSCP Class**	**Metering and buffering schemes**
Super-platinum	EF	Drop-Tail with a leaky bucket
Platinum	AF11	RIO[1] queue with token bucket
Gold	AF21	RIO queue with token bucket
Silver	AF31	RIO queue with token bucket
Bronze	AF41	RIO queue with token bucket
Normal	BE	RIO queue with token bucket

[1]RED (Random Early Detection) with distinction of In-profile and Out-profile packets

Table 5.5: DiffServ Metering and scheduling Parameters

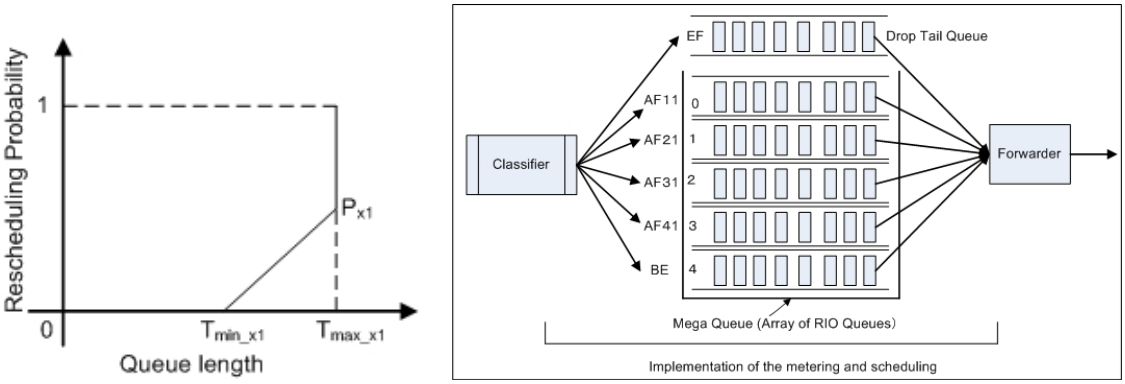
Queue model	**Queue Parameters**	**Parameter values**
Leaky Bucket	Token rate (Bytes/sec)	100,000
	Bucket depth(Bytes)	200,000
Token Bucket	Token rate (Bytes/sec)	100,000
	Bucket depth(Bytes)	400,000
RIO queue	Queue size (Bytes)	500,000
	Probabilities (P_{x1})[1]	0.5, 0.6, 0.7, 0.8, 0.9
	Thresholds (T_{Min_x1}, T_{Max_x1})[2]	0.9,1.0; 0.8,0.95; 0.7,0.95; 0.6,0.97; 0.5,1.0

[1]For AFx1, x=1,...,4. The last probability value is for BE.
[2]Semicolons separate min, max pair for each class. The last pair is for BE.

calculated.

DiffServ parameters

We implemented packet metering to check on in-profile and out-profile packets using leaky bucket[131] and token bucket[131] algorithms for EF and AF traffic classes respectively. This is because EF traffic should not allow for traffic burstiness while AF traffic can allow for burstiness. The EF buffer implementation was realized using a Drop-tail[131] queue with leaky bucket while RIO queues[131] with token buckets were used for all the AF classes and the BE class.



(a) RIO Algorithm **(b)** Metering and scheduling

Figure 5.8: Implementation of DiffServ in model routers

Parameters for these data structures are shown in table 5.5. RIO queue implementation was a little complex because we had to specify both *minimum* and *maximum* thresholds and the corresponding *"drop probabilities"* for all the AF classes and the BE class. In our implementation, dropping of packets is only done when the lowest, i.e BE, queue is full, hence the term, *"re-scheduling probability"* used in figure 5.8(a) instead of *"drop probability"*. We have an array of "RIO queues" as shown in figure 5.8(b). For a RIO queue at index *(i), (i=0,...,4)* in the array, if the queue length exceeds the minimum threshold T_{Min_x1}, *(x=1,...,4)*, the new *"AFx1"* packets are scheduled in a lower queue at index *(i+1)* of the array, with an increasing probability up to *Px1*. When the queue length exceeds the maximum threshold T_{Max_x1} and the current queue is not the last one in the array, all new *"AFx1"* packets are scheduled in a lower queue at index *(i+1)*. If the current index (i) is the last one in the array then the packets are dropped.

Path construction and packet delivery to receivers

XCAST does not depend on delivery tree construction. Instead, as specified in sections 3 and 4 of XCAST RFC document[3], XCAST packets always take the "right" path as determined by the *unicast routing table*. This implies that data delivery in an XCAST network is affected by the state of the routing tables of each interme-

diate node an XCAST packet passes through. In our model, the routing table of each node was initially constructed using a NETCONF[132, 133] based XML file. We added methods in the *"RoutingTable6" module* of OMNeT++ which parse the XML file based on NETCONF XML DTDs[132]. The NETCONF-XML file is filled with all possible paths within the mesh of nodes that forms the simulation model and OMNeT++'s *"RoutingTable6" module* loads these into the model at the initialization stage of the model. Additionally, the model IPv6 routers used in the simulation send out router advertisement *(RA)* messages at regular intervals to all their adjacent neighbours which the recipient routers then use to update their routing table information. This therefore ensures that at any given moment in time, the routing table of each of the model routers is up to date and the most optimal path is used to deliver both XCAST and unicast data to any particular receiver.

Receiver handling in XCAST6 and QS-XCAST6

XCAST6 is primarily designed for small group sizes hence group membership is usually limited. As specified in section 9.3.2.1 of the XCAST RFC document[3], the destination addresses are embedded within the IPv6 routing extension header. The IPv6 routing extension header length is expressed in 8-octets thus the theoretical upper bound of the number of XCAST6 destinations (group membership) is up to 127 receivers. In practice however, the possible number of receivers can be much less depending on the configuration of the MTU of routers in the network. This is because, the entire packet length also includes the data payload. The length of the data payload therefore affects how many destinations can be embedded within the IPv6 routing extension header without realizing IP fragmentation due to router MTU size limitations which is usually up to 1500 bytes. In our case, we embedded up to 75 destinations with a payload data of approximately 256 Bytes hence the total message length was 1450 Bytes.

Model verification for DiffServ functionality

We first verified QS-XCAST6 service differentiation using the model and a group size of 30 receivers and plotted the observations in figure 5.9. The model starts with 30 BE receivers and after 20 seconds 15 BE receivers change their QoS level to AF21 using the algorithm in *algorithm 5.2*. After 50 seconds the hosts are further changed such that we have 10 receivers for BE, AF21 and EF respectively. After 100 seconds we revert to the original state.

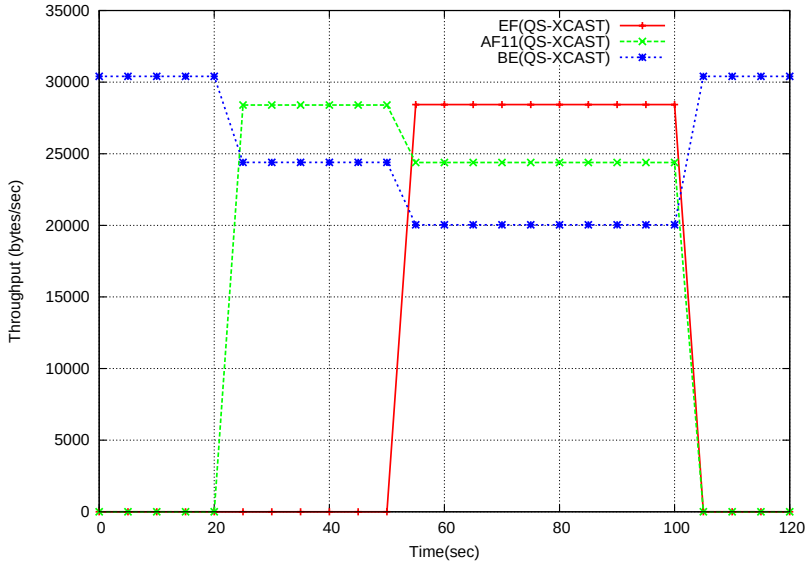


Figure 5.9: Service Differentiation verification using a group of 30 receivers.

Service differentiation is confirmed as required since the lower priority traffics are seen to reduce as expected whenever a higher priority traffic is injected into the network. This shows that each class is scheduled in its own independent queue. The BE actually rises after 100 seconds when we remove both EF and AF21 receivers and replace them with BE receivers. We then proceeded to investigate the model using various metrics as explained in section 5.5.1.

5.5.2 Average Throughput

The values of average throughput presented in figure 5.10 and those of average per-hop delay (figure 5.11 of section 5.5.3) were calculated by summing up the observed data in every corresponding DSCP class under each group size (*10...75*) and then dividing the total by the number of groups used (8 in this case). For example the EF values in figure 5.10 are averages of all EF observations for eight different group sizes varied from a group size of 10 to 75 receivers for both XCAST6 and QS-XCAST6.

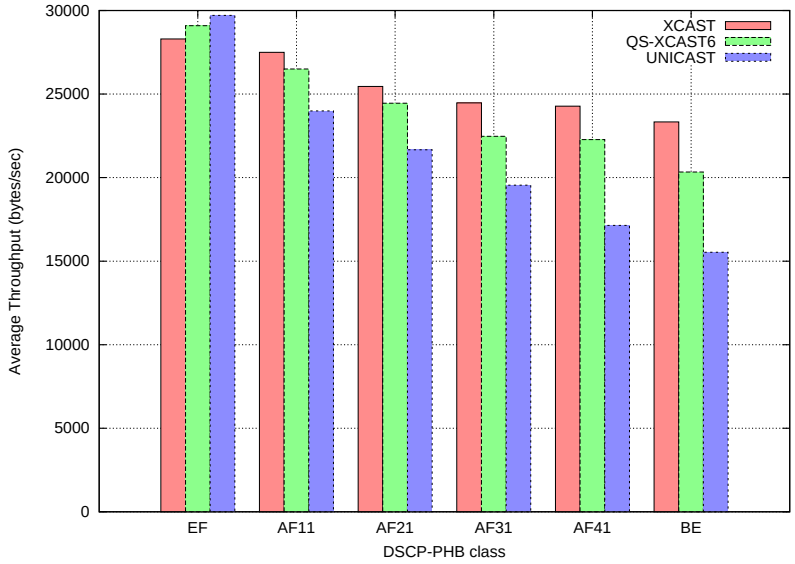


Figure 5.10: Comparative average throughput

In figure 5.10, in all cases except for the *EF DSCP* class, XCAST6 yields a higher throughput. This is because XCAST6 sends data to all the receivers in one packet and the DSCP of the XCAST6 packet is set to that with the highest priority from the list of DSCPs of the receivers hence even the lower priority DSCP classes get near-optimal treatment. However in XCAST6, the *EF* DSCP class suffers a little reduction in throughput and a longer delay when compared to the corresponding values in unicast and QS-XCAST6. This is because some of its resources are shared with the lower classes.

Unicast on the other hand, treats each of the DSCP-PHBs independently, therefore its *EF* class does not share any resource with lower DSCP classes. QS-XCAST6 finds a middle ground between these two extremes by ensuring that in as much as the data is delivered in one packet, each class still gets an appropriate treatment. Therefore the lower DSCP classes do not get near *Super-platinum services* that they are not paying for and the higher priority class does not suffer extensively due to the lower priority classes.

5.5.3 Average Per-hop delay

The method used for getting average per-hop delays for each DSCP class is the same as the approach used in calculating average throughput in section 5.5.2 above. As illustrated by figure 5.11, the average per-hop delay for all the DSCP classes reflects the disparity in DSCP treatment by each of the investigated protocols. XCAST6 still out-performs all the others hence the the lower priority DSCP classes still get very minimal delay compared to the same values registered by the other two protocols for each corresponding DSCP class. Figure 5.11 shows that for XCAST6, delays of all

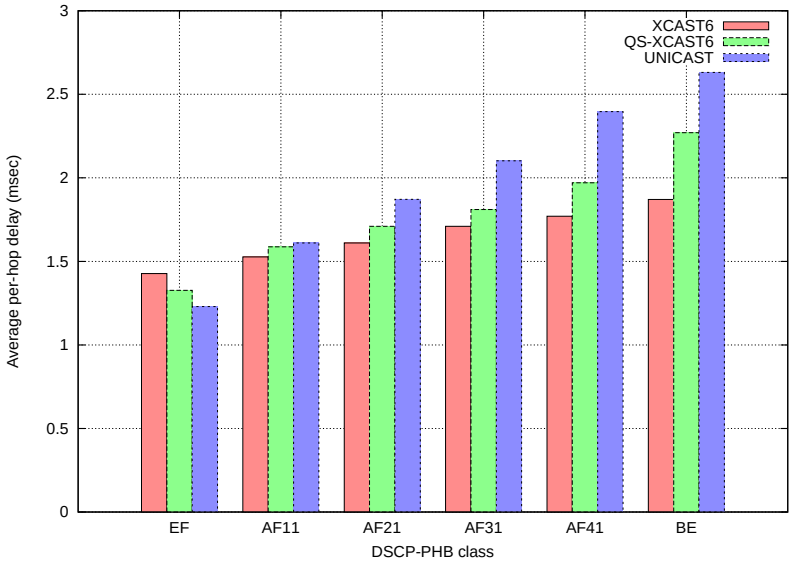


Figure 5.11: Comparative average per-hop delay

other classes tend to coalesce around that of *EF traffic* but for Unicast traffic each class is more distinct. QS-XCAST6 again finds an optimal middle ground for these cases ensuring that each class is treated fairly in accordance with its defined Per-Hop-Behavior. QS-XCAST6 achieves this by ensuring that during replication of XCAST6 packets at the branching routers, each copy is placed in its appropriate DSCP queue.

Elimination of DSCP confusion and Collusion Attack

Collusion attack exploits the possibility of low priority class of packets being treated at a higher priority in routers. In such case, the customers located downstream in the delivery path and are on lower priority SLAs collude to get better services but pay less. We use a sample network in figure 5.12 to show how this happens and how it is mitigated in QS-XCAST6. Each of the hosts (H1 to H6) is assigned DSCP classes that correspond to their SLAs as specified in their corresponding labels. The simulation is scheduled to run for 100 seconds then a measurement of throughput values on each host is taken. At this point, the DSCP values of hosts *H2, H3 and H5* are reduced to BE class (Normal plan) using the receiver initiated QoS SLA assignment algorithm in *table 5.2*. The model is then let to run for another 100 seconds before a second measurement of throughput values is taken. The results are plotted in figure 5.13.

As shown in figure 5.13(a), for XCAST6, all the hosts receive nearly the same amount of throughput irrespective of their DSCP classes. Throughput values for lower DSCP classes tend to coalesce around that of the EF class. This is because for XCAST6 even after replication, the packet copies still have the EF DSCP as the effective value. In QS-XCAST6 on the other hand, throughput values on each of the host is distinct and obeys the required DSCP precedence. This is because QS-XCAST6 dynamically re-writes the DSCP at every replication point according to the SLAs. Figure 5.13(b) shows that in XCAST6, even after changing the DSCP values of H2, H3 and H5 to BE ("Normal") after the first 100 seconds, they still continue to

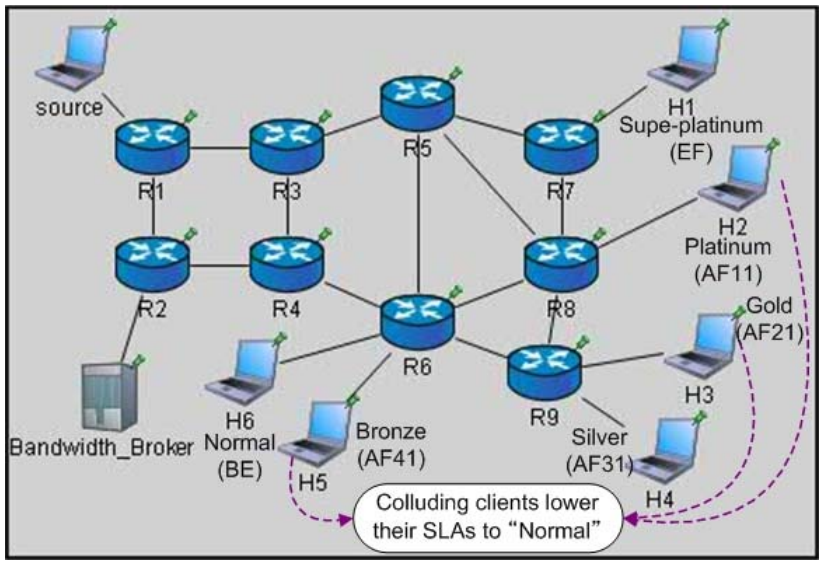
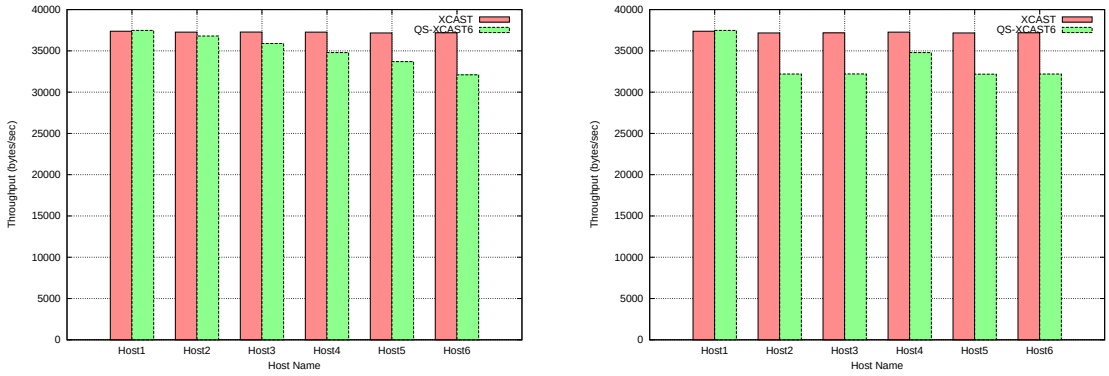


Figure 5.12: Collusion attack example in a group of 6 hosts.

receive the packets at EF level hence H2, H3 and H5 can easily collude to subscribe at BE level (and pay less) yet in the real sense they continue receiving near *Super-platinum* services. For QS-XCAST6, figure 5.13(b) shows that the throughput values of H2, H3 and H5 change to that of the BE level. This is because on replication, QS-XCAST6 places each copy of the packet in its appropriate queue thereby eliminating DSCP confusion which also eliminates any possibility of collusion attack.

5.5.4 Average Link Utilization

The link utilization statistics considers both the UDP traffic (*for IPTV simulation*) and the background TCP traffic in the network. The average values plotted in figure 5.14 were calculated from all group sizes (10 to 75) using the same approach explained at the beginning of section 5.5.2. Unicast link utilization was found to be so high; more than double for BE at about 68% and more than 90% for EF. Hence they do not compare well in the same graph with the XCAST6 and QS-XCAST6. We attribute this high link utilization by unicast to the fact that unicast has to send several successive packets for each receiver unlike XCAST that sends out the data



(a) Values with initial DSCP assignments.

(b) H2, H3 and H5 changed to (BE class).

Figure 5.13: Throughput values for a group of 6 hosts before and after class change.

to all receivers in only one packet. From figure 5.14 it is observed that QS-XCAST6 ensures greater bandwidth efficiency than XCAST6. For every DSCP class, XCAST6 utilizes more bandwidth on the final links (*"last mile"*) to each receiver than does QS-XCAST6. This is because by dynamically assigning DSCP values, QS-XCAST6 ensures that each host gets data at an agreed SLA even though the data to all receivers are transmitted in one packet. XCAST6 on the other hand delivers data at the highest priory DSCP since the DSCP fields of all the copies of the packet to all receivers are set to be that of the most demanding receiver.

QS-XCAST6 therefore proves to be an efficient method for IPTV data delivery compared to XCAST6 since for any given link with a definite bandwidth allocation, when using QS-XCAST6 the amount of bandwidth consumed is lower. This implies that for the constant bandwidth value on a link in the network, the remaining unconsumed bandwidth under QS-XCAST6 can still be utilized in connecting more clients than when XCAST is used. Therefore, from a service provider's point of view, QS-XCAST6 is very cost effective and serves all clients efficiently and reliably.

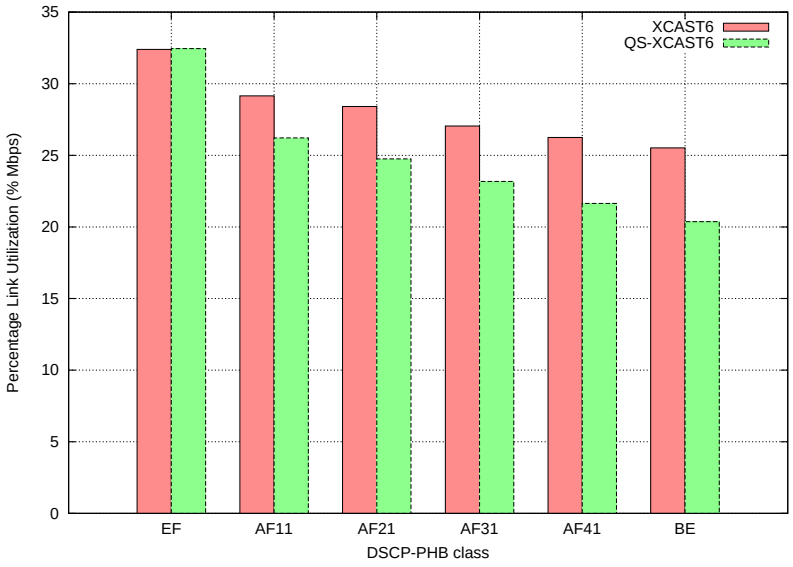


Figure 5.14: Comparative average link utilization

5.5.5 Effect of the Group Size

This comparison is done between XCAST6 and QS-XCAST6 and in order to enhance legibility, we plotted results of only 3 DSCP classes (*EF, AF21 and BE*) for each protocol.

Effect on Throughput

For both protocols, as the group size increases, throughput decreases marginally. This is illustrated in figure 5.15. XCAST6 registers a marginally higher performance than QS-XCAST6. QS-XCAST6 gives a clear distinction in throughput between the various QoS levels as noted especially by the wider difference in throughput values for *EF* and *BE* in QS-XCAST6. QS-XCAST6 thus gives the benefit of ensuring that each DSCP class gets its corresponding treatment as defined by the Per-Hop-Behaviour hence it introduces QoS awareness to XCAST6.

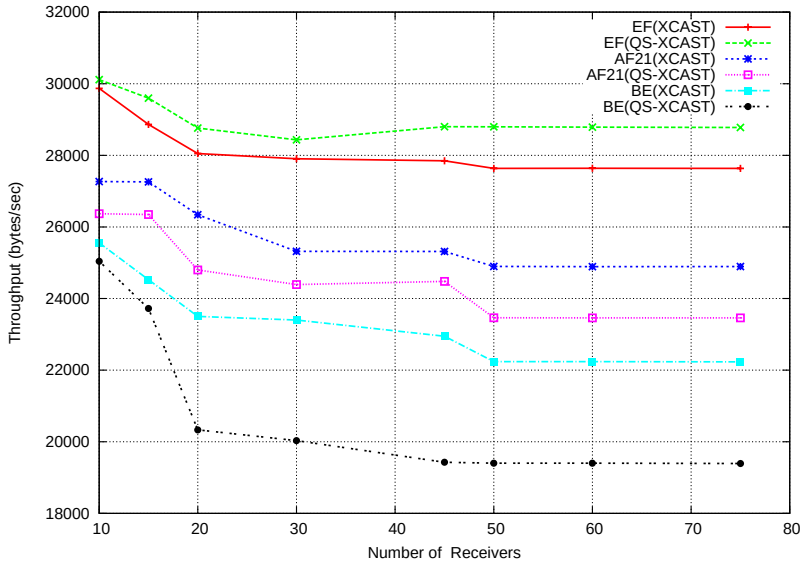


Figure 5.15: Throughput for varying group sizes

Effect on Average per-hop delay

Effect of an increase in the members in a group on the average delay for both protocols mirrors that of throughput. Once again, QS-XCAST6 shows a clear difference between the QoS classes in terms of their average per-hop delay. Hence each DSCP class is handled according to its respective priority level as shown in figure 5.16.

5.5.6 Scalability: Effects of the network scale

In instances like the Internet where QoS provisioning needs to span multiple DiffServ domains, in order to achieve an end-to-end allocation of resources across the separate domains, the Bandwidth Broker managing a domain will have to communicate with its adjacent peers. This allows end-to-end services to be constructed out of bilateral agreements as shown in figure 5.17.

An end system initiates a request for service to its domain's Bandwidth Broker (BB) with a fully-specified destination address of the intended receivers of the service. The local Bandwidth Broker realizes that the request is for a host in another DiffServ

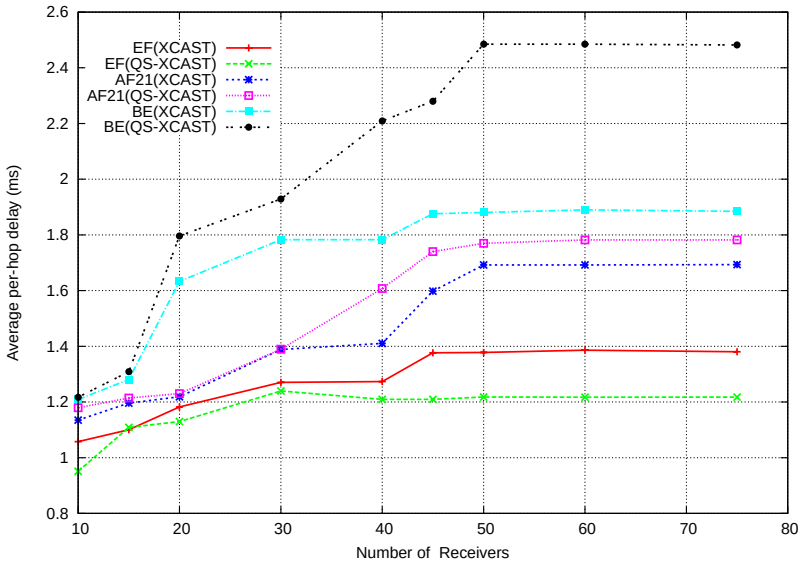


Figure 5.16: Average per-hop delay for varying group sizes

domain and requests the service to another domain. In the transit domain, this is in effect a pipe to another domain where the destination host is located. The Bandwidth Broker (BB), of the host's domain receives the request and liaises with the host to determine its QoS level. Then the request is sent back to the original domain via the transit domain. The Bandwidth Broker of the end system that initiated the request forwards the verified QoS requirements of the intended recipient and then the service delivery can begin. In this test, we compare cases where this request to initiate a QoS service delivery over multiple DiffServ domains is done via XCAST6 (*same for QS-XCAST6*) and when it is done via unicast for varying number of intended recipients.

The request messages are referred to as *Resource Allocation Requests (RAR)* and their answer messages as *Resource Allocation Answer (RAA)*. Since at this stage the *RAR* and *RAA* messages are of the same priority (*DSCP value*), it does not matter whether we use QS-XCAST6 or XCAST6. Figure 5.18 shows the time it takes to receive the *RAA* message so that data delivery can begin for various group sizes (*for unicast we track the* n^{th} *RAA for a group of n members*). Since XCAST6 out-

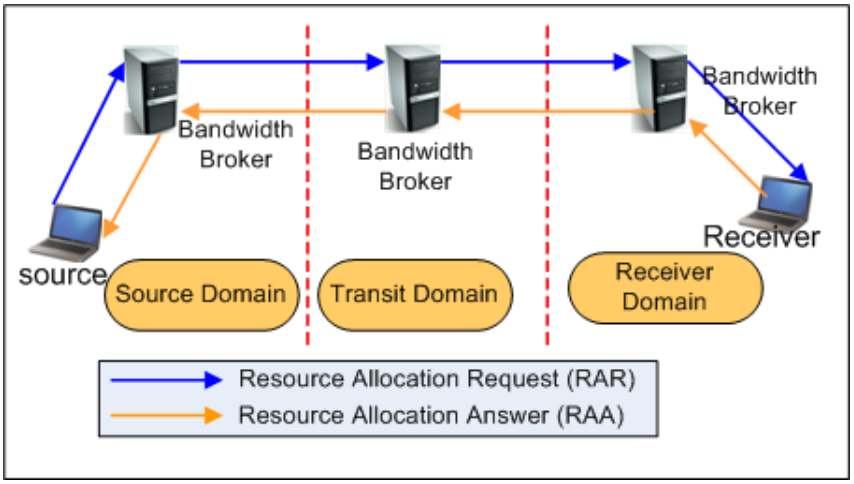


Figure 5.17: QoS Provisioning in multiple DiffServ domains

performs unicast, integration of XCAST would be really beneficial for QoS aware implementations spanning multiple DiffServ domains.

5.5.7 Impact on DiffServ Routers

In DiffServ, the edge routers act as policy enforcement points. They typically classify incoming packets into pre-defined aggregates, meters them to determine compliance to traffic parameters, marks them appropriately by writing (or re-writing) the DSCP values and shapes (buffers the packets to achieve a target flow rate) or drops the packets in an event of congestion. We assessed the impact of our proposal on DiffServ routers by analyzing the buffering patterns of ingress edge routers and core routers under both XCAST6 and QS-XCAST6 traffic. We plotted the results as shown in figure 5.19.

The number of buffered packets under XCAST6 remains higher than those under QS-XCAST6. For both protocols, buffering in edge routers is less than that of core routers. Interpretation of buffer evolution can be two fold in terms of: the impact on router's traffic load and that of forwarding fairness to other protocols.

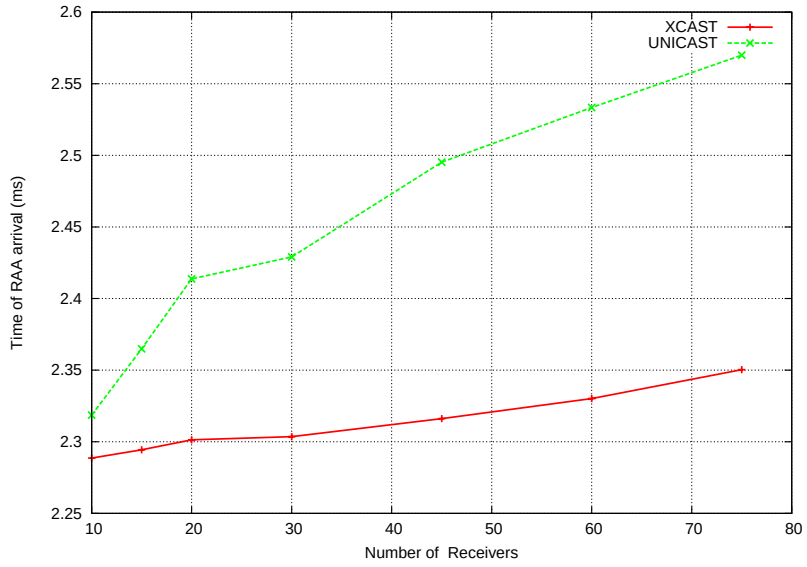


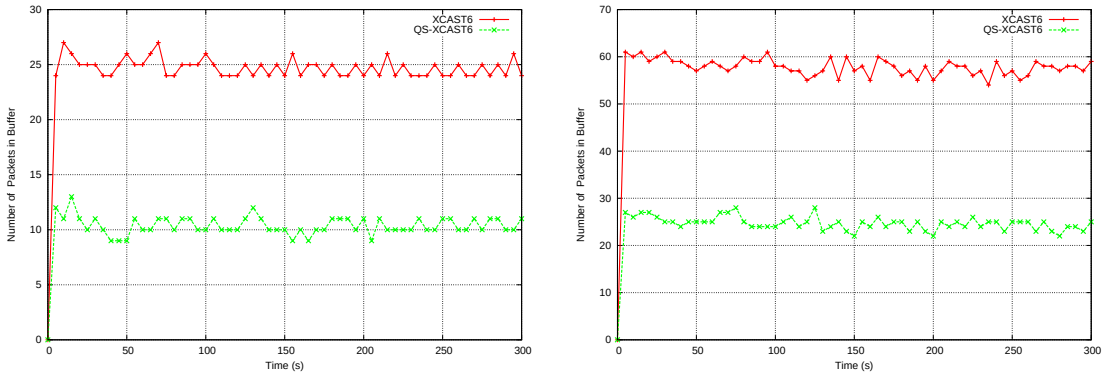
Figure 5.18: Time taken to receive Resource Allocation Answer (RAA) message

Impact on Traffic Load

Traffic load can be defined as the ratio of incoming traffic to outgoing traffic in a router[134]. This way, buffering can be assumed to be directly proportional to a router's load size. Interpreting results of figure 5.19 in this context, we can conclude that QS-XCAST6 adds less traffic load than XCAST6 to a DiffServ router.

Impact on Forwarding Fairness to other protocols

If we interpret the buffer evolution pattern in figure 5.19 in conjunction with observations made in figures 5.10 and 5.11 where XCAST6 registers higher throughput and lower average per-hop delay compared to QS-XCAST6, we conclude that most of the buffered packets are for the background TCP traffic running in the model. This is clearer if we consider that the background traffic here represents the standard (*non-prioritized*) packets thereby being buffered in the BE queue. This buffer evolution pattern points to the fact that QS-XCAST6 is likely to realize better *"packet forwarding fairness"* than XCAST6 between real-time(prioritized) and standard (non-



(a) Edge routers' buffer evolution **(b)** Core routers' buffer evolution

Figure 5.19: The impact of XCAST6 and QS-XCAST6 on DiffServ routers.

prioritized) traffic in the network since its buffering effects on the standard traffic is less than that of XCAST6.

5.5.8 Other effects of our solution

1. *Header size and packet length*: Embedding the list of destinations and their corresponding DSCP classes in the IP packet header increases the header size. This in turn increases possibilities of IP fragmentation problems. As observed in our case where according to XCAST RFC document[3], we can embed up to 127 unicast addresses but in practice this number is affected by the length of the payload data too. Therefore we were able to simulate with up to 75 destinations without breaching the router interfaces' MTU limits. To mitigate IP fragmentation problems, QS-XCAST6 could be combined with the G-XCAST solution[135] to ensure the group sizes are small enough to pass within the MTU limits.

2. *DSCP field modification*: In order to reduce overheads related to DSCP processing, we limit the DSCP processing to simple comparison and update is done only on the DSCP field of the IP header and not on the entire list of the embedded DSCPs. The alteration of the DSCP field however might bring challenges

in situations where IPsec is to be used with QS-XCAST.

3. *Feedback implosion*: The communication of bandwidth requirements by receivers to the source needs to be done in a way that ensures that the source is not overwhelmed by such messages. Approaches such as exponentially distributed timers[136] can be implemented between the receivers and the source. This potentially avoids the feedback implosion and enhances the scalability of bandwidth requirements handling at the source even for larger groups.

5.5.9 Further Discussion on Practicality

QS-XCAST6 can be used to improve on flexibility and efficiency of bandwidth allocation. If providers use QS-XCAST6 for delivery of real-time traffic, they can take advantage of the fact that the various IPTV plans have varying bandwidth thresholds to economize on bandwidth utilization on any given link. As an example, we use bandwidth allocation thresholds as shown in the second column of table 5.6 for each DSCP class and define a new ratio called *"QS-XCAST Gain", (QXG)* to determine the savings that can be realized using QS-XCAST6. We also use a scenario where the allocation is to be done for 100 MB on a given link. *"QS-XCAST Gain", (QXG)* is defined using formula (5.1) as the gain for different traffic loads in terms of the bandwidth required by XCAST6 (*Bw_XCAST6*) divided by the bandwidth required by QS-XCAST6 (*Bw_QSXCAST6*) to achieve the same QoS Service Level Agreement.

$$QXG = \frac{Bw\_XCAST6}{Bw\_QSXCAST6} \tag{5.1}$$

For the purpose of this illustration, we calculate *QS-XCAST Gain* as shown in table 5.6 based on the link utilization results observed in figure 5.14 (section 5.5.4).

From a bandwidth of 100 MB on a link, if we apply the indicated bandwidth allocation policy for each DSCP class and use QS-XCAST6, we obtain an estimated

Table 5.6: Bandwidth Allocation for XCAST6 and QS-XCAST6 using 100MB

IPTV plan	**DSCP Class**	**Bandwidth Allocation (%)**	**QXG**	**QS-XCAST6 Bandwidth Estimate**
Super-platinum	EF	35	0.999	35.04
Platinum	AF11	18	1.117	16.11
Gold	AF21	14	1.148	12.20
Silver	AF31	12	1.167	10.28
Bronze	AF41	11	1.213	9.07
Normal	BE	10	1.253	7.98
Total		100		90.68

bandwidth utilization of 90.68 MB saving the rest due to QS-XCAST Gain as shown in table 5.6. The saved bandwidth can be used for deploying other services hence ensuring efficient and economical bandwidth utilization. However, we note that the exact bandwidth economy realized by any IPTV service provider while using either XCAST6 or QS-XCAST6 is dependent on the ratio of the real-time traffic to the standard traffic in the network and also on the threshold allocations per given DSCP class. The lower the ratio of real-time to standard traffic, the higher the QS-XCAST6 gain and hence the better the performance of QS-XCAST6.

QS-XCAST6 effect on Router Performance

Having run this on a simulation environment, we have not been able to investigate the impact of QS-XCAST6 on the router's CPU and Memory utilization. However in section 5.5.7, we investigated possible impact on a DiffServ router's buffering pattern for both edge and core routers where we observed that QS-XCAST6 has lower buffering effects than XCAST6. If applying QS-XCAST6 impacts negatively on a router's forwarding performance then the router will support less aggregate throughput. However, today's commercial routers typically implement DiffServ Per-Hop-Behaviours in

ASICs[137] thereby ensuring that there is no forwarding penalty associated with DiffServ implementation.

5.6 Conclusion

We have proposed a model for providing Quality of Service in XCAST using DiffServ. We explored the various challenges that complicate integration of XCAST and DiffServ. We then showed how to overcome the challenges and tested our proposal by simulation using OMNeT++. Our proposed QoS-aware XCAST6(QS-XCAST6) proves to be very efficient when it comes to bandwidth utilization, out-performing the current XCAST6 thereby proving to be very conducive for offering services such as IPTV using XCAST6. QS-XCAST6 also impacts less on DiffServ router buffering patterns compared to XCAST6, showing a better fairness to non-priority traffic when compared to XCAST6. QS-XCAST6 only registers a slight drop in throughput compared to XCAST6 but ensures that all clients get the services at the agreed SLA. XCAST6 on the other hand allows even lower priority clients to enjoy better services than what they are paying for thereby ending up consuming more bandwidth than ought to have been the case. XCAST6 also leaves the network vulnerable to *collusion attack* which QS-XCAST6 totally eliminates. We therefore find that QS-XCAST6 is preferable for commercial service provision like in IPTV scenarios.

Chapter 6

Integrating XCAST with LISP

6.1 Overview

A number of concerns about the state of the current Internet have resulted in the research in an architecture for the Future Internet. These concerns include the difficulties with regards to scalability of the routing system and the impending exhaustion of the IPv4 address space. Separating the Identity and Location spaces on the Internet has been one of the proposed solutions to these problems. In this area, Location Identifier Separation Protocol (LISP) has been the most successful so far. This chapter therefore aims at realizing deployment of XCAST even in the Future Internet where locator and identifier spaces have been separated. To achieve this, we explore the Integration of XCAST with LISP.

6.2 Introduction

The near exponential growth of users connecting to the Internet and the explosion of the availability of mobile devices such as cell phones and tablets which are increasingly used by mobile users to connect to the Internet have fueled research into the Future Internet architecture. This is because the current Internet already faces

serious problems with regards to scalability of the routing system and the impending exhaustion of the IPv4 address space. Research in this area was actually touched off by the Internet Architecture Board(IAB)'s workshop on Routing and Addressing held in October 2006. Since the IAB workshop, several proposals have emerged that attempt to address the concerns expressed both at the workshop and in other forums[138, 139, 140, 141, 142, 143]. One common factor with all these proposals is that they aim at separation of locators and identifiers in the numbering of Internet devices. This is commonly referred to as *"Loc/ID split"*. However in this research, we focus on one Loc/ID split proposal known as the Locator/Identifier Separation Protocol (LISP)[9].

6.2.1 The Locator/Identifier split concept

The current Internet routing and addressing architecture combines two functions: *Routing Locators (RLOCs)*, which describes how a device is attached to the network, and *Endpoint Identifiers (EIDs)*, which defines ***"who"*** the device is, in a single numbering space, the IP address. This results in an *"overloading"* of functions in the IP address space. Researchers therefore argue that this *overloading* of functions results in forcing a number of constraints on the address space and makes it impossible to build an efficient routing system without forcing unacceptable constraints on end-system use of addresses.

Loc/ID split therefore proposes that these functions be split in such a way that EIDs and RLOCs use different numbering spaces. This is envisioned to offer several advantages including improved scalability of the routing system through greater aggregation of RLOCs. This split in turn will facilitate improved aggregation of the RLOC space, implement persistent identity in the EID space, and, possibly increase the security and efficiency of network mobility. These are basically the foundation onto which the LISP protocol is built.

6.2.2 Implementing the Locator/ID Separation

Two distinct approaches have so far come up on how this split can be achieved. These are *map-and-encap* and *Address re-writing*. The former works seamlessly for both IPv4 and IPv6 while the latter works only in IPv6 address space. We briefly discuss each of these approaches in the sections that follow.

Map-and-encap

Map-and-encap is a Loc/ID split scheme in which the handling of a packet is determined by whether the EID to which it is addressed is within the same domain as the source. When a source sends a packet to the EID of a destination that is outside its own domain, the packet traverses the domain infrastructure to a border router (or other border element). The border router maps the destination EID to an RLOC that corresponds to an entry point in the destination domain. Therefore an *EID-to-RLOC mapping system* is needed. This phase is the *"map"* phase of map-and-encap.

After a success in the mapping phase, the border router then prepends a new header(*"the outer header"*) to the packet it just received from a sending host(becomes the *"inner header"*). The source and destination addresses of the inner header are EIDs while those of the outer header are mostly RLOCs. The source address of the outer header is the RLOC address of the border router while its destination address is the RLOC returned by the mapping infrastructure. This process is called *encapsulation* and this phase is the *"encap"* of the *map-and-encap* model.

When an encapsulated packet arrives at the destination border router, the router *decapsulates* it to get both the inner and the outer headers. The outer header is usually destined to the border router so it will be discarded as the router sends the inner header to its destination. In this regards, EIDs need to be routable, usually within their local domain.

An advantage of the Map-and-encap schemes is that they do not require host

changes or any changes to the core routing infrastructure. The other one is that they work with both IPv4 and IPv6, while retaining the the original source address. However, there have been arguments[144] as to whether or not the encapsulation and decapsulation processes pose significant overheads.

Address Rewriting

This is the seond approach ot Loc/ID split and applies only to IPv6 address space. It aims at taking advantage of the 128-bit IPv6 address length in the IPv6 address space and use the top 64 bits as the routing locator (*"Routing Goop", or RG*), and the lower 64 bits as the endpoint identifier.Using this scheme, when a packet destined to another domain is sent by a host, the source address contains its identifier (frequently a *IEEE MAC address*) in the lower 64 bits, and a special value (meaning unspecified) in the RG. On the other hand, the destination address contains the fully specified destination address (RG and EID). When the packet arrives at the border router(egress router) of the local domain , the source RG is filled in (forming a full 128-bit address), and the packet is routed to the remote domain. When the packet reaches the destination domain (at the ingress router of the remote domain), the destination RG is rewritten with the unspecified value, ensuring that the host does not know what its RG is.

An outstanding weakness of this proposal is that besides its complexity, it requires changes on the network hosts. Further it works only on the IPv6 address space. Therefore our effort is focused on LISP that uses Map-and-encap and offers several advantages including incremental deployment in the existing network infrastructure.

6.3 More on LISP Protocol

We first introduced LISP in section 2.7. LISP aims at being a simple, incremental, network-based map-and-encap protocol for implementing the Loc/ID split using

EIDs and RLOCs. LISP requires no changes to host stacks and no major changes to existing network infrastructures. The greatest advantage of LISP comes in the mobility environment where it has the key advantage of improving site multihoming and decoupling of site addressing from provider addressing thereby reducing the size and dynamic properties of the core routing tables.

6.3.1 LISP with Static Nodes

First we note that LISP is "IP-protocol agnostic", therefore RLOCs can either be IPv4 or IPv6 addresses used for routing through transit networks. In order to reach a host, identified by its EID, one must first find the current Routing locator (RLOC) of the host. This is usually an RLOC address of the border router in charge of the end-host's (receiver's) domain. Once the RLOC associated with the EID is discovered, packets with headers from the EID numbering space are encapsulated in a second header from the RLOC space (*usually the RLOC address of the border router within the sender's domain*), and are routed to the destination, where the LISP header is removed before delivering packet to the destination host.

In LISP terminology, the border routers that perform packet encapsulation and decapsulation are referred to as *Tunnel Routers*. They are the entry and exit points of the tunnels that help tunnel LISP packets through non-LISP domains. We therefore have at least two types of Tunnel Routers, the *Ingress Tunnel Router (ITR)* and an *Egress Tunnel Router (ETR)*. However in other instances we also have a *Proxy Tunneling Router (PTR)*. LISP also introduces a publicly accessible Mapping System that is designed to serve the EID-to-RLOC mapping information. This is achieved by various technologies, the most common of which is the LISP-Alternate architecture (LISP-ALT).

6.3.2 LISP-Mobile Node

In LISP architecture, host applications bind to host's EID, which is used as the address for transport connections. The fact that host identifiers (EIDs) are separated from their locators (RLOCs) and also noting that applications bind only on EIDs, enable seamless endpoint mobility. This is because even in mobility scenarios, applications are able to bind to a permanent address, the host's EID. A mobile node can therefore change locations many times during an ongoing network connection without terminating the already established communication links. Each time, the LISP tunnel routers will encapsulate the packets to the new RLOC, preserving the connection session from breaking. LISP Mobile Node (LISP-MN)[89] architecture therefore builds on this foundation to come up with a robust mobility platform within the LISP architecture.

In LISP-MN, a mobile node (MN) is typically statically provisioned with an EID that it uses for all its connections. Each mobile node essentially behaves as a LISP site in accordance to the LISP architecture. Packets (except for management protocols such as DHCP) are LISP encapsulated by the mobile node, and routed based on the RLOCs to the destination site. In the event of a handover, MN receives a new RLOC and updates its EID-to-RLOC mapping in the associated mapping system to maintain reachability at its new location. The LISP-MN architecture leverages four existing LISP components:

i. A Mapping System,

ii. LISP-MN,

iii. LISP Internetworking components

iv. LISP NAT-traversal

The LISP-MN protocol is then, best understood, as the concatenation of three different phases:

i. Registering EID and obtaining an RLOC

ii. Signaling EID-to-RLOC bindings and transmitting data-packets

iii. Handover.

6.3.3 Multicast in LISP

Finally in this section we look at how multicast processing is handled in LISP protocol. In IP multicast, a multicast group address is an identifier of a grouping of topologically independent receiver host locations. A Multicast address itself does not determine the location of the receiver(s). Instead, the locations of receivers are determined by the multicast routing protocols and the network-based multicast state information created by the protocols. In the LISP context, a multicast group address is both an EID and a RLOC. Therefore no specific action is necessary for destination addresses; a group address that appears in an inner IP header (built by a source host) is used as the destination EID by an ITR as a destination address when it LISP-encapsulates the packet. This implies that the ITR uses the same group address as the destination RLOC. The source RLOC, as is usually the case, is the ITR IP address (that is, one of its RLOCs).

On the receiving side, Protocol Independent Multicast (PIM)[145] is used to translate the source-address Join/Prune messages from RLOCs to EIDs when multicast packets are forwarded by the ETR. However, in contrast to the unicast case (where a Map Request is sent by the ITR at forwarding time), a Map Request can be sent when the multicast tree is being built.

6.4 Why XCAST on LISP

The intention of XCAST researchers is to have it deployed for use in the real world. Despite the deployment challenges, most of which have been highlighted in chapter

three, it is important that XCAST research also looks ahead into deploying XCAST in the future Internet. For this reason, we look at finding ways of integrating XCAST with LISP since, as the current Internet scalability challenges continue to be a concern, Future Internet architecture solutions like LISP will be drawing closer to universal adoption over the Internet.

Another reason for LISP-XCAST integration is that both protocols share the objective of incremental deployment without significantly altering either the current hosts on the Internet or the Internet infrastructure itself. Just like XCAST Routing Engines can be incrementally deployed over the Internet, so are the LISP Tunnel Routers.

Finally, LISP Mobile Nodes (LISP-MN) offer simpler mobility solutions and upon successful LISP-XCAST Integration, XCAST mobility will certainly be simplified using XCAST aware LISP-MNs. Since the Future Internet is going to be characterized by massive mobility, this integration definitely positions XCAST for use even in the Future Internet.

6.5 LISP-XCAST Integration approaches

We identified three approaches that can be applied to realize successful integration of LISP and XCAST. In this section we shall be looking at these approaches in details. We finally settled on one of them and implemented on a network simulator (OMNeT++) for the purpose of evaluation. The three approaches are classified depending on what changes have to be made into the LISP infrastructure. These include:

i. No Modification to LISP infrastructure

ii. With Modification on LISP infrastructure

iii. With an XCAST Group Server and Group Information Maintained in xTR

Table 6.1 is a key that explains the different packet structures under each approach. It helps understand the packet structures used in figures 6.2, 6.3 and 6.4 together with their corresponding tables 6.2, 6.3 and 6.4. In the next sub-sections, we discuss each of these approaches with a view to highlighting the required changes to the LILSP infrastructure and the XCAST header structure and also how XCAST send and receive processes are undertaken in each case.

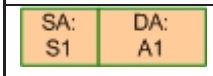
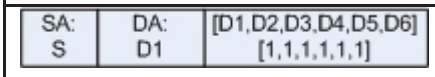
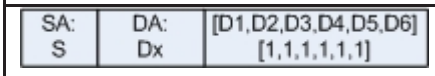
6.5.1 With No Modification to LISP infrastructure

The first approach is the one in which LISP architecture remains unaffected. The architecture remains as currently specified in the LISP Internet draft document[9] but modification is done to the XCAST Implementation on end hosts. There are two schools of though in this approach.

The first one proposes that owing to the fact that LISP uses *map-and-encap* approach that involves prepending of an outer LISP header to the packet, there is a need to deprecate the semi-permeable(outer)[3] header of XCAST. This reasoning is plausible because if one of the IPv4/IPv6 headers in XCAST is not deprecated, integrating XCAST with LISP will result in a packet with three consecutive IPv4/IPv6 headers as shown in figure 6.1. This will result in a packet that will likely be *"too big"* for the ordinary configuration of router MTUs (usually 1500Bytes) and will result in IP Fragmentation problems. Typically in this approach, traffic class and flow label fields of outer header will be removed and taken to what is now *"inner IP header"* in XCAST. The source address of inner header to remain as it is and the destination address will be picked from one of the list by the sender. This will often be any one destination whose corresponding bitmap is set to 1.

In the second school of thought we argue that deprecating the semi-permeable header would be detrimental since it effectively implies that the resultant XCAST packet can only be transported in networks that have LISP Tunnel routers (xTRs).

Table 6.1: Explanation of LISP-XCAST Packet structure

Packet Example	Explanation
	A LISP header that an ITR pre-pends to the XCAST packet. It forms the "outer" IP header. This if from ITR with RLOC S1 to an ETR with RLOC A1.
	An XCAST packet header for option 1. Source Addresses is EID of the sender and destination address is an EID of an XCAST aware host among the receivers. The host in the DA field will process the packet and send copies to other hosts in the list where the corresponding bit in the bitmap is set to 1.
	An XCAST packet header for option 2. Source Addresses is EID of the sender and destination address is an EID of any of the hosts in the destination list. Since xTRs are also XCAST aware, the address in the DA does not matter. They use the embedded list in doing map-and-encap and decapsulation operations.
	An XCAST packet header for option 3. Source Addresses is EID of the sender and destination address is a "well known" EID obtainable from the group server. The map server has a list of xTRs that understand the "well known" group EID. Therefore an ITR will simply send the encapsulated packet to an ETR that understands the "well known EID, XCST_HOSTS".

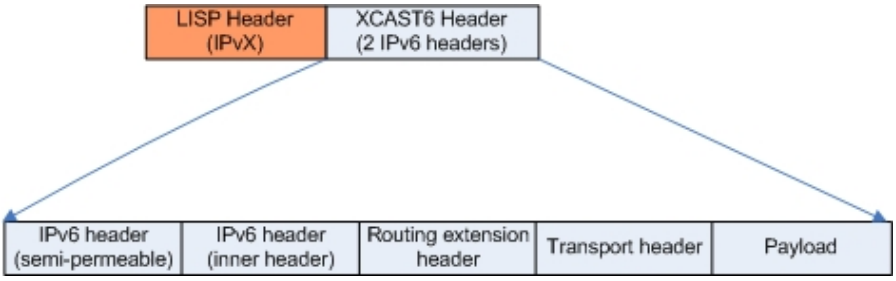


Figure 6.1: XCAST-LISP header structure with option one.

Without the LISP xTRs, the current Internet routers will fail to identify the XCAST packet effectively. Irrespective of the school of thought employed above, XCAST sending and receiving will be handled as explained in the section that follows using an XCAST6 example.

XCAST Sending Process in this scenario

i. A source end-host gets end-host EIDs for all XCAST receivers from a group server

ii. The source creates an IPv6 packet with its own EID as the source address.

iii. The source embeds all receivers' EIDs in the routing extension header.

iv. The source also selects one of the receivers' EID and puts it as the destination address.

v. The source then sends the XCAST packet to an ITR (this is an improved XCAST header where the semi-permeable header has been removed).

vi. The ITR does map-and-encap and sends the XCAST packet to the ETR of the receiver in the destination address.

vii. The target ETR decapsulates the packet and sends it to the receiver whose EID is in the destination field.

viii. On receiving the XCAST packet the host checks how many other receivers should get the packet.

ix. It resets the bit corresponding to its own EID in the bitmap list.

x. Makes a copy of the packet which it sends to its own Transport layer

xi. It then modifies the destination address of the remaining copy to one of the EIDs whose corresponding bit is still set in the bitmap.

xii. Finally it sends out this copy to its own ITR for further delivery

Figure 6.2 uses a simple XCAST network with a single source and 6 receivers (D1,D2,...,D6), to illustrate how this method can be realized. The greatest weakness of this approach is that it leads to cascaded delivery of XCAST data (a hop-by-hop delivery style). Therefore this approach might not be not be good for real-time multimedia delivery since the time difference between when the first receiver receives a packet and when the last receiver in the cascaded deliver path does so can greatly jeopardize the conversation quality. This is not acceptable in some situations like in videoconferencing which is a key application area for XCAST.

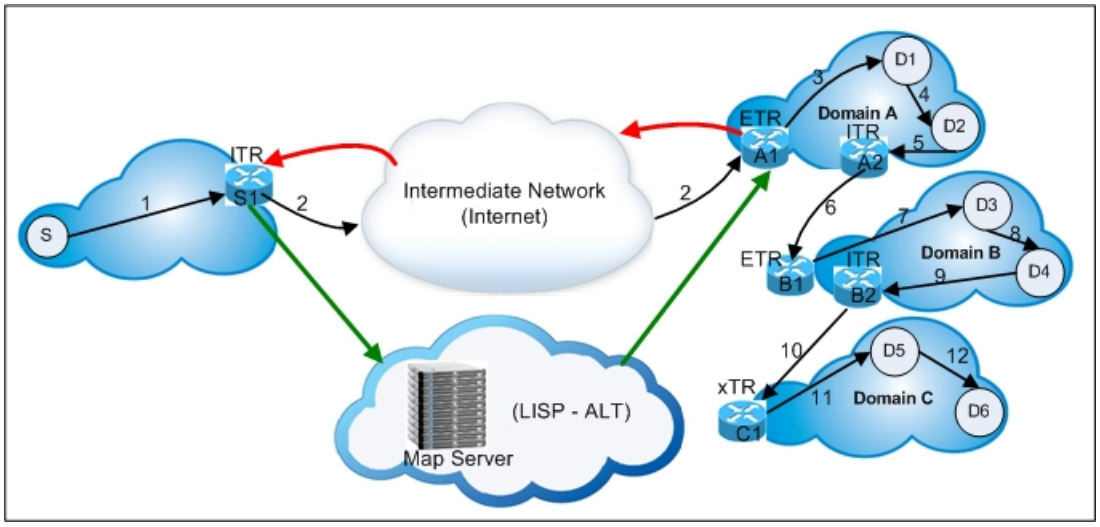


Figure 6.2: Sample network where no change is done to underlying LISP architecture

In figure 6.2, the following conventions have been used:

- The green lines represent the probe packets for requesting the RLOC of the ETR from the map server possibly through a LISP ALT infrastructure.

Table 6.2: How an XCAST packet changes in figure 6.2

Flow step	Corresponding packet structure
1	SA: S | DA: D1 | [D1,D2,D3,D4,D5,D6] [1,1,1,1,1,1]
2	SA: S1 | DA: A1 | SA: S | DA: D1 | [D1,D2,D3,D4,D5,D6] [1,1,1,1,1,1]
3	SA: S | DA: D1 | [D1,D2,D3,D4,D5,D6] [1,1,1,1,1,1]
4	SA: S | DA: D2 | [D1,D2,D3,D4,D5,D6] [0,1,1,1,1,1]
5	SA: S | DA: D3 | [D1,D2,D3,D4,D5,D6] [0,0,1,1,1,1]
6	SA: A2 | DA: B1 | SA: S | DA: D3 | [D1,D2,D3,D4,D5,D6] [0,0,1,1,1,1]
7	SA: S | DA: D3 | [D1,D2,D3,D4,D5,D6] [0,0,1,1,1,1]
8	SA: S | DA: D4 | [D1,D2,D3,D4,D5,D6] [0,0,0,1,1,1]
9	SA: S | DA: D5 | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,1]
10	SA: B2 | DA: C1 | SA: S | DA: D5 | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,1]
11	SA: S | DA: D5 | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,1]
12	SA: S | DA: D6 | [D1,D2,D3,D4,D5,D6] [0,0,0,0,0,1]

- The red lines indicate map-reply message sent back to ITR in the source domain.
- The green and red lines are for one-time transactions only used to establish the ETR addresses in cases of cache-miss at the ITR of the source domain.

6.5.2 With Modification on LISP infrastructure

The cascaded delivery of XCAST data of the first approach is very detrimental in delivery of multimedia data hence that approach is not plausible. We therefore propose a second approach in which the both ingress and egress LISP Tunnel Routers (xTRs) are made to be XCAST aware. With this option, the XCAST processing at

the xTRs changes to be as follows:

1. Processing in ITR:

    - An ITR does a mapping (getting RLOCs) for all the EIDs in the routing extension header
    - An ITR replicates the packet for each of the unique RLOCs found during the lookup
    - An ITR updates the bitmap for each copy such that only destinations corresponding to the RLOC in each copy have their bits set to 1 in the bitmap
    - An ITR does encapsulates each copy of the packet with an appropriate LISP header
    - An ITR sends out each copy to the ETRs that have the corresponding RLOCs in each copy.

2. Processing in ETR:

    - An ETR decapsulates the received copies and sends out to packets to hosts within its domain for each corresponding EID

This means that the xTRs have must be able to distinguish between XCAST non-XCAST packets. This is can be done using the traffic class or flow label fields as currently used by the outer IPv6 header.

Modification on XCAST

Using this approach, we still have the two schools of thoughts explained earlier and either of them can be applied to XCAST implementation. Irrespective of the school of thought opted for, sending, reception and processing of XCAST packets will follow the procedure explained below. However it is important to note that XCAST processing at the end-hosts will remain as defined in the XCAST RFC document[3] and there is

no need for cascaded delivery. A possible side effect of this approach is a considerable overload of the xTRs. However this needs further investigation.

XCAST Sending Process in this scenario

i. A source end-host gets end-host EIDs for all XCAST receivers from a group server

ii. The source creates an IPv6 packet with its own EID as the source address.

iii. The source embeds all receivers' EIDs in the routing extension header.

iv. The source sets the destination address to "well known" ALL_XCAST_NODES address ("ffoe::114").

v. The source then sends the XCAST packet to an ITR (Again this is an improved XCAST header where the semi-permeable header has been removed).

vi. The ITR does map-replicate-and-encap as described above under xTRs

vii. The ITR sends the copies to respective ETRs

viii. The target ETR decapsulates the packet

ix. If the packet has more than one EIDs with corresponding bits set to 1, the ETR replicates the packet for each EID.

x. The ETR then zeros all the bits in the bitmap and sends out each copy to their final recipients.

6.5.3 With XCAST Group Server in the LISP Mapping System

The third approach is where we seek to take advantage of LISP's handling of multicast group address(see section 6.3.3). Rather than getting the ETRs send back their RLOCs to the sender's ITR, we have an XCAST group server installed within the

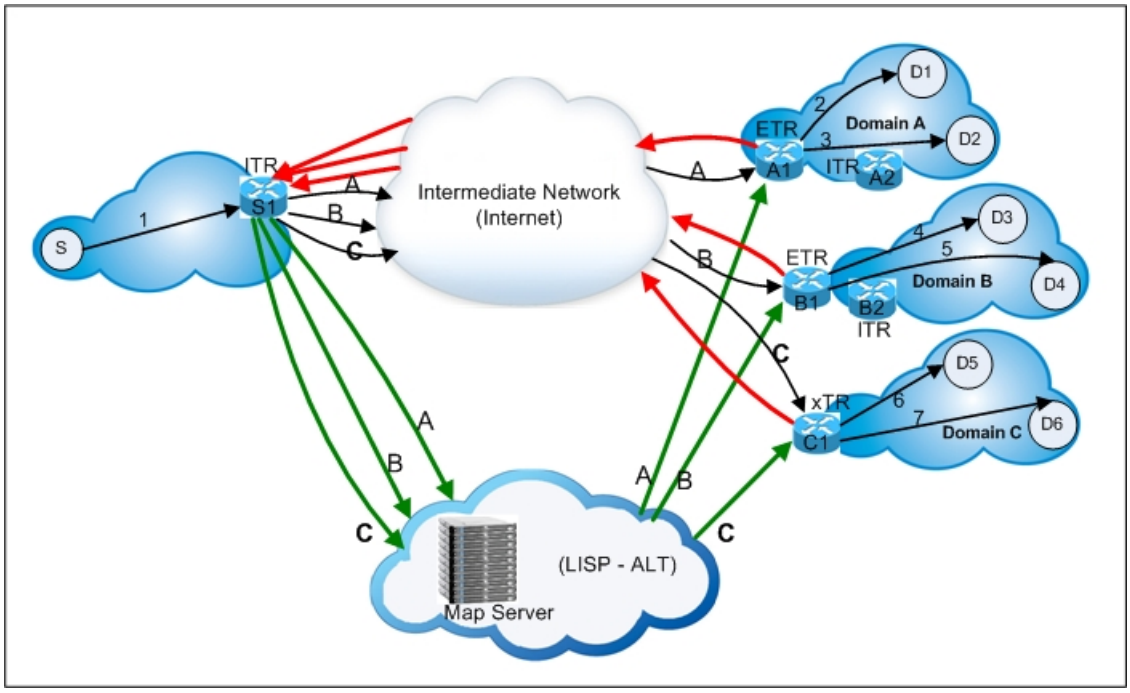


Figure 6.3: Sample LISP network with XCAST aware xTRs

LISP mapping system, from which the map server gets information of members in a group. For example, we can have members being associated with an EID like "XCST_HOSTS" and xTRs register with the group server to be associated with this EID. The *"XCST_HOSTS"* is to be treated like a multicast group address therefore it is both an EID and an RLOC at the same time. Each xTR also knows other xTRs and hosts that participate in this group.

When a packet is sent, the sender simply puts the EID *"XCST_HOSTS"* as the destination address and sends to its domain ITR. the ITR then asks the map server for resolution. The map server consults the XCAST group server and sends back the RLOC of any of the ETRs in the requested domains that participated in this group (Alternatively it can send RLOC of all the participating ETRs). In this case we call the "XCST_HOSTS" a "well known" EID since the members know each other.

On getting an ETR which understands the "well known", EID, the sender's ITR just does a map-and-encap to the XCAST packet and sends the packet to the

Table 6.3: How an XCAST packet changes in figure 6.3

Flow step	Corresponding packet structure
1	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [1,1,1,1,1,1]
A (Black lines)	SA: S1 | DA: A1 | SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [1,1,0,0,0,0]
2	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [1,0,0,0,0,0]
3	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,1,0,0,0,0]
B (Black lines)	SA: S1 | DA: B1 | SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,0,1,1,0,0]
4	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,0,1,0,0,0]
5	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,0,0,1,0,0]
C (Black lines)	SA: S1 | DA: C1 | SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,1]
6	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,0]
7	SA: S | DA: Dx | [D1,D2,D3,D4,D5,D6] [0,0,0,0,0,1]

identified ETR. When the packet reaches such an ETR (e.g. A1 in figure 6.4), the ETR replicates the packet and updates the bitmap according to XCAST algorithm and sends the copies to the nodes in its own domain and also the other ETRs in the domains of other participating hosts.

A further modification can be done to the LISP-aware XCAST especially on IPv6 (XCAST6). This is because IPv6 packet has some fields that are currently not used. These fields can be used for "flagging" purposes. In such a case, the flag can be used to switch between the second and third (in sections 6.5.2 and 6.5.3) implementation approaches above in an xTR. For example, if the bit is set then option 2 (section 6.5.2) is used especially for delay-sensitive contents like a videoconferencing session. If it is not set, the option 3 (section 6.5.3) is used.

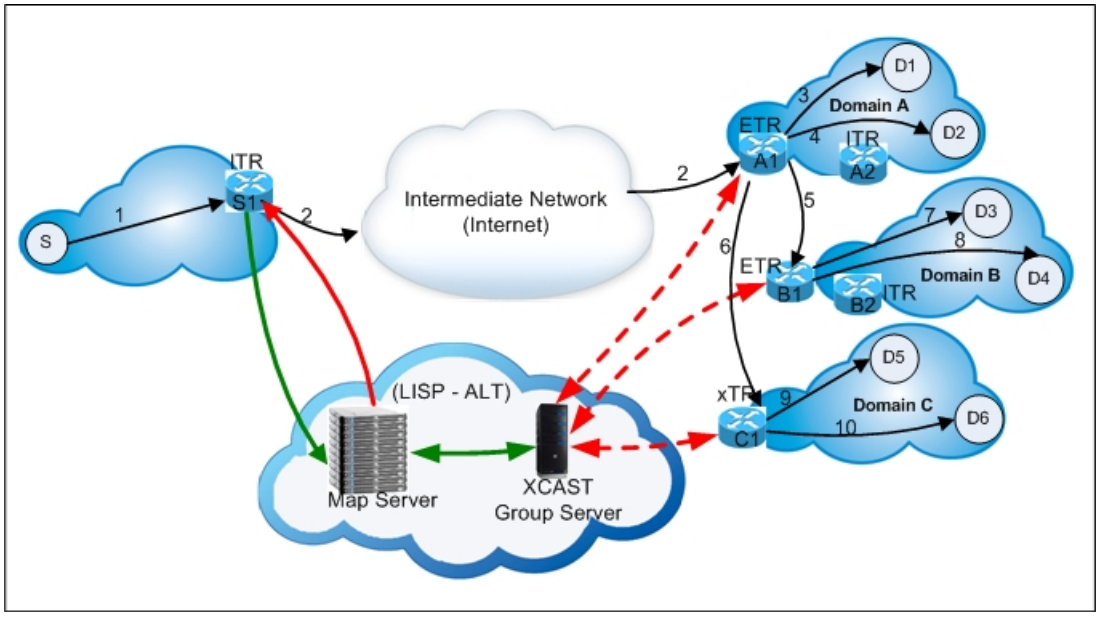


Figure 6.4: Sample network in which XCAST aware xTRs register with group server

6.6 Implementing LISP in OMNeT++

In order to evaluate our proposed integration approaches above, we require a platform that supports both XCAST and LISP. Since we had already implemented XCAST in the INET Framework of OMNeT++[119, 16], we had to find how to integrate LISP into the same environment since OMNeT++ currently does not have inbuilt support for LISP. We implemented LISP data plane, control plane and a test application for use in the evaluation. The LISP implementation on OMNeT++ is summarized as follows:

i. **Data Plane**

   a. IPv6-in-IPv6 encapsulation

ii. **Control Plane**

   a. Map-Register with a statically configured single Map-Server

   b. Mapping lookups using Map-Resolvers and statically assigned Map-Server

Table 6.4: How an XCAST packet changes in figure 6.4

Flow step	Corresponding packet structure
1	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [1,1,1,1,1,1]
2	SA: S1 | DA: A1 | SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [1,1,1,1,1,1]
3	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [1,0,0,0,0,0]
4	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,1,0,0,0,0]
5	SA: A1 | DA: B1 | SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,0,1,1,0,0]
6	SA: A1 | DA: C1 | SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,1]
7	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,0,1,0,0,0]
8	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,0,0,1,0,0]
9	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,0,0,0,1,0]
10	SA: S | DA: XCST_HOSTS | [D1,D2,D3,D4,D5,D6] [0,0,0,0,0,1]

c. Sending of Map-Request Messages

d. Processing of Map-Reply Messages

e. Sending of Map-Register Messages

f. Sending of Map-Notify Messages

g. RLOC probing at an interval of 30 seconds.

iii. **Test application**

a. Modified the UDPBasicApp that comes with INET Framework

To achieve this, we introduced numerous classes in the OMNeT++ source code. We also modified several existing classes among them the IPv6 class so as to allow for LISP routing capabilities in OMNeT++ network models. Below is a list of the new classes we introduced into OMNeT++ to implement LISP capabilities:

i. LISP (Core LISP characteristics)

ii. LISPSimpleMapTable

iii. LISPMapResolver

iv. LISPRouting6

v. IPv6 (Modified to suppor both LISP and Non-LISP routing)

vi. LISPControlMessages

vii. LISPEncapControlMessages

viii. LISPMappingRecord

ix. LISPMapServer

x. LISPMapResolverAccess

xi. LISPRouting6Access

xii. LISPMobileNode (WirelessHost6)

xiii. WirelessRouter6

In the subsequent sub-sections we describe how we knit these classes to achieve LISP functionality in the INET Framework.

6.6.1 The Network Layer Module

In OMNeT++ the network layer functions are mostly implemented in the *IPv4* and *IPv6 modules*. These modules form the key points where routing decisions are made in OMNeT++. We implemented LISP functionality by combining the new classes introduced in section 6.6 into new LISP related modules. The core LISP functionality is implemented in the *LISPRouting6* module. This module is then linked up with the *IPv6* module to achieve the LISP routing functionalities within LISP nodes.

While the *LISPRouting6* module offers several LISP-related functions, key in its implementation are the LISP packet *encapsulation* and *decapsulation*. These are implemented within two methods we call *lisp_hooks* and are named *lisp6_output()* and *lisp6_input()* respectively. The *lisp6_output()* method handles encapsulation of packets destined to other LISP domains and it also invokes methods that perform EID-to-RLOC mapping. The *lisp6_input()* method does the reverse by handling incoming packets from other LISP domains and performs the packet *decapsulation*.

The *lisp_hooks* are implemented in the *LISPRouting6* module and therefore nodes that implement LISP functionality must have *LISPRouting6* in them. *LISPRouting6* module is then invoked within the *routepacket()*, *routeXcast6Packet()* and *assembleAndDeliver()* methods of the *IPv6 module*. In section 4.4.1 we had explained how XCAST integration in the network layer is achieved and by having *lisp_hooks* now hooked into the *IPv6 module* via the *LISPRouting6* module, we have effectively achieved LISP-XCAST integration in OMNeT++.

6.6.2 LISP Nodes

We added the following new nodes into the INET framework that can be used in modeling LISP networks:

LISPxTR

LISPxTR is our implementation of the LISP Tunnel Routers as specified in the LISP draft document[9]. It is a new node we introduce into the INET Framework by extending the *Router6* module with LISP routing behaviour. Since the classes that implement LISP routing functionalities have been aggregated into the *LISPRouting6* module, the *LISPxTR* at its simplest is a *Router6* module with *LISPRouting6* added in its network layer. We also added the *LISPMapResolver* module to help in communication with the MapServer node. Communication between the the *IPv6* module

and the two LISP modules is achieved through INET's *NotificationBoard* module. The *LISPxTR* module implements both ITR and ETR behaviour. This simplifies the process of building either single-homed or multi-homed LISP domains with multiple LISP gateways.

LISP static Nodes

The static nodes are not affected and the usual OMNeT++'s NetworkHost6 is able to process the LISP. This is because the network layer modules of the host nodes do not embed the *LISPRouting6* module.

LISPMapServer

The *LISPMapServer* node acts as an interface to the mapping system and receivers map-register and map-request messages from the *MapResolver* module. usually, these messages are sent over UDP on well known port 4342. The *LISPMapServer* node contains the *MapServer* module that queries its database and performs EID-to-RLOC resolutions. The key responsibility of this module is to receive and process the LISP control messages and return reply messages with the requested information, usually to the *LISPxTR* and *LISP-MN* modules.

Currently we have not fully implemented the *LISP Alternative Topology (LISP+ALT)* but we look forward to implementing it in the INET Framework.

6.7 Integration on LISP-MN

LISP-Mobile Node *(LISP-MN)* module implementation in OMNeT++ is by modification of the *WirelessHost* module. Currently the *WirelessHost* module only supports *IPv4* protocol stack. Additionally, the existing wireless communication modules in the INET Framework support only one wireless interface per client. We therefore

created two new nodes, *WirelessHost6* which supports IPv6 and *LISPMN* which is a *WirelessHost6* that supports LISP protocol. To allow for testing of the multi-homing capabilities of the LISP Mobile Node architecture, the LISP-MN is required to support multiple wireless interfaces.

6.7.1 Wireless Host6

In the existing INET Framework, the *ChannelControl* module only keeps a single interface so it also had to be modified to keep a list of all wireless nodes. The *ChannelControl* module then keeps a list of all wireless entities per interface and not per node like in the original model. This way, a wireless host can be registered with multiple wireless interfaces at the *ChannelControl* module. This required changes to the *AbstractRadio* module, the *ChannelControl* module, the *ChannelAccess* module and the *BasicMobility* module.

6.7.2 LISP-MN

To include the LISP mobile node architecture, we extended the newly created *WirelessHost6* module by including an enhanced version of *LISPRouting6* module. We added extra conditional codes in the IPv6 module to handle both packet sending and delivery correctly. Since the RLOC address of the mobile node can change at handover, the *MapResolver* module has to be made aware of such changes. In our implementation, these changes are signaled through the *NotificationBoard*.

Once the wireless interface gets a new careof-address, the registration process is started by the *MapResolver* module. This event is signaled between the *InterfaceTable6* module and the *MapResolver* module through the *NotificationBoard*. During the registration process, the MapResolver sends a *map-register* message to the configured *MapServer* node. The MapServer acknowledges the map-register message with a *map-notify* message. Once the MapResolver receives this message, it starts

the configured update process of remote caches of communication partners and this completes the handover process.

6.7.3 WirelessRouter6

We also had to implement a *WirelessRouter6* module that supports IPv6 protocol for the INET Framework. This is used in communication with WirelessHost6 and the LISP-MN and allows for the inter networking between the wired and the wireless domains in the simulation model.

6.8 Evaluation

With all these nodes in place, we were now able to evaluate our integration approaches as proposed earlier. This is discussed in details in this section.

6.8.1 Comparative average latency

A comparative study of the average per-hop latency was conducted between the approach where no modification is done to LISP and that in which the LISP xTRs are made XCAST aware (approach in section 6.5.1 vs that of section 6.5.2). As shown in figure 6.5, a higher average per-hop delay is observed in the approach in which the LISP xTRs are non XCAST-aware. This is due to the cascaded delivery of XCAST data in a "hop-by-hop" fashion from one host to the next. If LISP xTRs are made XCAST-aware, a better performance is observed.

6.9 Conclusion

We extended the OMNeT++ by also implementing LISP into it. We then proposed three different approaches through which LISP-XCAST integration can be achieved.

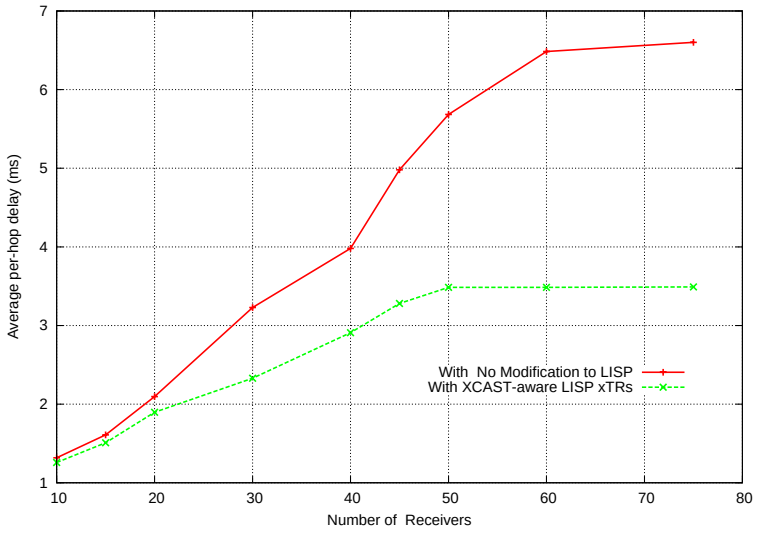


Figure 6.5: Sample network in which XCAST aware xTRs register with group server

We compared two of these approaches and concluded that one of them which involves introducing XCAST-aware tunnel routers is more acceptable. LISP-XCAST integration is important because it will ensure that XCAST can not only be deployed in the current Internet, but also in the Future Internet in which the routing locators and host identifiers are in separate numbering spaces. LISP also simplifies mobility, there LISP-XCAST integration also ensures that XCAST mobility can be easily realized.

Chapter 7

Conclusions and Future work

7.1 Conclusions

The work presented in this dissertation aims at realizing multipoint communication over the Internet using XCAST. It is divided into two parts. The first part deals with deployment of XCAST in the real world. The second part deals with testing of XCAST in a simulation environment.

In the second part, OMNeT++ network simulation tool has been used to implement the simulation environment for XCAST, Differentiated Services Architecture (DiffServ) and LISP. Using OMNeT++ we have investigated how QoS can be guaranteed for sensitive applications running in XCAST network and how XCAST can be integrated with LISP.

Realizing multipoint communication using XCAST was founded on a set of minor objectives including simplicity, cost-effectiveness, efficiency, scalability and incremental deployment. These minor objectives formed the foundation of the key objectives of this research. Hence in figure 7.1, we refer to them as the "building blocks of the research objectives". Above them in the hierarchy shown in figure 7.1 are the mechanisms employed by this study in ensuring that the objectives are achieved. These form

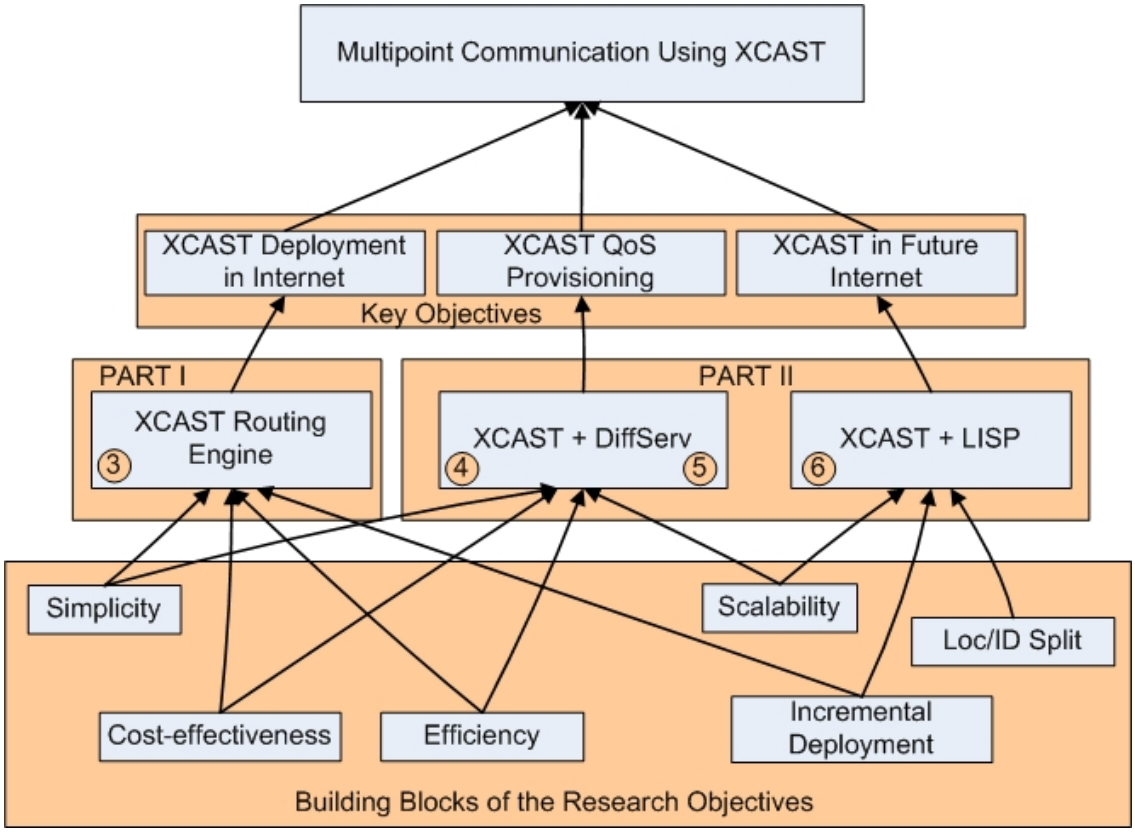


Figure 7.1: Putting it up altogether

the pieces of work we have presented in chapters three to six of this dissertation. In figure 7.1, the circled numbers show which chapter of this dissertation implements the corresponding objective. These pieces of work converge to help achieve the three key objectives which are important in realizing multipoint communication using XCAST. This is depicted in figure 7.1 by the flow of arrows which shows convergence at the top of the *"pyramid"*. The study achieves:

i. Simple, cost-effective, efficient, incremental deployment of XCAST.

ii. Simple, scalable XCAST QoS provisioning.

iii. Scalable, LISP-aware XCAST for the Future Internet.

The subsequent sections look at how the three objectives contribute to the overall theme of this research.

7.1.1 XCAST6 Deployment in the Internet

XCAST has several advantages but a key limitation impeding its usage on the Internet is the fact that currently there are no routers which understand it. This leaves the possibility of XCAST usage to an inefficient approach where XCAST packets are processed at the XCAST-aware end hosts. In setting out to realizing multipoint communication over the Internet using XCAST, one part this research was investigating how deployment of XCAST protocol in the Internet can be achieved in a manner that ensures efficient processing of XCAST packets.

This study explored various options for achieving XCAST deployment in the Internet. The study proposed an external component that can be used to *"assist"* routers so that they can efficiently process XCAST packets. The study investigated three methods by which this external component can be implemented. This included using programmable network processors, vendor supplied SDKs and a "software-based routing". After analyzing the strengths and weaknesses of each method, the study settled on the "software-based routing" approach in which the *"software-router"* is implemented in FreeBSD operating System. This external component that processes XCAST packets is what is called *XCAST6 Routing Engine* in this dissertation.

Using XCAST6 Routing Engine has the advantage of being simple to implement, cheap (since FreeBSD is open source) and above all provide efficient processing of XCAST packets. The XCAST6 Routing Engine therefore eliminates the cascaded delivery of XCAST data that leads to inefficient processing of XCAST packets in non XCAST-aware routers.

Other than realizing a simple, cost-effecctive and efficient XCAST deployment, another significant contribution of the study is that performance evaluation of the XCAST6 Routing Engine reveals that XCAST processing does not subject network routers to any unusual load. The XCAST6 Routing was evaluated against a number of metrics including throughput, latency, latency distribution, CPU utilization, Memory

utilization and the CPU context switch behaviour. All these metrics have revealed that XCAST processing is within the normal range as other network protocols. This is very important because these results can be used to confirm to the commercial router vendors that they can integrate XCAST into commercial routers without any fear of performance degradation on their routers. Such a step is very important because it will help realize a universal usage of XCAST in the Internet.

7.1.2 QoS-Aware XCAST

XCAST protocol is used for transport of multimedia data in applications such as videoconferencing programs and online games. These application programs are often used in realtime communications. Quality of data used in these applications must therefore be very good so as not to impact negatively on the user experience. Taking the need for good quality into consideration, realing communication using XCAST should therefore have mechanisms to guarantee Quality of Service to applications running in an XCAST network. This study therefore proposed how to make a QoS-aware XCAST that can be used for both realtime and non-realtime packet delivery.

The study looked at various possible options for QoS provisioning in XCAST and proposed the integration of XCAST with the Differentiated Services Architecture in order to achieve a QoS-aware XCAST. In order to implement the proposal, the study first chose a network simulator, OMNeT++ and added XCAST into its protocol stack. Chapter four of this dissertation gives a narrative of how XCAST implementation in OMNeT++ can be realized. XCAST implementation in OMNeT++ is then evaluated with respect to a number of metrics in order to verify its correctness.

The study then proceeds to implement the Differentiated Services (DiffServ) Architecture in OMNeT++. There are a number of challenges when trying to integrate DiffServ into a multipoint communication environment. In chapter five of this dissertation we looked into these challenges in details and how to overcome them in

order to realize a QoS-aware XCAST. Evaluation of the QoS-aware XCAST shows that it provides efficient bandwidth utilization, better packet forwarding fairness between XCAST and non-XCAST traffic, lower traffic load on routers and elimination of collusion attack (*"Good Neighbour Effect"*).

This way, the main objective of this study has been realized since the resulting QoS-aware XCAST can be relied on for use even with applications where QoS guarantee is required.

7.1.3 XCAST on LISP

The third objective of this study was to achieve an XCAST implementation that is compatible with LISP. This is because, research on the Future Internet has recently picked up and LISP is one of the protocols that are being viewed to be used in the Future Internet. Therefore a LISP compatible XCAST will not only simplify XCAST mobility (using LISP-MN) but also ensure that XCAST deployment in the Future Internet is devoid of numerous challenges.

This study implemented LISP in OMNeT++ and integrated it with XCAST. The study initially explored three possible methods for integrating XCAST with LISP. After analyzing the probable strengths and weaknesses of each option, the study implemented two of the option. The first option is the case in which there is no modification done to the LISP infrastructure while the second option requires that the LISP tunnel routers (xTRs) be XCAST-aware. The study compares these two approaches and finds out that XCAST-aware xTRs simplifies XCAST-LISP integration and registers better performance in form of lower delays. XCAST-LISP integration is very important especially for mobility where XCAST can leverage on the LISP-MN architecture to achieve efficient handover and dual homing capabilities.

7.1.4 Summary of the appendices

There are six appendices in this dissertation. In the appendices, we have presented a comprehensive survey of protocols and architectures for integrating Differentiated Services QoS provisioning into multipoint communication environments. We have classified these approaches into five categories. Appendix A gives an explanation of our classification methodology. Each of the remaining appendices describe a specific approach to DiffServ-Multicast QoS provisioning. For every class, we have described two protocols. We have also looked at the key approaches to packet replication for each protocol or architecture then analyzed the strengths and weaknesses of each technology with respect to the specification of the DiffServ architecture. We included QS-XCAST6 in the comparison model and highlighted how it solves the per-host level QoS heterogeneity which in turn makes it avoid the Good Neighbour Effect while maintaining a stateless core. The trade-off however is in the increased packet header size due to embedded DSCPs. Finally, we observe that all the other proposed approaches have their own unique strengths and weaknesses which must be considered in the deployment phase but non of them can be applied to XCAST other than our proposed QS-XCAST6.

7.2 Future Research Directions

As future work, we shall be seeking to implement the XCAST6 applications relying on the routing engine, that can be used in the QoS research in multipoint communication and also to investigate among other things, security considerations for XCAST6.

Currently we have only implemented 14 per-hop-behaviours in our DiffServ architecture. We shall be seeking to implement as many PHBs as possible so as to increase the number of testing scenarios that can be done using this tool. We shall also be adding several statistics collection classes. This will help in ensuring that the

simulation models can be used to report on a huge number of metrics than currently captured. Such an environment would simplify exhaustive testing of QoS in XCAST. We shall then extend the simulation to investigate other possible security loopholes that can be attributed to QoS provisioning in XCAST.

As far as XCAST-LISP integration is concerned, we shall be looking at a detailed implementation of other aspects of the LISP infrastructure such as LISP-Alternate Architecture (LISP-ALT). We shall also be adding NAT traversal mechanisms and Dynamic Host Configuration Protocol (DHCP) to this architecture. These two functionalities are key to implementing a realistic LISP-MN architecture. We shall also be investigation various aspects of XCAST when combined with LISP-MN to achieve XCAST mobility.

Appendices

Appendix A

IP Mulitcast-DiffServ Integration

As observed in chapter 2, multicast deployment in the Internet has been impeded by among other factors, the scalability and QoS issues. To improve multicast state scalability, several mechanisms have been proposed[146, 147, 148, 149, 150]. However these approaches are not directly concerned with multicast QoS. The other attempts that have been made with regards to multicast QoS include[151, 152, 153, 154, 155, 156, 157, 158].

The approaches intended to solve QoS problems however exhibit various limitations. For example, some of them cannot scale well with large groups while others cannot handle heterogeneous QoS effectively. Failure to handle QoS heterogeneity results from their lack of tightly enforcing *"QoS precedence"*; the requirement that in an environment where members of a group have varying QoS needs, each link in the multicast tree must guarantee the data to be serviced at a QoS level not lower than the highest level issued by any of its downstream receivers.

It is important to note however that the mentioned approaches do not specifically deal with QoS provisioning on multicast using DiffServ. In this section we narrow down to the approaches that use DiffServ QoS provisioning. While multipoint communication and Differentiated Services can be complementary technologies for providing

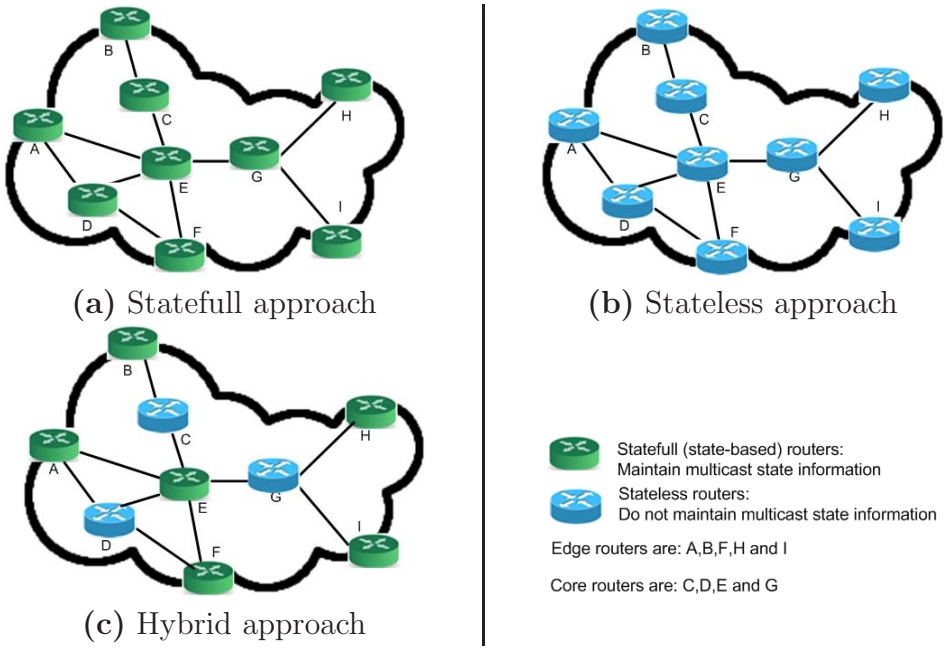
QoS in IP networks[159, 160, 161, 162, 163], their integration is a non-trivial undertaking owing to serious impediments posed by their architectural differences i.e. QoS in multipoint communication is receiver driven while DiffServ QoS is sender-driven.

QoS from a multipoint communication protocol is receiver driven owing to the fact that receivers can choose groups that meet their QoS requirements to join and through feedback based approaches, they can inform the sender of any changes in their QoS requirements. In DiffServ on the other hand, packet marking and flow policing is controlled by the ingress edge routers. Additionally, multipoint communication technologies like Internet Standard Multicast (ISM) lead to maintenance of multicast-state information in routers. DiffServ architecture on the other hand, specifies that the core network must be kept simple. Core routers must thefore not maintain state information. This complicates DiffServ-multicast integration.

In DiffServ architecture, the network routers are classified into two main groups of *core routers* and *edge routers*. The *edge routers* are further categorized as either *ingress edge-routers* or *egress edge-routers* depending on their locations relative to the source of the packets transmitted in the network. These routers form a domain where the ingress edge-routers' principal task is to mark the packets with specific codes called DiffServ Code Point (DSCP) and each DSCP is expected to allow the flow in the network to be shaped in a specific defined behaviour called Per-Hop-Behaviour (PHB)[7, 8, 64, 65, 66].

In these appendices, we provide a survey of ten protocols and architectures that have been proposed for using DiffServ architecture in multipoint communication networks. We classify the approaches into five categories and look at two examples from each category under each appendix. Finally, we summarize with analyses on the pros and cons of each approach for all the five categories.

Integrating multipoint communication with DiffServ is even complicated when other dynamics like admission Control[164], fault management, QoS Precedence and



(a) Statefull approach

(b) Stateless approach

(c) Hybrid approach

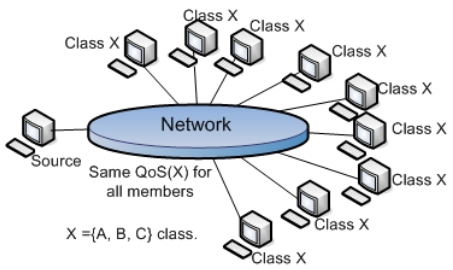
Figure A.1: Different approaches to handling of multicast-state-information.

management of QoS heterogeneity are to be factored in. This chapter looks at various approaches that have been proposed to provide QoS using an integration of the two technologies of multipoint communication and Differentiated Services. We classify them based on how they handle the *"state-information maintenance"* into *statefull (state-based), stateless and hybrid* approaches. Figure A.1 shows a graphical illustration of state-maintenance as used in this classification. We further classify the *"hybrid"* approach into three sub-classes thus ending up with five categories.

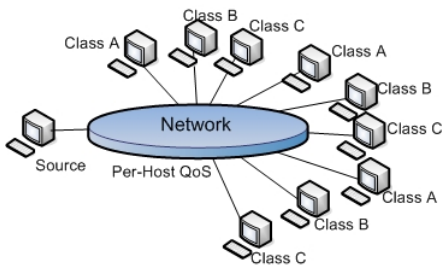
A.1 Integration Difficulties

DiffServ and Multicast should essentially be complementing each other in QoS provisioning whereby multicast plays a significant role in situations where DiffServ is to be used in multipoint communication environments. However, current multicast architecture does not handle QoS efficiently because:

1. Multicast routing state does not scale well. Routers need to keep states "per-



(a). Same QoS per group



(b). Heterogenous QoS in a group

Figure A.2: An overview of QoS provisioning strategies.

group" and in some multicast variants, "per-group, per-source" information must be maintained. A large number of groups results in a large number of state information to be stored by the routers. This translates to larger memory requirements and a slower packet forwarding rate. DiffServ specification on the other hand forbids state-maintenance in the core routers. Core statelessness is therefore hard to achieve when integrating DiffServ with multicast.

2. A multicast tree is associated with a single group. This complicates resource reservation while integrating multicast with DiffServ. This is because to meet this requirements, resources should be reserved separately for each multicast tree associated with a given sender so as to allow for simultaneous sending by multiple sources with QoS constrains. However, creating and maintaining a multicast tree per group is time and resource consuming. As the number of members increases, the multicast tree complexity might also increase. This brings in additional challenges in multicast tree management and scalability of control management in the multicast tree.

3. Various nodes in a multicast session might exhibit varying QoS requirements. One solution is to split the members into groups whereby each group is served at a specified QoS SLA. Figure A.2(a) shows such a scenario. Other form of QoS heterogeneity exists whereby members of the same group exhibit different QoS requirements. This results in *"in-group" or "host-level"* QoS heterogeneity as shown

in figure A.2(b). Supporting heterogeneous-QoS in situations where members of a group have varying QoS requirements is very complex.

4. The architectural differences in approach to QoS provisioning between multipoint communication technologies (e.g. multicast) also poses difficulties in integration. QoS provisioning in multicast is receiver-driven. The receivers join multicast groups that offer QoS levels which meet their requirements. Receivers can also use feedback messages to the sender to inform the sender of changes in their QoS requirements. In DiffServ on the other hand, QoS is sender-driven. This is because the packet shaping and policing is done in the ingress edge routers on the sender-side. The core network simply forwards the packets and the receivers have no control on the packet shaping and policing.

A.2 QoS Provisioning Approaches

Various approaches have been proposed to solve the above challenges and help realize provisioning of QoS services in multipoint communication[165, 166, 167, 168]. These approaches can be broadly classified into three main categories depending on how they handle multicast state information maintenance in routers. As summarized in table A.1, the approaches can therefore be classified as:

(I) Stateful,

(II) Stateless or

(III) Hybrid.

However, the hybrid class can also be further categorized into *selective, edge-based and aggregation-based* approaches. As summarized in table A.2, we end up with a detailed classification scheme of five classes, namely:

(a) State-based (Statefull) approaches. (Appendix: B)

Table A.1: Broad classification of QoS provisioning approaches

	Statefull	Stateless	Hybrid
Explanation	All routers maintain multicast state information.	Routers do not maintain multicast state information.	Only some routers maintain state information
QoS handling	Easy to implement "Per-group" QoS	Can implement "Per-group" and "per-tree" QoS	Easy to implement "Per-group" QoS
Examples	DAM, Quasimodo. See (B.1) and (B.2)	QS-XCAST, DSM-Cast. See (C.1) and (C.2)	DCM, QMD, EBM, AQoSM. See (D.1), (D.2), (E.1) and (F.1)

Table A.2: Detailed Classification of QoS Provisioning approaches

Category	State Information handling	Replication Level
State-based	Both edge and core routers maintain state information	per-group in routing tables
Stateless	No state maintenance	Replication information is embedded in the packer header
Selective	Only selected ***"key nodes"*** maintain state. Key nodes can be core or edge routers.	Uses tunneling
Edge-based	Nodes that maintain state are purposely deployed at the edge of the network	Uses tunneling
Aggregation-based	Map several distribution trees onto one distribution tree	Uses tree-aggregation

(b) Stateless approaches. (Appendix: C)

(c) Selective approaches. (Appendix: D)

(d) Edge-based approaches. (Appendix: E)

(e) Aggregation-based approaches. (Appendix: F)

In the subsequent appendices, we look into details the key ideas behind some of these approaches. We also highlight their fundamental strengths and weaknesses.

Table A.3: Summary of the DiffServ-Multipoint Communication integration approaches.

Category	Example	Advantages	Disadvantages
State-based	DAM, QUASI-MODO	i. Has the most efficient bandwidth usage	i. Do not conform to DiffServ requirement of statelessness at the core ii. Scalability problem due to state maintenance
Stateless	QS-XCAST, DSMCast	i. No state maintenance ii. Scale well with the number of groups	i. Not scalable for large group-size ii. Incur Processing overhead in routers

Selective	DCM,QMD, HBH, RE-UNITE	i. Meets DiffServ requirement: The core is completely multicast-unaware ii. Easy to implement iii. Highly scalable	i. Incur performance degradation compared to stateful approaches ii. Only works well for sparse groups where replication is less at the core.
Edge-based	EBM, MMT	i. Meets DiffServ requirement: The core is completely multicast-unaware and only edge routers maintain state information ii. Highly scalable	i. Incur performance degradation compared to stateful approaches ii. Only works well for sparse groups where replication is less at the core.

Aggregation	AQoSM, Harmonic DiffServ	i. Core routers only maintain minimal state information (***"for super multicast groups"***). ii. Partially conform to DiffServ requirements	i. Aggregation only reduces state maintenance at the core but does not eliminate it ii. Only partially meets DiffServ requirements iii. Most aggregators introduce new entities that can be single point of failure. Distributing these entities increases protocol complexity

Table A.3 gives a summary of the examples, advantages and disadvantages of each of the above mentioned approaches. In the following subsections, we look at the fundamental principle behind each of these classes and atleast two examples of protocols or architecture for each class.

Appendix B

State-based approaches

The state-based protocols and architectures derive the name from the fact that in this category, both the edge and core routers in the DiffServ domain maintain per-group state-information for each of the multicast groups and each multicast group has a corresponding multicast tree. A grave drawback of this approach is that state maintenance results in scalability problems due to substantial increase in router loads as more members join a multicast group. Statefulness at the core also contradicts the fundamental concept of DiffServ architecture which stipulates statelessness at the core and pushes complexities to the edge of the network. Some of the protocols and architectures that fall in this category include DAM (DiffServ Aware Multicast)[169] and QUASIMODO[170](which is an adaptation of the PIM-SM Model[145]). In the next subsections, we offer a more detailed description and analyses of these two examples.

B.1 DiffServ Aware Multicast(DAM)

DAM[169] is an approach aimed at tackling two major multicast-DiffServ integration problems, namely: Neglected Reservation Subtree(NRS)[49] and the inherent architectural conflicts between multicast and DiffServ. DAM is realized using three features namely:

i. Weighted Traffic Conditioning (WTC),

ii. Receiver Initiated Marking(RIM) scheme and

iii. Heterogeneous DSCP Headers encapsulation (HDHE).

WTC aims at maintaining the negotiated SLAs while RIM is primarily to accommodate QoS heterogeneity in the multicast groups and also to harmonize the *receiver-driven* versus the *sender-driven* QoS approaches between multicast and DiffServ. HDHE on the other hand is aimed at providing fairness between multicast and non-multicast traffic.

The basic idea of WTC is that at the edge routers of a DiffServ domain where traffic conditioning occurs, the admitted "multicast traffic" is counted as "multiple unicast traffics" as illustrated in figure B.1. WTC reserves bandwidth by overestimating the bandwidth consumption on the links based on the traffic flows that enter and the number of copied flows that leave the domain. The edge routers therefore maintain and update WTC lookup tables in which the multicast state information are stored. Each entry in WTC lookup table contains three entities namely: *multicast group ID*, *DSCP*, and *the number of replications*.

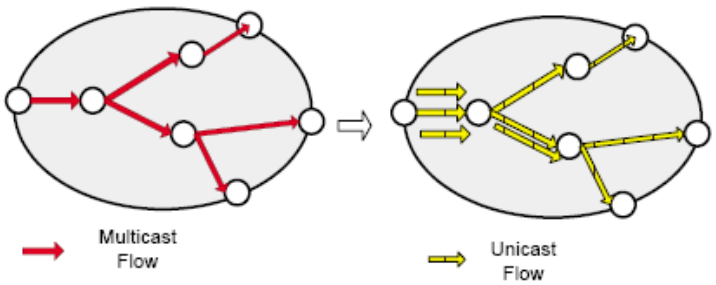


Figure B.1: Multicast Traffic in DAM. Source:[169]

In DAM, the RIM feature solves the *receiver-driven* vs *sender-driven* architectural conflicts exhibited in multicast and DiffServ architectures respectively. In RIM, this

is achieved by piggy backing a receiver's QoS requirements on the multicast JOIN message. DAM defines four levels which describe the possible types of QoS a receiver might request during a join operation. Using RIM, the network entities are signaled for admission control purposes to ensure that a receiver joins at a QoS level specified in the SLA and its branch is marked with correct DSCP that will ensure it gets the best available QoS meeting the SLA specification.

HDHE works by ensuring that when a multicast flow enters a DiffServ domain and has nodes with heterogeneous QoS requirements, the markings for each of the branches are encapsulated in the packet header at the ingress edge router of the domain. Figure B.2 illustrates HDHE operations. The argument here is that since the number of branches within a DiffServ domain are not expected to be so many, the HDHE will not pose significant overheads in processing due to the increased packet header length

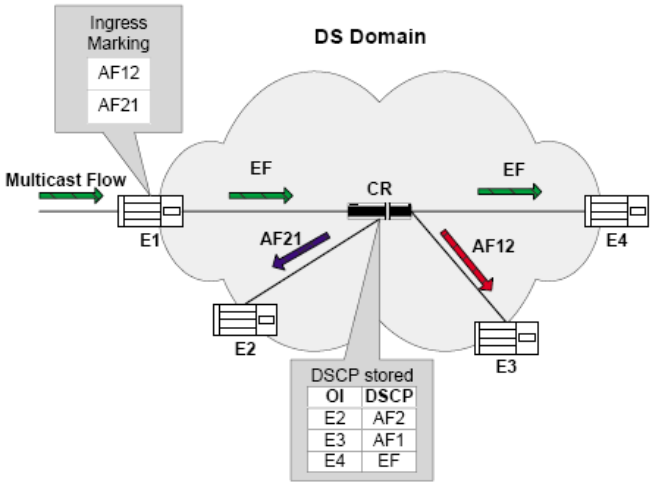


Figure B.2: DAM heterogenous QoS Example. Source:[169]

DiffServ Aware Multicast is generally a simple approach but a few drawbacks can be identified as follows:

i. Using WTC ensures adequate bandwidth is reserved even for the the replicated

traffic. However, in as much as SLAs may not be violated in this approach, the reserved bandwidth may never be wholly consumed leading to possible bandwidth wastage.

ii. The number of branches is hard to know in advance and in case of groups which result in massive branching within the domain, performance penalties will still occur.

iii. The WTC also requires that the edge routers maintain and update flow-specific information. While this state is maintained at the edge and therefore conforms with DiffServ requirements, the challenge comes in the necessary steps needed for load-balancing so as to reduce the WTC lookup table.

iv. Implementing DAM implies that the architecture of the edge routers (*which are the traffic conditioners in the DiffServ domain*) must be modified to accommodate weighted multicast metering since this is not implemented in many routers currently.

It is also important to note that in order to facilitate marking using RIM, each multicast routers need to have one of more DSCP field setup for the multicatst flow.

B.2 QUASIMODO

The acronym QUASIMODO is derived from ***Qua****lity of* ***S****erv****i****ce-aware* ***M****ulticasting* ***O****ver* ***D****iffServ and* ***O****verlay networks*. The approach in real sense is an adaptation of the *Protocol Independent Multicast Scarce Mode(PIM-SM) model*[145]. QUASIMODO[170] aims at solving among others the *Neglected Reservation Subtree(NRS) problem*[49] that arises while integrating IP multicast with DiffServ. NRS is a problem that arises due to the dynamic join or leave events in a multicast tree especially when a new member joins the QoS-constrained group without explicitly reserving required resources. This can adversely affect the existing traffic since the replicated packets

get the same DSCP of the original packet and thus experience the corresponding treatment thereby consuming un-reserved resources. Therefore, resources should be reserved separately for each Multicast tree associated with a given sender so as to allow sending by multiple sources with QoS constraints. QUASIMODO proposes to solve this problem by combining two ideas of:

i. Using a probe-based approach to ensure that resource availability along a new QoS path is verified.

ii. Providing for QoS precedence in cases of heterogeneous QoS by marking replicated packets with a special DSCP value before forwarding.

Because it is based on PIM-SM, it assumes that a Rendezvous Point(RP) router is defined in the network and the RP forms the root of all traffic sources. QUASIMODO also defines Designated Routers (DRs) which act on behalf of the host as far as PIM-SM is concerned by managing all group management informations (using IGMP[171]) on one side and on the other side emits PIM join/prune message towards RP. Additionally, the network also contains DiffServ routers which are non-multicast aware and are transparent to the multicast protocol hence forming a *"DiffServ Overlay"* in the network. Figure B.3 shows an example of a QUASIMODO network with the mentioned components.

QUASIMODO adds an additional field in the Multicast Routing Table(MRT) which specifies the DSCP(s) to be used for each each router *output interface(oif)*. The DSCP is then used to mark replicated packets that are forwarded along the considered *output interface(oif)*. QUASIMODO proposes that in order to support heterogeneous-QoS, traffic associated with different multicast trees should be treated independently. It also exercises admission control using a probe-based procedure called *GRIP (Gauge and Gate Reservation with Independent Probing)* which is based on the idea of *"implicit-signaling"*, that is, GRIP uses a data plane operation to

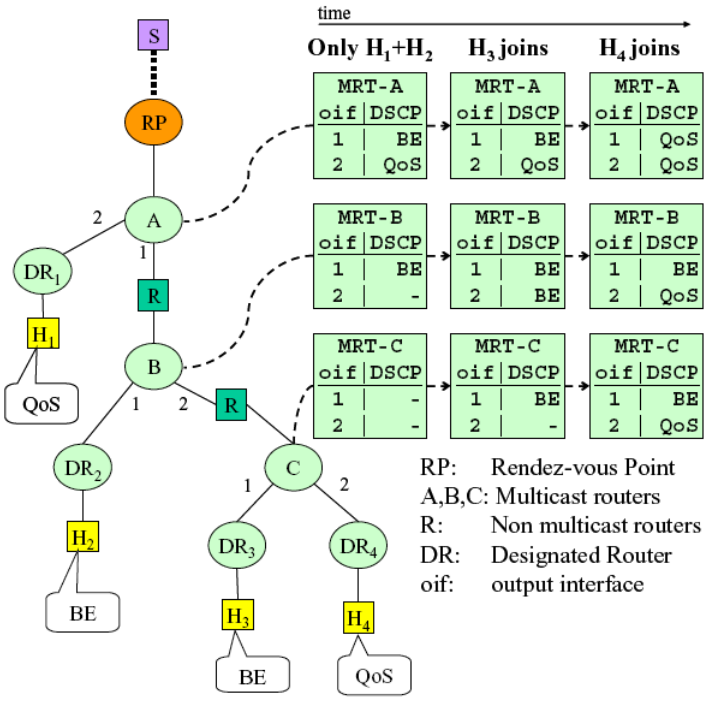


Figure B.3: QUASIMODO Architecture. Source:[170]

convey at the network borders, information on whether a network link is congested and cannot accept a new flow.

The drawback of this approach however is that besides relying on the PIM-SM, QUASIMODO works on many other assumptions about the MRT that are currently not supported in the commercial routers and the processing algorithm itself especially for handling member join events are very complex. The *"per-tree"* based QoS model also introduces serious scalability problems because routing decision made by each router is based on the fully-specified "multicast" address and the forwarding table size and volume of IGMP messages to be exchanged for maintaining the session grow linearly with the group size. In effect, contrary to the QUASIMODO proponents assertion, it has the potential of violating the scalable design principle of DiffServ architecture. The per-tree QoS also has the limitation that all clients on the same multicast tree must have the same QoS requirements and therefore the QoS hetero-

geneity is limited to the tree-level and not per-host.

Marking all packets leaving a given router interface with the same DSCP also has the subtle effect that in the event of full QoS Precedence, even clients who are not paying for QoS services will receive high priority treatment for services being paid for by their neighbours, a condition usually referred to as *Good Neighbour Effect (GNE)*. GNE has the potential of opening the network to collusion attack whereby malicious clients collude to get high quality services for services they are not actually paying for.

Appendix C

Stateless Protocols

In DiffServ context core routers only rely on marking of the packet to determine the unciast QoS. The stateless protocols for multipoint communication adopt a similar methodology whereby the necessary multicast information is embedded inside the IP packet. Therefore core routers do not need to maintain any state information about the multicast tree. The embedded information is either the multicast tree information or a list of destination addresses explicitly enumerated within the IP packet itself.

Stateless protocols have the basic advantage that they do not require the routers to maintain per-state information hence they do not unduly overload the routers and additionally comply with the DiffServ requirement of statelessness at the core of the network. Statelessness by these protocols increases their scalability especially in terms of the number of groups that they can support. However, they may not scale well in terms of the group size since encapsulation takes up some space in the IP packet header and increases the packet length. This scalability issue can be reduced by encapsulating only the tree for the DiffServ domain and not the entire end-to-end multicast tree. In effect, this makes the size of the tree to be dependent only on the egress points of the DiffServ domain and not the number of receivers.

The second weakness of these protocols is that they are likely to incur extra

penalty in form of packet header processing overheads. So far only two protocols are known to take this approach namely: XCAST[3, 6, 4, 5] (especially its QoS-aware variant, QS-XCAST[16] which we recently proposed) and DSMCast[172].

C.1 QS-XCAST (QoS Aware XCAST)

In XCAST[3] the sender explicitly specifies the destination addresses of all the receivers as a list of unicast addresses embedded in the IP packet header then sends the packet to a router. Along the transmission path, each router examines the IP packet header in order to determine the next-hop for each destination specified in the list. The router then groups together the destinations with the same next-hop and finally forwards a packet with an appropriate XCAST header to each of the identified next hops. The process is repeated until all the destinations are reached.

The XCAST packet header also comprises of a bitmap with bits corresponding to each destination, which the routers use to determine which of the embedded destinations the packet needs to be delivered and to which ones a copy of the packet has already been delivered. Therefore if a bit corresponding to a given destination is set to 1, it means the packet needs to be delivered to that destination. Each of the branching routers updates this bitmap for each copy of XCAST packet during replication. Figure C.1 is an illustration of a basic XCAST network.

To integrate XCAST with DiffServ, in addition to the list of destinations and a bitmap, the XCAST header also embeds a list of DSCPs in which a DSCP value corresponds to each destination. Further, adaptive re-writing of the *DS (Traffic class) field* of the IP header at the branching routers must be provided for. During the replication of XCAST packets at the branching routers, the bitmap field is used to determine which corresponding address in each copy of the packet the data needs to be delivered. The respective DSCPs of these destinations are evaluated to obtain

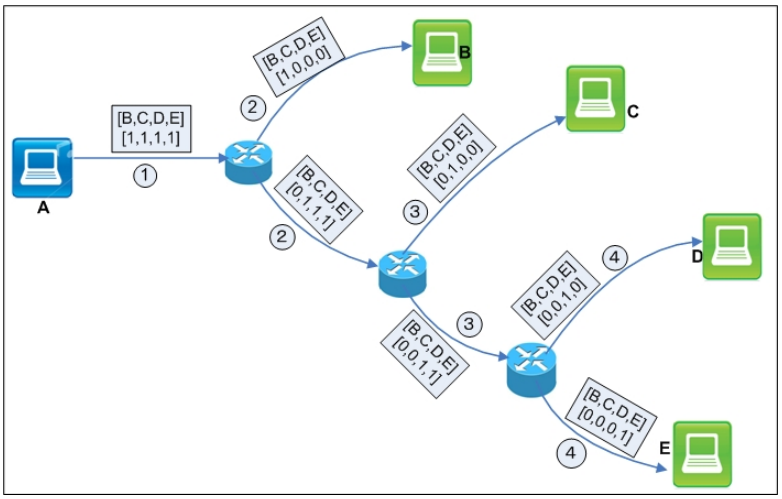


Figure C.1: XCAST6 overview

a new *QoS precedence* order. The DSCP fields of the packet copies where the new *most demanding DSCP* differs from the original DSCP value are then updated with the most demanding DSCP for each copy before the copies are transmitted to their corresponding next-hop routers. This QoS-aware version of XCAST is known as *QS-XCAST*[16, 119] and integrates well with DiffServ for QoS provisioning.

The advantage of the QS-XCAST approach is that it offers *"per-receiver"* level QoS hence supports *"host-level"* QoS heterogeneity as opposed to the per-group or per-tree based approaches to QoS heterogeneity that most of the other solutions here can offer. Because of its *"per-receiver"* level QoS, QS-XCAST also eliminates *DSCP confusion* problem in the routers where it ensures that each copy of the packet is enqueued in the appropriate router queue that can guarantee the packets prescribed DSCP-PHB. The greatest advantage of the *"per-receiver"* level QoS heterogeneity as offered by *QS-XCAST* is the total elimination of the network's susceptibility to *"collusion attack problems"*. In this kind of problem (*collusion attack*), several downstream receivers can take advantage of the network topology and collude to pay substantially minimal cost for the best QoS which is indeed paid for by one or more upstream receivers with neither the service provider's nor the the upstream receiver's knowledge.

Such upstream receivers are usually referred to as *"Good Neighbours"* hence *collusion attack* can also be at times referred to as *"Good Neighbour Effect (GNE)"*.

The QS-XCAST approach also spans multiple DiffServ domains. This gives it the benefit of being able to be deployed in a global Internet as long as the multiple DiffServ domains have Bandwidth Brokers that are able to offer inter-domain policy enforcement functionalities.

C.2 DSMCast

Unlike XCAST, DSMCast[172] does not embed the list of all destinations in the header. Instead, DSMcast reduces the problem to edge-to-edge transport across a single DiffServ domain by only embedding the multicast tree information of each group in a special header called *Tree Encapsulation Header (TEH)*. Encapsulation is done at the ingress edge routers and decapsulation follows at the egress edge router of the domain.

DSMCast is divided into four components, namely:

i. The transport mechanism,

ii. The extensions for heterogeneous QoS

iii. The tree construction mechanism

iv. The egress(member) join/leave protocol

In the subsequent subsections, we look at each of these components in details.

i. ***The transport mechanism***: All replication (or routing) for multicast packets is controlled by the *Tree Encapsulation Header (TEH)*. When a multicast packet arrives at an ingress router of the domain, the ingress router adds a *TEH* to the multicast. The *TEH* is inserted between Layer 3 and layer 4 (IP and UDP) header of the packet. When the packet is received by the core routers, the router

simply checks which DSCP unicast queue to place the packet in and sends it over. The packet is then treated according to the PHB defined for the DSCP class. The DSCP class can be re-written to accommodate QoS heterogeneity among members of a multicast group.

ii. ***Tree Encapsulation Header (TEH)***: The TEH is a sequence of bits that are responsible for routing and replicating the packets and comprises of 3 fields: *NumEntries*, *Options* and *Routing* entries.

iii. ***Variable QoS Extensions***: DSMCast uses this to support heterogeneous QoS among members in the same multicast group. This is realized by allowing the *DS (Traffic class)* field of an IP header to change as the multicast packet is replicated. The best PHB is propagated up the tree towards the ingress router.

iv. ***Tree construction***: The TEH is constructed and maintained by the ingress router. DSMCast assumes that the edge routers have an accurate and complete knowledge of the entire DiffServ domain possibly getting this information from the underlying routing protocols in the network, e.g. OSPF. On member join or leave, the ingress router updates its tree information accordingly. Additionally, DSMCast does not specify any specific tree construction algorithm and therefore assumes that there are mechanisms to deal with fault detection and reconfiguration in the network.

v. ***The egress(member) join/leave protocol***: DSMCast handles a join request from an egress router using two special routines called bid/probe routines. An egress router that wants to join the network sends a *Bid-Request* message to all edge routers. Each edge router that receives the *Bid-Request* message responds with a *Bid-Probe* message towards the new egress router. This helps the edge route to know which at point to join the group.

In addition to the general drawbacks of encapsulation-based protocols, the other

weaknesses of DSMCast are:

1. It is intended for a single DiffServ domain only since the TEH is removed at the egress end of the DiffServ domain. It is therefore not suitable for a global scale deployment like the real-world Internet.
2. DSMCast assumes that the edge routers know the network topology, bandwidth and PHB of each link in the network hence it is not clear on how DSMCast fits in a DiffServ architecture where the *Bandwidth Broker(BB)*[130, 173] is the key manager of a DiffServ domain since in such a domain, the bandwidth Broker is the Policy Decision Maker and the edge routers are simply Policy Enforcemnet Points (PEPs).

Appendix D

Selective Approaches

Under selective approaches, we classify protocols and architectures in which only routers at the branch nodes of the multicast trees are designated to maintain multicast forwarding states and perform packet replications. These protocols therefore implement QoS selectively hence the classification, *"Selective Approaches"*. In this approach, multicast packets are encapsulated and transmitted using unicast between the selected branch nodes. The effect is that non-branching nodes are relieved from maintaining state information. A number of protocols have been proposed that follow this approach including: DCM (Distributed Core Multicast)[174], QMD (QoS Aware Multicast for DiffServ)[175], REUNITE[148] and HBH(Hop-by-Hop)[176]. The branching points are usually selected on the least-cost-routes basis. However, QMD expands further on the selective nature to exploit other facets other than the least-cost based routes. In this class of protocols, we shall look at DCM and QMD.

D.1 Distributed Core Multicast (DCM)

DCM[174] is a sparse mode routing protocol that like XCAST scales well with the number of small sized-groups. However unlike XCAST, DCM does not employ header encapsulation. Instead, it follows other sparse mode multicast routing protocols such

as PIM-SM[145] and Core-Based Trees (CBT)[50] that build a single delivery tree per multicast group which is shared by all senders in the group. This tree is usually rooted on a central router referred to as the **"core"** in CBT and **"rendezvous point(RP)"** in PIM-SM. DCM is based on an extension of the CBT approach. It uses several core routers, called *Distributed Core Routers (DCRs)* and a special control protocol called *Membership Distribution Protocol (MDP)* for communication between the DCRs. The MDP runs between the DCRs serving the same range of multicast addresses and enables the DCRs to learn about other DCRs that have group members. Figure D.1 shows an example of data distribution using DCM architecture in a large single domain network[1].

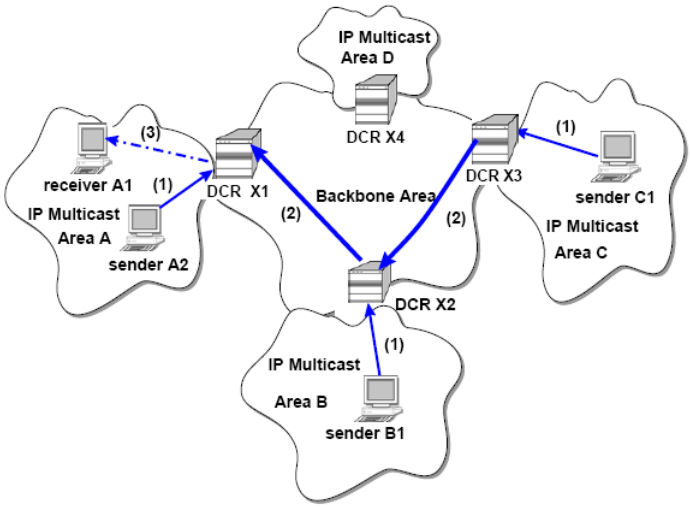


Figure D.1: DCM Architecture. Source:[174]

DCM considers a network model organized hierarchically into two levels. The top level is a single backbone area to which all other areas connect similar to the OSPF[177] network model. While DCM uses the *"area"* concept as in OSPF, it does

[1]The model shows four non-backbone areas that communicate via the backbone. It also shows a single multicast group M and DCRs X1, X2, X3 and X4 that serve M. In Step (1), the senders A2, B1 and C1 send data to the corresponding DCRs inside their areas. In step (2), the DCRs distribute the multicast data across the backbone area to DCR X1 that needs it. In step (3), a local DCR sends data to the local receivers in its area.

not require the underlying unicast link state routing and does not restrict the underlying unicast routing protocol to be OSPF. The DCRs are then placed at the edge of the backbone area but inside each non-backbone area there can still exist several DCRs serving as core routers for the area. Within non-backbone areas, multicast routers keep group membership information for groups that have members inside the corresponding area. The multicast state information kept by these routers is *"per-group"* and not *"per-source"*. Within the backbone, non-DCR routers do not keep the membership information for groups that have members in the non-backbone areas. However if deemed necessary, the backbone routers may keep group membership information for a small number of reserved multicast groups that are used for control purposes inside the backbone.

In DCM since the DCRs are close to any sender or receiver, converging traffic is not sent to a single central router in the network. If a host wants to send multicast data, it sends the the data to its local DCR and data sent from a sender to a group within the same area is not forwarded to the backbone. The multicast data delivery process is such that the DCRs act as backbone access points for the data sent by senders inside their area to receivers outside their area. A DCR also forwards the multicast data received from the backbone to receivers its area. Admission control in DCM is exercised in such away that if a host wants to join the multicast a group, the host sends a join message which is propagated by hop-by-hop to the DCR inside its area that serves the multicast group.

The only notable drawback of DCM is that its QoS heterogeneity handling can only go as low as "per-group" QoS and not "per-host". Additionally, being in as much as it is selective, the core is not completely stateless as required by DiffServ specifications.

D.2 QoS-Aware Multicast for DiffServ(QMD)

QMD[175] is a QoS-aware multicast routing protocol that aims at removing the complex multicasting control plane functionalities from the core routers. In QMD, for each multicast tree, only a small set of on-tree routers *(referred to as key nodes)* maintain the multicast routing states using *Multicast Forwarding Tables (MFT)* and therefore are the only ones that forward multicast traffic. Each MFT entry is composed of three fields: *the group id*, *key nodes of the group's children* and *the DSCPs for the traffic to children key nodes*. Multicast traffic is carried in the **"DATA"** message which includes the *group address* and the *real multicasting data*. A DATA message is a unicast packet with the destination address as the next key node. The DSCP of the DATA message is set to be that of its corresponding multicasting code point as read from the MFT. When a key node receives the DATA message, it first retrieves the group address and then obtains the corresponding MFT entry. If the entry has only one child key node, the current key node sends the DATA message to the child otherwise it duplicates the message and sends copies to all the children.

The key nodes of a multicast group uniquely identify a QoS-satisfied multicast tree connecting the group members. The other on-tree routers between any two key nodes do not maintain any multicast routing states. QMD is premised on the requirement that a Bandwidth Broker (BB) is installed in the DiffServ domain. The Bandwidth Broker (BB) does admission control and is also a decision point of other policies in the domain while edge routers remain to be policy enforcers in the domain. The core routers therefore maintain only minimal data forwarding states. QMD proposes the addition of multicast modules to the BB such that the BB's multicast module will handle most of the multicast control plane functions within the BB's domain such as processing the join/leave requests for admission control, finding new branches and informing the selected on-tree nodes to install data forwarding states.

In QMD, a QoS satisfied branch connecting any member to a multicast tree

is identified by *a list of key nodes*. To find a QoS-satisfied branch, QMD uses a slightly modified version of the Dijkstra algorithm called *"QMD-DIJKSTRA"*. QMD-DIJKSTRA models a DiffServ domain as a connected directed graph in which the set of nodes and their connecting links are used together with the link bandwidths to construct paths that meet the QoS constraints. In QMD, a set of *key nodes* can not only identify a multicast routing tree but also provide a predictable QoS services to the group members. The set of key nodes are divided into two namely: *branching nodes* and *milestone nodes*. The *branching nodes* are those ones that have more than one child in a multicast routing tree while the milestone nodes are used to enhance QoS support and provide desirable QoS to the multicast receivers by avoiding situations where some packets might need to be dropped due to link bandwidth exhaustion.

To realize the notion of *key nodes* and *non-key nodes* among the on-tree routers, the data plane implementation of QMD uses the idea of *"recursive unicast"* just as described in other selective multicast approaches such as REUNITE[148] and HBH(Hop-by-Hop)[176]. In *"a regular recursive unicasts"*, only the routers at the branching nodes maintain multicast forwarding state and the multicast traffic is repeatedly unicasted between the branching nodes without bothering other on-tree nodes. However unlike the QMD's *recursive unicast* approach, in *regular recursive unicast*, all other on-tree nodes still need to maintain the multicast tree control information and process join/leave events.

The drawback of QMD approach is that it does not consider how to harmonize the *receiver-driven QoS* versus the *sender-driven QoS* approaches of multicast and DiffServ architectures respectively which inherently undermine each other. Also, even though QMD moves complexity to the edge routers, it still demands state maintenance in some core nodes (*the key nodes*) so the core is not completely stateless.

Appendix E

Edge-based Approaches

Protocols and architectures in this class are almost similar to the *selective-approach* based protocols. The key difference is the fact that for this category, the branch nodes are purposely deployed at the edge (ingress) routers of the network. The net effect is that the core is completely multicast un-aware. It therefore conforms to the DiffServ architecture's requirement of statelessness at the core. The key advantage of this approach is that its protocols are easy to implement and are highly scalable. However, they incur performance degradation in terms of bandwidth consumption compared to the state-based and aggregation-based approaches. This approach is therefore highly suitable for sparse groups where replication of packets at the core is less likely to occur compared to that of dense groups which might need replication of packets at the core hence are likely to impact badly on this approach. The two protocols that fit in this family are the EBM(Edge Based Multicast)[178] and MMT(MPLS Multicast Tree)[179].

E.1 Edge Based Multicast(EBM)

EBM[178] works a cross a single DiffServ domain and not on a global scope like the Internet. However, it exploits the intelligence of edge routers and maintain core

statelessness. It achieves this by defining a new entity called a Multicast Broker (MB) for group management and ensures that the core routers are kept stateless and multicast-unaware. Since the core routers are multicast unaware, they do not maintain any multicast state information which is a great advantage of this protocol. Figure E.1 shows a sample architecture for EBM.

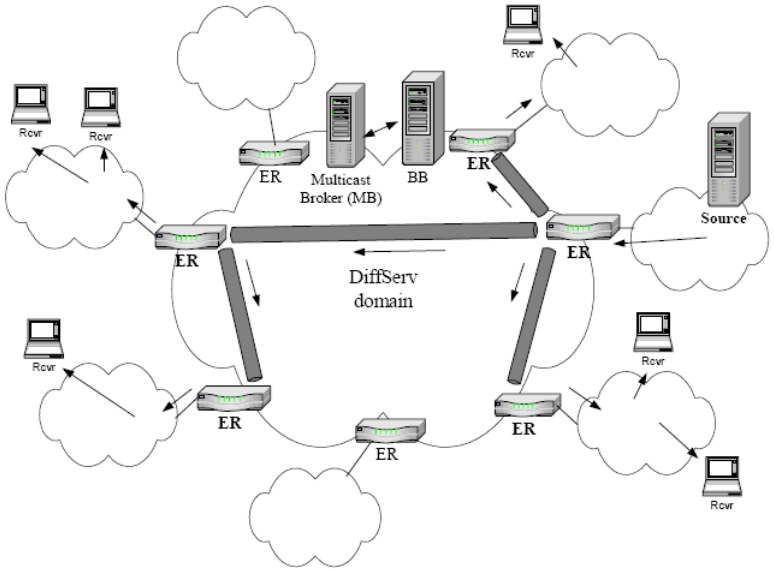


Figure E.1: EBM Architecture. Source:[178]

In EBM, replication of packets occur only at the edge routers. However the replication information can be handled in two different ways:

i. It may be carried in the packet in the form of an encapsulated tree.

ii. Alternatively it may be maintained as state information at the edge routers.

Responsibilities of the Multicast Broker(MB) include tree construction, tree re-arrangement and security management. MB also does admission control, managing the egress join/leave requests from group members. It also manages multicast trees for the DiffServ domain or individual groups at an ingress router. Additionally, the MB manages QoS interactions for multicasting such as resource reservations. The MB

can either be centralized or distributed. The reason for distribution being to avoid the scenario where the MB becomes a single point of failure in the domain. The MB uses an algorithm referred to as *"Edge Cluster Tree (ECT) algorithm"* for tree construction which helps reduce the multicast transport to an edge-to-edge transport functionality.

The principle of ECT algorithm is to cluster together egress points with similar QoS-classes so as to balance the cost of the tree versus the additional hops required for edge-based branching. The ingress node tunnels the packets to clusters that then tunnel the packets to the egress points in their clusters and other downstream clusters. The ECT algorithm itself can be broken into two key phases of *cluster construction* and *cluster linkage*.

EBM defines a *"cluster"* as a collection of all egress points within a given number of hops (H) of an edge router(ER) whose Per-Hop-Behaviors(PHBs) can be satisfied by the *PHB* used for packets sent to *ER*. The cluster construction therefore involves locating all the *ERs* that meet the requirements imposed by this definition. Once identified, the the clusters are linked together via tunnels to construct the multicast distribution tree for the domain.

There are some unclear aspects regarding EBM implementation. For example, it is not explicitly stated whether EBM assumes the presence or absence of a BB in the DiffServ domain. In the event of the BB's presence, its role and that of the Mulitcast Broker(MB) would greatly be conflicting. The MB can be single point of failure therefore for effective EBM the distributed EBM option should be prioritized.

E.2 MPLS Multicast Tree(MMT)

As the name suggests MPLS Multicast Tree[179], is a multicast implementation of the MPLS protocol in which only the branching routers maintain multicast state. It con-

structs a multicast tree by considering only the branching routers along the multicast delivery path. Since only routers at the branching points maintain multicast state, MMT protocol converts multicast flows into "multiple unicast" flows. In addition, MMT also uses a degree of multicast tree aggregation, a concept we explain under the section, *"aggregation-based protocols"* in this paper. Using the aggregation technique, MMT ensures that multiple channels share the ingress and egress LSRs hence several multicast channels share branches of their trees rather than constructing a tree for each channel. Unicast LSPs are used between the branching node routers of the multicast tree.

Just like in other protocols that use "tree managers" or "Multicast Brokers" in the DiffServ domain, MMT introduces a new entity for the purpose of multicast traffic engineering. The entity in MMT is called a *"Network Information Management System (NIMS)"*. Each domain contains a *NIMS* for each group. The *NIMS* is charged with collecting join and leave messages from all group members in that domain. MMT also borrows from PIM-SM[145] in that the NIMS is elected through a mechanism similar to the one used to elect the **RP** router in PIM-SM[145].

The responsibilities of NIMS include:

i. Admission control

ii. Keeping all the necessary information about the LSPs such as:

   - All sources and destinations of each multicast group
   - All the bandwidth associations between all nodes in the network

iii. Keeping information about the network topology. Nodes inform the NIMS of all changes in the network topology due to factors such as LSP and router failures.

iv. Keeping all the necessary information about the LSPs including all sources and destinations of each multicast group as well as all the bandwidth associations.

Again, MMT shares with the QMD approach in terms of the use of a modified version of Dijkstra's algorithm. After collecting all join messages, the NIMS computes the multicast tree for that group in the domain using a constrained form of Dijkstra's algorithm. The computations means discovering all branching node routers for that group. On receiving a multicast message, a branching node router creates a multicast forwarding state for the multicast channel. Once branching node routers and their next hops are identified, packets will be sent from a branching node router to another until they reach their destination.

The key strength of MMT is that packets are sent on branches using established MPLS tunnels between the edge routers through the core routers. Consequently, the multicast LSP construction, the multicast flows association and the multicast traffic aggregation are transformed into simple unicast problems.

A significant drawback is that like the "MB" in EBM protocol, a NIMS in MMT can be a single point of failure therefore distributed NIMS should be used.

Appendix F

Aggregation protocols

The last category in our classification of these protocols can be seen as a special sub-category of the state-based protocols. However, they differ slightly from the fully state-based approaches in that they try to reduce the amount of state information maintained at the core by adding approaches whereby the multicast trees for numerous groups are combined *("aggregated")* into a limited number of ***"super-multicast"*** groups at edge routers. The core routers therefore only service multicast information for the ***"super-multicast"*** groups.

State information maintenance at the core routers still remains their drawback since it conflicts with the DiffServ architecture. Some these protocols include AQoSM[180] and Harmonic DiffServ[181].

F.1 Aggregated QoS Multicast(AQoSM)

AQoSM[180] is not a protocol but an architecture for providing scalable and efficient QoS Multicast in DiffServ networks. In [180], the authors demonstrate how to use the architecture to design MPLS-based AQoSM protocol with admission control and MPLS multicast tree management. AQoSM uses the concept of aggegated multicast but its key innovation is the decoupling of the concept of *"groups"* from the concept

of *"distribution tree"* by ***"mapping"*** many groups to one distribution tree. This way, many groups can be multiplexed on a single tree and a group can be switched easily between distribution trees. This allows for multicast groups to be routed and re-routed very quickly by assigning different labels (e.g. tree IDs) to the packets.

AQoSM is premised on the fact that, aggregation of groups into fewer multicast trees leads to routing state reduction and less tree management overheads thereby complying with the tenets of the DiffServ architecture. Admission control can then be carried out on the level of aggregated trees instead of individual links thereby improving resource efficiency due to statistical multiplexing of multiple groups on a single tree. However it is important to note that like DSMCast[172], AQoSM is intended for QoS provisioning within a single DiffServ domain, particularly the backbone(*transit*) domain and therefore cannot span a global scope like the Internet.

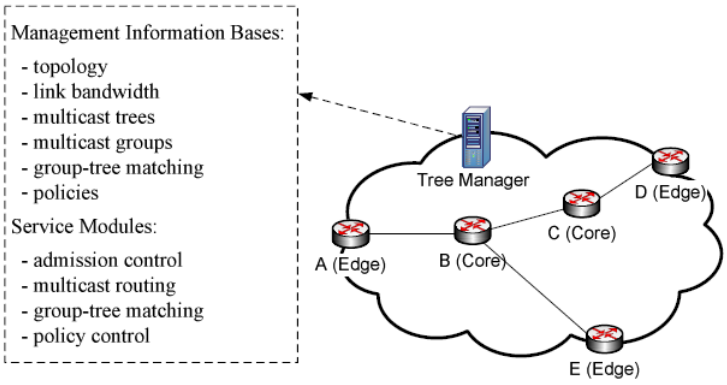


Figure F.1: AQoSM Tree Manager. Source:[180]

To manage groups and trees, AQoSM architecture incorporates a logical entity called *"tree manager"*. Its responsibilities include tree maintenance and "group-to-tree" matching. Figure F.1 illustrates a sample AQoSM network. The *tree manager* consists of several service modules including: admission control, group-tree matching, routing and policy control modules. The *tree manager* also embeds a Management Information Base for collecting up-to-date information including: the network topology,

the available resources, the group membership and their QoS requirements. Additionally, it also keeps the link-state information which it obtains from routers in the domain. Principally, each router in the DiffServ domain is expected to measure its traffic load and send to the tree manager.

We however note the following potential drawbacks in AQoSM.

i. It is not clear how the *tree manager* fits into the typical DiffServ architecture in which the Policy Decision Point(PDP) is in the Bandwidth Broker(BB). This is becasue some of the roles played by the *tree manager* are stipulated to be done by the BB. Probably, it might be helpful if the *tree manager* were an additional module in the BB.

ii. Fitting of too many responsibilities on the tree manager is likely to create a single point of failure in AQoSM network. It would be helpful to expand the AQoSM specification to allow for redundant *tree managers* to allow for fault tolerance in an AQoSM network.

iii. AQoSM, like all other aggregation-based protocols only reduces the amount of state-information maintained by the core routers but does not eradicate it. Therefore the number of aggregates still determines whether the network is truly scalable or not.

F.2 Harmonic DiffServ

Harmonic DiffServ[181] is an approach for providing scalable support of IP multicast for groups with members exhibiting heterogeneous QoS requirements in a DiffServ network. It focuses on provisioning of QoS heterogeneity and mitigating scalability issues due to state-information maintenance in the routers. Harmonic proposes to achieve these objectives through:

i. Logically partitioning the multicast tree into a limited number of *clusters* such that multicast trees in the same cluster aggregate together to share a common multicast address (channel). The purpose of the aggregation is to limit the size of the routing table by merging multicast trees into several *clusters*, each of which then forms a unique multicast group for transmission. Using this approach, the size of the routing table then depends on the number of multicast addresses in use rather than the number of multicast trees.

ii. Further marking packets within each multicast session by a set of DSCPs which lead packets into specific QoS treatments in each router in a way that the respective QoS requirements of every multicast tree are realized properly. In an attempt to meet the DiffServ requirement of reducing the load in the core routers, Harmonic DiffServ also proposes to move major adaptation tasks of clustering and DSCP Management to the edge routers. However the core routers will still need to maintain a few multicast groups (*the clusters*) and run the DiffServ model.

iii. Proposes a heuristic clustering scheme. Based on the heuristic, the scheme is divided into either Fixed Encoding(FE) or Dynamic Encoding (DE)

The number of *sub-clusters* that a *clusters* is divided into is an important factor in Harmonic DiffServ because it determines the number of DSCPs hence should be minimized. Nonetheless, proponents of the Harmonic DiffServ concur that multicast-tree clustering problem used in their model is harder than the graph-coloring problem[182, 183] which is known to be NP-complete. Therefore they propose a heuristic-based encoding approach to it on which DSCP-PHB associations are derived. The encoding approaches are either Fixed Encoding(FE) in which a particular DSCP is associated with a fixed or a dynamic (Dynamic Encoding (DE)) Per-Hop-Behaviour where the DSCP-PHB association varies adaptively. The DE heuristic is found to consume less number of DSCPs than the FE approach. However this comes at the cost of slightly

more computational costs and control overheads.

It is important to note that inspite of reducing the number of multicast trees through clustering, Harmonic DiffServ still suffers some limitation with regards to scalability because the core routers are not completely multicast-unaware. The state-information maintained by the core routers increases with the increase in the number of the computed *clusters*. Another significant drawback of Harmonic DiffServ is the degree of QoS heterogeneity that it can support since it implements *group-level* QoS heterogeneity and not *host-level* QoS heterogenity, implying that the best case scenario is where all members of each *"cluster"* demand similar QoS level.

F.3 Summary

Most of the approaches mentioned actually borrow alot from each other and strict classification for some of them might be hard. However, we classified them based on the dorminant approach in their implementation. From the literature, it is possible to deduce that other than the edge based approaches, all the remaining categories allow for replication of multicast packet to happen at both core and edge routers.

For edge based approaches, replication specifically takes place at the edege routers. It is also important to note that for selective and edge-based approaches, replication information is mostly carried through tunneling techniques. For state-based and aggregation approaches replication information is however retrieved from the multicast routing tables that are maintained as part of multicast state information. The stateless (encapsulation-based) approaches, the replication information is included within the IP packet header. The examples, advantages and disadvantages of these approaches are summarized in table A.3 in section A.2.

Bibliography

[1] S. E. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended lans," *ACM Transactions on Computer Systems*, vol. 8, pp. 85–110, May 1990.

[2] D. R. Cheriton and S. Deering, "Host Groups: A Multicast Extension for Datagram Internetworks," in *Procedings of Symposium on Data Communications*, 1985.

[3] R. Boivie, N. Feldman, Y. Imai, W. Livens, and D. Ooms, "Explicit multicast (xcast) concepts and options," *RFC 5058*, November 2007.

[4] Y. Imai, T. Kurosawa, and E. Muramoto., "Xcast6 (version 2.0) protocol specification,," *Internet Draft, draft-ug-xcast20-protocol-spec-00.txt,*, 2008.

[5] O. E. Abade, N. Kawaguchi, Y. Imai, T. Kurosawa, and E. Muramoto, "Design and implementation of an xcast6 routing engine," *Internet Draft, draft-abade-xcast20-routing-engine-spec-00.txt*, October 2009.

[6] O. E. Abade, K. Kaji, and N. Kawaguchi, "Design, implementation and evaluation of a routing engine for a multipoint communication protocol: Xcast6," *International Journal of Computer Science and Network Security*, vol. 11, pp. 200–209, May 2011.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated services," *RFC 2475*, December 1998.

[8] K. Nichols, S. Blake, F. Bakers, and D. Black, "Definition of the differentiated services field (ds field) in the ipv4 and ipv6 headers," *RFC 2474*, December 1998.

[9] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, "Locator/id separation protocol (lisp)," *IETF draft-ietflisp-22. (Work In Progress)*, February 2012.

[10] A. Varga.., "The omnet++ discrete event simulation system," *Proceedings of the European Simulation Multiconference*, pp. 319–324, June 2001.

[11] "Omnet++ community site."
http://www.omnetpp.org.

[12] "The FreeBSD Project."
http://www.freebsd.org.

[13] K. Kobayashi, A. Ogawa, S. Casner, and C. Bormann, "Rtp payload format for dv (iec 61834) video," *IETF RFC3189*, January 2002.

[14] "DV Format."
http://en.wikipedia.org/wiki/DV.

[15] "Digital Video Encoding (DV, DVCAM, DVCPRO)."
http://www.digitalpreservation.gov/formats/fdd/fdd000183.shtml.

[16] O. E. Abade, K. Kaji, and N. Kawaguchi, "QS-XCAST: A QoS Aware XCAST Implementation," in *Proceedings of Fifth International workshop on OMNeT++, OMNeT++2012, DesenZano, Italy.*, March 2012.

[17] O. E. Abade, K. Kaji, and N. Kawaguchi, "Scalable qos for xcast using differentiated services architecture," *Journal of Information Processing, (IPSJ - JIP). (To appear)*, vol. 21, no. 1, 2013.

[18] "The INET Framework Project site."
http://inet.omnetpp.org.

[19] "IP Multicast."
http://fengnet.com/book/CNF/ch05lev1sec1.html.

[20] S. Deering, "Host extensions for ip multicasting," *RFC 1112*, August 1989.

[21] "IP Multicast."
http://en.wikipedia.org/wiki/Multicast#IP_multicast.

[22] D. Cotroneo, G. Iannello, S. Russo, and G. Ventre, "A real time-based architecture for qos multimedia provisioning," *Microprocessors and Microsystems*, vol. 27, pp. 55–63, March 2003.

[23] D. A. Menasce, H. Ruan, and H. Gomaa, "Qos management in service-oriented architectures," *Performance Evaluation*, vol. 64, pp. 646–663, August 2007.

[24] P. Florissi, Y. Yemini, and D. Florissi, "Qosockets: a new extension to the sockets api for end-to-end application qos management," *Computer Networks*, vol. 35, pp. 57–76, December 2000.

[25] Y. Ito and S. Tasaka, "Feasibility of qos control based on qos mapping over ip networks," *Computer Communications*, vol. 31, pp. 2589–2595, December 2000.

[26] L. Chunlin and L. Layuan, "Qos based resource scheduling by computational economy in computational grid," *Information Processing Letters*, vol. 98, pp. 119–126, May 2006.

[27] B. Sun and L. Li, "Qos-aware multicast routing protocol for ad hoc networks," *Journal of Systems Engineering and Electronics*, vol. 17, pp. 417–422, June 2006.

[28] B. Praveen, J. Praveen, and C. S. R. Murthy, "A survey of differentiated qos schemes in optical burst switched networks," *Optical Switching and Networking*, vol. 3, pp. 134–142, August 2006.

[29] F. Buccafurri, P. D. Meo, M. Fugini, R. Furnari, A. Goy, G. Lax, P. Lops, S. Modafferi, B. Pernici, D. Redavid, G. Semeraro, and D. Ursino, "Analysis of qos in cooperative services for real time applications," *Data & Knowledge Engineering*, vol. 67, pp. 463–484, December 2008.

[30] Y. Cui, J. Wu, and K. Xu, "Precomputation for intra-domain qos routing," *Computer Networks*, vol. 47, pp. 923–937, April 2005.

[31] H. Kochkar, T. Ikenaga, K. Kawahara, and Y. Oie, "Multi-class qos routing strategies based on the network state," *Computer Communications*, vol. 28, pp. 1348–1355, July 2005.

[32] M. Ramalho, "Intra- and interdomain multicast routing protocols: A survey and taxonomy," *IEEE Commun. Surveys and Tutorials*, vol. 3, pp. 2–25, Jan 2000.

[33] J. Hou and B. Wang, "Multicast routing and its qos extension: Problems, algorithms, and protocols," *IEEE Network*, Jan 2000.

[34] J. C. Pasquale, G. C. Polyzos, and G. Xylomenos, "The multimedia multicasting problem.," *Multimedia Systems*, vol. 6, no. 1, pp. 43–59, 1998.

[35] L. Sahasrabuddhe and B. Mukherjee, "Multicast routing algorithms and protocols: A tutorial," *IEEE Network*, Jan 2000.

[36] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman, "An evaluation of scalable application-level multicast built using peer-to-peer overlays," in *Proceedings of IEEE INFOCOM*, 2003.

[37] C. Yeo, B. Lee, and M.H.Er, "A survey of application level multicast techniques," *Computer Communications*, vol. 27, pp. 1547–1568, April 2004.

[38] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable application layer multicast," in *Proceedings of ACM SIGCOMM*, 2002.

[39] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas, "Protocol independent multicast - sparse mode (pim-sm): Protocol specification (revised)," *RFC 4601*, August 2006.

[40] G. Fairhurst, "Pim routing," in *Proceedings of IP Multicast Workshop at Networkshop*, 2006.

[41] H. Holbrook and B. Cain, "Source-specific multicast for ip," *RFC 4607*, August 2006.

[42] F. Baker and P. Savola, "Ingress filtering for multihomed networks," *RFC 3704*, March 2004.

[43] I. Wijnands, A. Boers, and E. Rosen, "The reverse path forwarding (rpf) vector tlv," *RFC 5496*, March 2009.

[44] I. Wu and T. Eckert, "Router-port group management protocol (rgmp)," *RFC 3488*, February 2003.

[45] M. Handley, I. Kouvelas, T. Speakman, and L. Vicisano, "Bidirectional protocol independent multicast (bidir-pim)," *RFC 5015*, October 2007.

[46] Cisco, "Cisco white paper on: Bidirectional pim deployment guide," *Cisco White Paper*, February 2008.

[47] H. Muller and A. Brandstadt, "The np-completeness of steiner tree and dominating set for chordal bipartite graphs," *Theoretical Computer Science*, vol. 53, pp. 257–265, April 1987.

[48] A. Strigel and G. Manimaran, "A surevy of qos multicasting issues," *IEEE Communications Magazine*, pp. 82–87, June 2002.

[49] R. Bless and K. Wehrle, "Ip multicast in differentiated services (ds) networks," *RFC 3754*, 2004.

[50] A. Ballardie, "Core based trees (cbt) multicast routing architecture," *RFC 2201*, 1997.

[51] S. Deering, D. L. Steiin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "The pim architecture for wide-area multicast routing," *ACM Transactions on Networks*, pp. 153–162, April 1996.

[52] D. Waitzman, C. Partridge, and S. Deering., "Distance vector multicast routing protocol," *RFC 1075*, November 1998.

[53] S. Deering, S. Hares, C. Perkins, and R. Perlman, "Overview of the 1998 iab routing workshop," *IETF RFC2902*, August 2000.

[54] R. Boivie and N. Feldman, "Small group multicast," *Internet Draft draft-boivie-sgm-00.txt*, March 2000.

[55] D. Ooms and W. Livens, "Connectionless multicast," *Internet Draft draft-ooms-cl-multicast-00.txt*, June 1999.

[56] I. Yuji, "Multiple destination option on ipv6(mdo6)," *Internet Draft draft-imai-mdo6-02.txt*, September 2000.

[57] L. Aguilar, "Datagram routing for internet multicasting," in *Proceedings of the ACM SIGCOMM symposium on Communications architectures and protocols: tutorials and symposium*, June 1984.

[58] R. Braden, D. Clark, and S. Shenker, "Integrated services in the internet architecture: an overview," *IETF RFC1633*, June 1994.

[59] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource reservation protocol (rsvp)," *IETF RFC2205*, September 1997.

[60] L. Siregar, R. Budiarto, M. Omar, and A. Rosli, "Quality of service performance for xcast in ipv6 network," *Proceedings of DFmA 2008*, October 21-22 2008.

[61] L. Siregar, R. Aji, Z. Hasibuan, and R. Budiarto, "Quality of service for iptv using xcast in ipv6 network," *Proceedings of NETAPPS 2010*, September 22-23 2010.

[62] "Differentiated Services."
http://en.wikipedia.org/wiki/Differentiated_services.

[63] "Differentiated Service Architecture in IP Network."
http://www.javvin.com/protocolDiffServ.html.

[64] B. Davie, A. Charny, J. Bennett, K. Benson, J. L. Boudec, W. Courtney, S. Davari, PMC-Sierra, V. Firoiu, and D. Stiliadis, "An expedited forwarding phb (per-hop behavior)," *RFC 3246*, March 2002.

[65] J. Heinanen, T. Finland, F. Baker, W. Weiss, and J. Wroclawski, "Assured forwarding phb group," *RFC 2597*, June 1999.

[66] D. Grossman, "New terminology and clarifications for diffserv," *RFC 3260*, April 2002.

[67] X. He, Q. Chu, and M. Zhu, "Minimizing preemption cost for path selection in diffserv-ware mpls networks," *Computer Communications*, vol. 29, pp. 3825–3832, November 2006.

[68] T. Kimura and S. Kamei, "Qos evaluation of diffserv-aware constraint-based routing schemes for multi-protocol label switching networks," *Computer Communications*, vol. 27, pp. 147–152, February 2004.

[69] A. M. Alkharasani and M. Othman, "M2i2tswtcm: A new efficient optimization marker algorithm to improve fairness bandwidth in diffserv networks," *Journal of Network and Computer Applications*, p. http://dx.doi.org/10.1016/j.jnca.2012.01.021, 2012.

[70] M. Elshaikh, M. Othman, S. Shamala, and J. Desa, "A new fair marker algorithm for diffserv networks," *Computer Communications*, vol. 31, pp. 3064–3070, September 2008.

[71] C. Casetti and M. Mellia, "A lightweight marker with partial state information for diffserv networks," *Computer Networks*, vol. 49, pp. 541–560, November 2005.

[72] C. Bourasa and A. Sevasti, "Sla-based qos pricing in diffserv networks," *Computer Communications*, vol. 27, pp. 1868–1880, December 2004.

[73] W.-H. Hsu, Y.-P. Shieh, and J. Chen, "Multiple path selection algorithm for diffserv-aware mpls traffic engineering," *Computer Communications*, vol. 33, pp. 25–41, August 2010.

[74] I. Akyildiz, T. Anjali, L. Chen, J. de Oliveira, C. Scoglio, A. Sciuto, J. Smith, and G. Uhl, "A new traffic engineering manager for diffserv/mpls networks: design and implementation on an ip qos testbed," *Computer Communications*, vol. 26, pp. 388–403, March 2003.

[75] H. Chen, H.-K. Su, and B.-C. Cheng, "An objective-oriented service model for voip overlay networks over diffserv/mpls networks," *Computer Communications*, vol. 30, pp. 3055–3062, November 2007.

[76] R. Tandra, N. Hemachandra, and D. Manjunath, "Diffserv node with join minimum cost queue policy and multiclass traffic," *Performance Evaluation*, vol. 55, pp. 541–560, August 2003.

[77] Z. Quan and J.-M. Chung, "Queue length analysis of non-preemptive diffserv networks," *AEU - International Journal of Electronics and Communications*, vol. 57, pp. 338–340, March 2003.

[78] W.-H. Hsu, Y.-P. Shieh, and J. Chen, "Analysis of nonpreemptive priority queueing of diffserv networks with bulk arrivals," *AEU - International Journal of Electronics and Communications*, vol. 33, no. 13, pp. 409–414, 2003.

[79] C. Awada, B. Sanso, and A. Girard, "Diffserv for differentiated reliability in meshed ip/wdm networks," *Computer Networks*, vol. 52, pp. 1988–2012, July 2008.

[80] J. Yao, L. Xiao, C. Nie, D. Tung, C. Wong, and Y. H. Chew, "Resource allocation for end-to-end qos provisioning in a hybrid wireless wcdma and wireline ip-based diffserv network," *European Journal of Operational Research*, vol. 191, pp. 1139–1160, December 2008.

[81] Q. LIU, H. LI, Y. feng JI, and Y. jun QIAO, "Resources allocation in an intserv/diffserv integrated epon system," *The Journal of China Universities of Posts and Telecommunications*, vol. 16, pp. 108–113, June 2009.

[82] S. D. Cnoddera, O. Elloumib, and K. Pauwels, "Rate adaptive shaping for the efficient transport of data traffic in diffserv networks," *Computer Networks*, vol. 35, pp. 263–285, February 2001.

[83] T. Ahmed, R. Boutaba, and A. Mehaoua, "A measurement-based approach for dynamic qos adaptation in diffserv networks," *Computer Communications*, vol. 28, pp. 2020–2033, November 2005.

[84] I. T. Okumus, H. A. Mantar, J. Hwang, and S. J. Chapin, "Inter-domain qos routing on diffserv networks: a region-based approach," *Computer Communications*, vol. 28, pp. 174–188, February 2005.

[85] H. Liu, H. Xu, and S. Zhao, "Consistent-degradation macroblock grouping for parallel video streams over diffserv networks," *Computer Communications*, vol. 35, pp. 151–158, January 2012.

[86] V. Fuller, D. Farinacci, D. Meyer, and D. Lewis, "Lisp alternate topology (lisp+alt)," *IETF draft-ietf-lispalt-05. Work in Progress*, October 2010.

[87] C. Perkins, "Ip mobility support for ipv4," *IETF RFC3344*, August 2002.

[88] D. Johnson, C. Perkins, and J. Arkko, "Ip mobility support for ipv6," *IETF RFC377544*, June 2004.

[89] D. Farinacci, D. Meyer, and D. Lewis, "Lisp mobile node," *IETF draft-meyer-lisp-mn-07. (Work In Progress)*, April 2012.

[90] N. Kawaguchi, S. Nishiura, O. E. Abade, T. Kurosawa, T. Jinmei, and E. Muramoto, "Nat free open source 3d video conferencing using samtk and application layer router," in *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, CCNC09, Las Vegas USA.*, January 2009.

[91] "Scalable Adaptive Multicast Toolkit (SAMTK) project." http://www.samtk.org.

[92] T. Yoneda, E. Muramoto, C.-C. Hsu, K. Konishi, and T. Matsumoto, "Evaluation of congestion control method using multiple-constant bit rate streams over xcast6," in *Proceedings of Internet Conference, IC2005, Tokyo, Japan.*, 2005.

[93] T. Yoneda, E. Muramoto, M. Hsu, K. Konishi, and T. Kawahara, "Sender initiated congestion control using multiple constant bit rate for small group multicast," *IPSJ Transaction on Information and Media Technologies*, vol. 24, pp. 808–820, February 2007.

[94] S. Krishnan, "The case against hop-by-hop options," *IETF draft-krishnan-ipv6-hopbyhop-04. (Work In Progress)*, March 2010.

[95] B. Fenner, "Experimental values in ipv4, ipv6, icmpv4, icmpv6, udp, and tcp headers," *RFC 4727*, November 2006.

[96] K. Ramakrishnan, S. Floyd, and D. Black, "The addition of explicit congestion notification (ecn) to ip," *RFC 3168*, September 2001.

[97] Y. Imai., "Bsd implementations of xcast6," *Proceedings of ASiaBSDCon2008 Tokyo,*, March 2008.

[98] "Develop with the JUNOS SDK."
https://developer.juniper.net/content/jdn/en/develop-overview/junos-sdk/getting-started.html.

[99] "Cisco Application eXtension Platform."
http://www.cisco.com/en/US/prod/collateral/routers/ps9701/qa_c67_463943.html.

[100] "Network Processors."
http://en.wikipedia.org/wiki/Network_processor.

[101] "An Introduction to Network Processors."
http://140.116.82.38/members/html/ms03/dclin/technique_paper/NP/network_processors_introduction.pdf.

[102] "Open Flow."
http://www.openflow.org/.

[103] D. Harrington, R. Presuhn, and B. Wijnen, "An architecture for describing simple network management protocol (snmp) management frameworks," *RFC 3411*, December 2002.

[104] R. E. Ed, "Netconf configuration protocol," *RFC 4741*, December 2006.

[105] "Widely Integrated Distributed Environment (WIDE) project."
http://www.wide.ad.jp.

[106] "The FreeBSD Project PMC tools."
http://wiki.freebsd.org/PmcTools.

[107] H. Kim, H. Sung, and H. Lee, "Performance analysis of the tcp/ip protocol under unix operating systems for high performance computing and communications," in *Proceedings of the High-Performance Computing on the Information Superhighway, HPC-Asia '97.*, May 1997.

[108] W. Matthias, C. Thomas, and W. Georg, "Oasis: An overlay abstraction for re-architecting large scale internet group services," *Berlin Heidelberg:Springer-Verlag - Lecture Notes in Computer Science*, vol. 5630, pp. 95–106, June 2009.

[109] X. Shen, H. Yu, J. Buford, and M. Akon, "Multicast routing in structured overlays and hybrid networks," *Berlin Heidelberg:Springer-Verlag - Handbook of Peer-to-Peer Networking*, June 2009.

[110] C. Diot, B. N. Leivine, B. Lyles, H. Kassem, and D. Balensiefen, "Deployment issues for the ip multicast service and architecture," *IEEE Network*, vol. 14, pp. 78–88, January 2000.

[111] T. Hardjono. and G. Tsudik., "Ip multicast security: Issues and directions," *Annales de Tlcommunications*, vol. 55, pp. 324–340, January 2000.

[112] N. Y. Times, "Cheap, ultrafast broadband? hong kong has it," *New York Times,* `http://www.nytimes.com/2011/03/06/business/06digi.html?\_r=1s`, November 2007.

[113] M. Kolberg and J. Burford, "An xcast multicast implementation for the oversim simulator," *Proceedings of Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE*, January 2010.

[114] "The overlay simulator `http://www.oversim.org/wiki`,"

[115] M. Framework, "Mobility framework `http://mobility-fw.sourceforge.net/`,"

[116] T. Gamer and M. Scharf., "Realistic simulation environments for ip-based networks," *Proceedings of 1st International Workshop on OMNeT++, ICST, Marseille, France*, March 2008.

[117] A. El-Sayed, V. Roca, and L. Mathy, "A survey of proposals for an alternative group communication service," *IEEE Networks*, vol. 17, no. 1, pp. 46–51, 2003.

[118] A. Popescu, D. Constantinescu, D. Erman, and D. Ilie, "A survey of reliable multicast communication," *Proceedings of the 3rd Euro-NGI conference on Next Generation Internet Networks (NGI 2007), Trondheim, Norway*, 2007.

[119] O. E. Abade, K. Kaji, and N. Kawaguchi, "Quantitative Simulation of XCAST6 Performance Using OMNeT++," in *Proceedings of Asian Internet Engineering Conference, AINTEC'11, Bangkok, Thailand.*, 2011.

[120] S. S.-B. Mozafar Bag-Mohammadi, Nasser Yazdani, "On the efficiency of explicit multicast routing protocols," *Proceedings of the 10th IEEE Symposium on Computers and Communications*, 2005.

[121] M. Bag-Mohammadi and N. Yazdani, "A fast and efficient explicit multicast routing protocol," *IEICE Transaction on Communications*, vol. E88, pp. 4000 – 4007, October 2005.

[122] F. Y. Alzyoud, T.-C. Wan, and I. J. Mohamad, "The effect of using xcast based routing protocol in wireless ad hoc network," *IEEE TENCON 2009*, November 2009.

[123] A. Strigel and G. Manimaran, "A survey of qos multicasting issues," *IEEE Communications Magazine*, pp. 82 – 87, Junet 2002.

[124] A. Strigel and G. Manimaran, "A scalable approach for diffserv multicasting," *IEEE International Conference on Communications*, vol. 8, pp. 2327 – 2331, August 2001.

[125] A. Strigel and G. Manimaran, "Dsmcast: A scalable approach for diffserv multicasting," *Computer Networks*, vol. 44, no. 6, pp. 713–735, 2004.

[126] J.-H. Cui, L. Lao, M. Faloutsos, and M. Gerla, "Aqosm: Scalable qos multicast provisioning in diffserv networks," *Computer Networks*, vol. 50, pp. 80 – 105, 2006.

[127] S.-R. Tong and C.-C. Chang, "Harmonic diffserv: Scalable support of ip multicast with qos heterogeneity in diffserv backbone networks," *Computer Communications*, vol. 29, pp. 1780 – 1797, 2006.

[128] B. Yang and P. Mohapatra, "Multicasting in differentiated service domains," *Proceedings of IEEE GLOBECOM*, 2002.

[129] Z. Li and P. Mohapatra, "Qos-aware multicasting in diffserv domains," *Proceedings of Global Internet Symposium*, 2002.

[130] K. Nichols, V. Jacobson, and L. Zhang, "A two-bit differentiated services architecture for the internet," *RFC 2638*, July 1997.

[131] J. Evans and C. Filsfils, *Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice*. 500 Sansome Street, Suite 400, San Francisco, CA 94111 USA: Morgan Kaufmann, 2007.

[132] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, "Network configuration protocol (netconf)," *RFC 6241*, June 2011.

[133] P. Shafer, "An architecture for network management using netconf and yang," *RFC 6244*, June 2011.

[134] K. Singh, "Router buffer traffic load calculation based on a tcp congestion control algorithm," *International Journal of Computational Engineering & Management*, vol. 15, pp. 20–23, January 2012.

[135] A. G. Ali BOUDANI and B. COUSIN, "Gxcast: Generalized explicit multicast routing protocol," *Proceedings of International Symposiym on Computers and Communications, (IEEE ISCC 2004)*, June 2004.

[136] J. Nonnenmacher and E. W. Biersack, "Scalable feedback for large groups," *IEEE Transactions on Networking*, vol. 7, pp. 375–386, June 1999.

[137] C. Filsfils and J. Evans, "Deploying diffserv in backbone networks for tight sla control," *IEEE Internet Computing*, vol. 15, pp. 58–65, January 2005.

[138] D. Massey, L. Wang, B. Zhang, and L. Zhang, "A proposal for scalable internet routing and addressing," *IETF Internet Draft, Work in Progress, draft-wang-ietf-efit-01.txt*, February 2007.

[139] T. Narten, "On the scalability of internet routing," *IETF Internet Draft, Work in Progress, draft-wang-ietf-efit-01.txt*, February 2010.

[140] D. Meyer, L. Zhang, and K. Fall, "Report from the iab workshop on routing and addressing," *IETF RFC 4984*, September 2007.

[141] F. Templin, "The ipvlx architecture," *IETF Interner Draft: draft-templin-ipvlx-08.txt*, May 2007.

[142] C. Vogt, "Six/one: A solution for routing and addressing in ipv6," *IETF Internet Draft: draft-vogt-rrg-six-one-02*, October 2009.

[143] C. Vogt, "Ivip (internet vastly improved plumbing) architecture," *IETF Internet Draft: draft-whittle-ivip-arch-03.txt*, January 2010.

[144] F. Templin, "The subnetwork encapsulation and adaptation layer (seal)," *IETF RFC5320*, February 2010.

[145] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast-sparse mode (pim-sm)," *RFC 2362*, 1998.

[146] J. Tian and G. Neufeld, "Forwarding state reduction for sparse mode multicast communications," *Proceedings of IEEE INFOCOM*, March 1998.

[147] D. Thaler and M. Handley, "On the aggregability of multicast forwarding state," *Proceedings of IEEE INFOCOM*, March 2000.

[148] I. Stoica and H. Zhang, "REUNITE: a recursive unicast approach to multicast," in *Proceedings of IEEE INFOCOM*, 2000.

[149] P. Radoslavov, D. Estring, and R. Govindan, "Exploiting the bandwidth-memory tradeoff in multicast state aggregation," *Technical Report, USC Department of CS*, vol. 99-697, July 1999.

[150] L. Costa, S. Fdida, and O. Duarte, "Hop-by-hop multicast routing protocol," *Proceedings of SIGCOMM'01*, August 2001.

[151] R. Sriram, G. Marimaran, and C. S. R. Murthy, "Preferred link-based delay-constrained least cost routing in wide area networks," *Computer Communications*, vol. 21, no. 18, pp. 1655–1669, 1998.

[152] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-Aware Multicast Routing Protocol," in *Procedings of IEEE INFOCOM*, 2000.

[153] G. Manimaran, H. S. Rahul, C. Siva, and R. Murthy, "A new distributed route selection approach for channel establishment in real-time networks," *IEEE/ACM Transactions on Networks*, vol. 7, pp. 698–709, October 1999.

[154] R. Sriram, G. Manimaran, and C. S. R. Murthy, "A rearrangeable algorithm for the construction of delay-constrained dynamic multicast trees," *IEEE/ACM Transactions on Networks*, vol. 7, pp. 514–529, August 1999.

[155] C. Donahoo and Zegura, "Core migration for dynamic multicast routing," in *Proceedings of ICCCN*, 1995.

[156] E. Fleury, Y. Huang, and P. K. McKinley, "On the performance and feasibility of multicast core selection heuristics," in *Proceedings of ICCCN*, 1998.

[157] A. Striegel and G. Manimaran, "A scalable protocol for member join/leave in diffserve multicast," in *Proceedings of IEEE LCN 2001, Tampa, FL*, 2001.

[158] L. Schwiebert and R. Chintalapati, "Improved fault recovery for core based trees," *Computer Communications*, vol. 23, April 2000.

[159] C.-L. Chen, "A study of ipv6 labeling forwarding model supporting diffserv," *Procedia Engineering*, vol. 15, pp. 5590–5594, 2011.

[160] Z. Mammeri, "Framework for parameter mapping to provide end-to-end qos guarantees in intserv/diffserv architectures," *Computer Communications*, vol. 28, pp. 1074–1092, June 2005.

[161] M. Brunner, A. Gonzalez, and P. Martinez, "From dynamic ip transport service ordering to diffserv network configuration," *Computer Networks*, vol. 43, pp. 25–41, September 2003.

[162] K.-S. S. Eun-Hee Cho and and S.-J. Yoo, "Sip-based qos support architecture and session management in a combined intserv and diffserv networks," *Computer Communications*, vol. 29, pp. 2996–3009, September 2006.

[163] M. D. Stojanovic and V. S. Acimovic-Raspopovic, "On efficient traffic engineering with dv-based routing protocols in diffserv-aware ip networks," *AEU - International Journal of Electronics and Communications*, vol. 60, pp. 387–398, May 2006.

[164] M. Li, D. B. Hoang, and A. J. Simmonds, "Fair intelligent admission control over resource-feedback diffserv network," *Computer Communications*, vol. 28, pp. 1770–1777, September 2005.

[165] F. Faucher, L. Wu, B. Davie, S. Davari, P. Vaananen, R. Krishnan, P. Cheval, and J. Heinanen, "Multi-protocol label switching support of differentiated services," *IETF RFC 3270*, May 2002.

[166] A. Fei, J. Cui, M. Gerla, and M. Faloutsos, "Aggregated multicast: an approach to reduce multicast state," in *Proceedings of the sixth Global Internet Symposium (GI2001)*, November 2001.

[167] A. Fei and M. Gerla, "Receiver initiated multicasting with muliple qos constraints," in *Proceedings of IEEE INFOCOM*, March 2000.

[168] S. Ganti, N. Seddigh, and B. Nandy, "Mpls support of differentiated services using e-lsp," *Internet Draft: draft-ietf-mpls-diff-ext.txt.00*, April 2001.

[169] B. Yang and P. Mohapatra, "Multicasting in Differentiated Services Domain," in *Proceedings of IEEE GLOBECOM*, 2002.

[170] G. Bianchi, N. Blefari-Melazzi, G. Bonafede, and E. Tintinelli, "QUASIMODO: QUAlity of ServIce-aware Multicasting Over Diffserv and Overlay networks," in *Proceedings of IEEE Network*, 2003.

[171] B. Fenner, H. He, B. Haberman, and H. Sandick, "Internet group management protocol (igmp) / multicast listener discovery (mld)-based multicast forwarding (igmp/mld proxying)," *RFC 3754*, 2006.

[172] A. Striegel and G. Manimaran, "Dsmcast: A scalable approach for diffserv multicasting," *Computer Networks*, vol. 44, pp. 713–735, April 2004.

[173] "Bandwidth Broker."
http://en.wikipedia.org/Bandwidth_Broker.

[174] L. Blazevic and J.-Y. L. Boudec, "Distributed Core Multicast (DCM): a multicast routing protocol for many groups with few receivers ," in *Proceedings of Networked Group Communication.*, 1999.

[175] Z. Li and P. Mohapatra, "QoS-aware Multicasting in DiffServ Domains," in *Proceedings of Global Internet Symposium.*, 2002.

[176] L. Henrique, M. K. Costa, S. Fdida, and O. Duarte, "Hop by hop multicast routing protocol - ACM Digital Library," in *Proceedings of SIGCOMM*, 2001.

[177] J. Moy, "Ospf version 2," *RFC 2328*, 1998.

[178] A. Strigel, A. Bouabdallah, H. Bettahar, and G. Manimaran, "EBM: A new approach for scalable multicasting," in *Proceedings of Fifth International Workshop on Network Group Communication (NGC).*, September 2003.

[179] A. Boudani, B. Cousin, and J.-M. Bonnin, "MPLS Multicast Traffic Engineering," in *Proceedings of IEEE GLOBECOM*, 2005.

[180] J.-H. Cui, L. Lao, M. Faloutsos, and M. Gerla, "Aqosm: Scalable qos multicast provisioning in diffserv networks," *Computer Networks*, vol. 50, pp. 80–105, March 2006.

[181] S.-R. Tong and C.-C. Chang, "Harmonic diffserv: Scalable support of ip multicast with qos heterogeneity in diffserv backbone networks," *Computer Communications*, vol. 29, pp. 1780–1797, '.

[182] "Graph coloring."
http://en.wikipedia.org/wiki/Graph_coloring.

[183] I. Mendez-Diaz and P. Zabala, "A cutting plane algorithm for graph coloring," *Discrete Applied Mathematics*, vol. 156, pp. 159–179, April 2008.

List of Publications

O. E. Abade, K. Kaji, and N. Kawaguchi, "Scalable qos for xcast using differentiated services architecture," *Journal of Information Processing, (IPSJ - JIP). (To appear)*, vol. 21, no. 1, 2013.

O. E. Abade, K. Kaji, and N. Kawaguchi, "Design, implementation and evaluation of a routing engine for a multipoint communication protocol: Xcast6," *International Journal of Computer Science and Network Security*, vol. 11, pp. 200–209, May 2011.

O. E. Abade, K. Kaji, and N. Kawaguchi, "QS-XCAST: A QoS Aware XCAST Implementation," in *Proceedings of Fifth International workshop on OMNeT++, OMNeT++2012, DesenZano, Italy.*, March 2012.

O. E. Abade, K. Kaji, and N. Kawaguchi, "Quantitative Simulation of XCAST6 Performance Using OMNeT++," in *Proceedings of Asian Internet Engineering Conference, AINTEC'11, Bangkok, Thailand.*, 2011.

N. Kawaguchi, S. Nishiura, O. E. Abade, T. Kurosawa, T. Jinmei, and E. Muramoto, "Nat free open source 3d video conferencing using samtk and application layer router," in *Proceedings of the 6th IEEE Conference on Consumer Communications and Networking Conference, CCNC09, Las Vegas USA.*, January 2009.

O. E. Abade, N. Kawaguchi, Y. Imai, T. Kurosawa, and E. Muramoto, "Design and

implementation of an xcast6 routing engine," *Internet Draft, draft-abade-xcast20-routing-engine-spec-00.txt*, October 2009.

Acknowledgement

This dissertation, submitted in partial fulfillment of the requirements for the award of the degree of Doctor of Engineering from the Graduate School of Engineering, department of Computational Science and Engineering of Nagoya University has been realized through the assistance from many people to whom I humbly express my gratitude.

First I would like to thank Professor Nobuo Kawaguchi, who did not only grant me the chance to join Nagoya University by accepting me into his laboratory as a government scholar but also mentored me during the period of this research. I extend my heartfelt gratitude to all my doctoral committee members. Your guidance made me improve on this work greatly. I personally thank my examiners; Professor Takeshi Furuhashi and Professor Tetsu Iwata both from the department of Computational Science and Engineering, Nagoya University together with Professor Akira Kato of Keio University for all your dedication to read and critique different pieces of this research work.

All members of the Ubiquitous Communications Laboratory, am grateful for your assistance. I further extend my special gratitude to all members of my family. Your love, prayers, support and encouragement kept me going even when things were never easy. I am forever grateful and proud to have been born a midst you. To my fiancee, Jessica, your love and patience with me despite the massive geographical separation are virtues I will forever cherish.

Finally, I humbly thank the special team of people, the XCAST funs club; Eichi Muramoto, Yuji Imai and Takahiro Kurosawa, who made me find a topic to research on. May you be blessed abundantly. To everyone who made my life easier in Japan, thank you very much. God bless you all.

Realization of Multipoint Communication over the Internet using XCAST.