Journal of Information Processing Vol.21 No.1 131–144 (Jan. 2013)

[DOI: 10.2197/ipsjjip.21.131]

Regular Paper

Scalable QoS for XCAST Using Differentiated Services Architecture

Odira Elisha Abade[1,2,3,a)] Katsuhiko Kaji[1,2,b)] Nobuo Kawaguchi[1,2,c)]

Received: February 1, 2012, Accepted: September 10, 2012

Abstract: Explicit multiunicast (XCAST) has been proposed as a multicasting scheme with complementary scaling properties which can solve the scalability problems of conventional IP Multicast. XCAST is suitable for videoconferencing, online games and IPTV. This paper 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.

Keywords: XCAST6, QoS, multipoint communication, DiffServ, QS-XCAST, collusion attack, Good Neighbour Effect

1. 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 [1], [2] 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) [1], [2], [3], [4], [5].

Explicit multiunicast (XCAST) [6], 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 [7], [8], [9], [10], [11], [12], [13] and implementation [6], [14] in the Internet leaving out its Quality of Service. In addition to difficulties facing muticast QoS provisioning [15], 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 [6]. Therefore most of the current solutions on integrating DiffServ with Multicast which are dependent on construction and aggregation of multicast trees

[1] Graduate School of Engineering, Nagoya University, Nagoya, Aichi 464–8603, Japan
[2] WIDE Project, Japan
[3] School of Computing & Informatics, University of Nairobi, Nairobi, Kenya
[a)] abade@ucl.nuee.nagoya-u.ac.jp
[b)] kaji@nuee.nagoya-u.ac.jp
[c)] kawaguti@nagoya-u.jp

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 paper 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++ [16], [17] 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 paper is organized as follows. In the next section we briefly describe the XCAST protocol and then highlight the issues that complicate XCAST-DiffServ integration. We also briefly introduce currently existing DiffServ-Multicast integration approaches and explain why they are not applicable for XCAST-DiffServ integration. In section three we describe our proposed solution and in section four 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. Conclusion and future work are given in section five.

2. XCAST6 Overview and QoS Multicasting

XCAST protocol concept and options have been defined by the IRTF in Ref. [6] for both IPv4 and IPv6. The implementation of XCAST on IPv6 is usually referred to as XCAST6 [6], [12], which is the focus of our work.

2.1 XCAST6 Overview

In contrast to the conventional IP Multicast and other multicast variants [18], [19], in XCAST6, the sender explicitly specifies the destination addresses of all receivers as a list of unicast addresses embedded in the IPv6 packet header. Succinctly, the sender embeds a list of IPv6 addresses of the destinations in the routing extension header of the IPv6 packet and then sends the packet to a router. Along the transmission path, each router examines the IPv6 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 XCAST6 header to each of the identified next hops. The process is repeated until all the destinations are reached. The XCAST6 packet header also comprises of a bitmap with bits corresponding to each destination, which

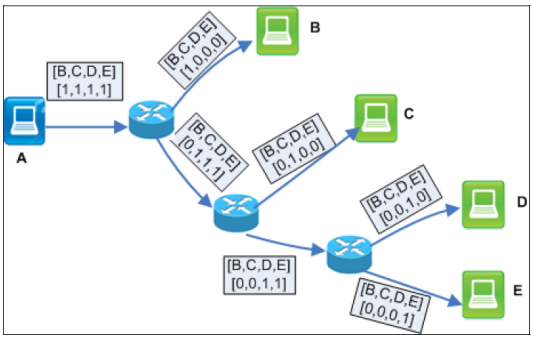


Fig. 1 XCAST6 overview.

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 XCAST6 packet during replication. In **Fig. 1**, the sender (A) sends an XCAST6 packet to B, C, D and E. The destination addresses B, C, D, E have corresponding bitmaps which if set means the packet is to be delivered to a corresponding destination and if reset, means otherwise. On each XCAST6 router, if need be, the XCAST6 packet is duplicated, bitmaps updated and delivered upward according to the destination's next hop with respect to the current branching router.

2.2 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 mul-

tiple 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 Ref. [6] 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 **Fig. 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*) [3], *AF41*, *AF31*, *AF21* [4] and *Best Effort* (*BE*) [5] 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.

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*),

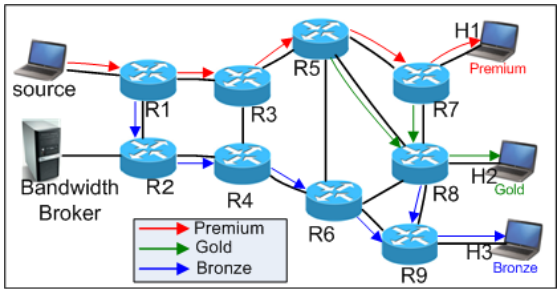


Fig. 2 XCAST6 network with heterogeneous QoS requirements.

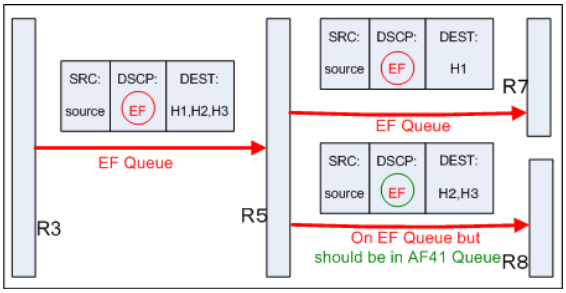


Fig. 3 XCAST6 DSCP confusion problem.

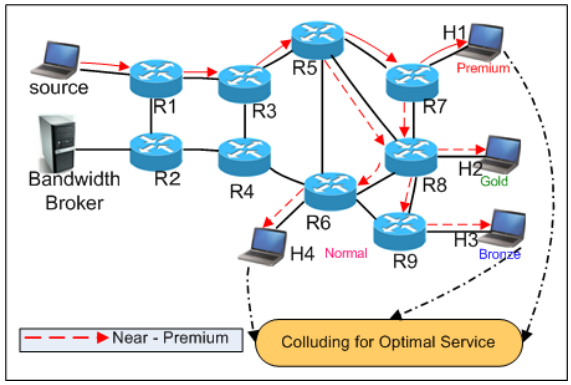


Fig. 4 XCAST6 collusion attack problem.

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 .

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 **Fig. 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*:

In **Fig. 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

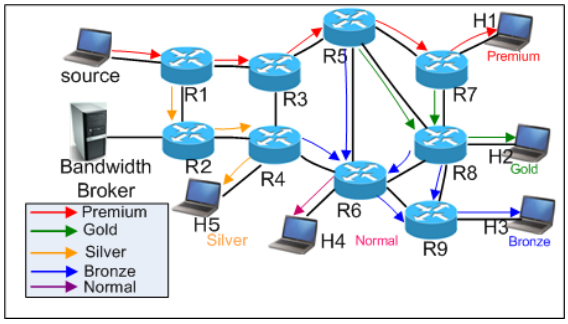


Fig. 5 XCAST links exhibiting the allowable DSCPs.

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 **Fig. 5**.

2.3 Previous Work on XCAST QoS Provisioning

While alot 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 Ref. [20], 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 Ref. [21], 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 [6] but use DiffServ architecture for QoS provisioning.

2.4 Existing Multipoint DiffServ Solutions

In Section 8.3 of XCAST concepts and options [6], 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 [22], [23] 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. Cui et al., proposed AQoSM [24] that uses the con-

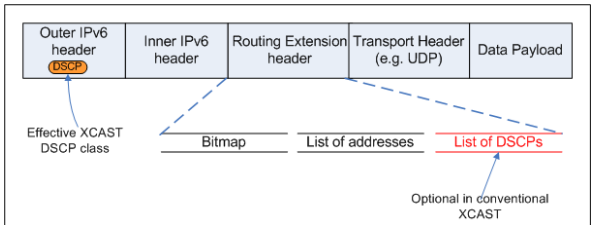


Fig. 6 A block summary of QoS-aware XCAST header.

cept 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 [25], DAM (DiffServ Aware Multicasting) [26] and QMD (QoS-aware Multicasting in DiffServ domains) [27] 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.

3. 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 Fig. 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 Ref. [6], the XCAST6 header will be as shown in **Fig. 6**. The DSCP class of the XCAST packet is therefore determined by the DSCP value in the outer IPv6 header.

3.1 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 [28]. 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 Algorithm 2.

Algorithm 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).

Algorithm 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 the 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 the 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.

3.2 Algorithms for the Proposed Solutions

3.2.1 The Extended XCAST Processing Algorithm

For solution (1) in Section 3.1 above, we modify the XCAST processing algorithm at the routers according to *Algorithm 1*. When used with the QoS network in Fig. 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.

3.2.2 Receiver Initiated QoS Level Assignment

RFC2638 [28] 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 3.1 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 2.

4. 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 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.

4.1 Simulation Model

The proposed QoS aware XCAST is tested using OMNeT++ simulation tool [16], [17], [29]. OMNeT++ currently does not have inbuilt XCAST header encapsulation and routing models. However, previously [9], 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 back-

Table 1 Simulation parameters.

Simulation parameter	Value
Message size	1,450 Bytes
Message frequency	50 ms
Queueing scheme	Shown in Tables 2 and 3
Max Queue capacity	20 MB
MAC Tx rate	100 Mbps
Background Traffic frequency	50 ms

Table 2 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

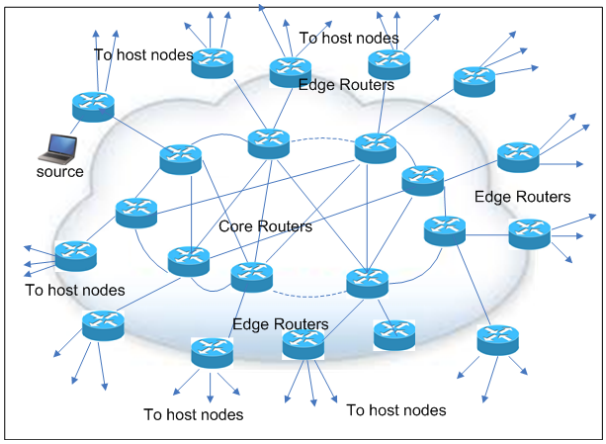


Fig. 7 Model network for IPTV Service.

bone network while the edge routers form the points where IPTV clients are hooked onto the network similar to the illustration in **Fig. 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 1**.

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 2**. 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 4.1.1 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 calculated.

4.1.1 DiffServ Parameters

We implemented packet metering to check on in-profile and out-profile packets using leaky bucket [30] and token bucket [30] 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 [30] queue with leaky bucket while RIO queues [30] with token buckets were used for all the AF classes and the BE class.

Parameters for these data structures are shown in **Table 3**. 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 **Fig. 8** (a) instead of "*drop probability*". We have an array of "RIO queues" as shown in Fig. 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.

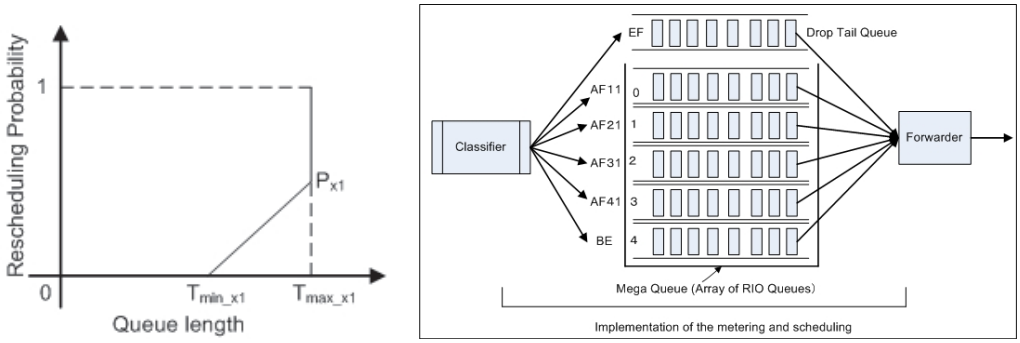
4.1.2 Path Construction and Packet Delivery to Receivers

XCAST does not depend on delivery tree construction. Instead, as specified in Sections 3 and 4 of XCAST RFC document [6], 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 intermediate node an XCAST packet passes through. In our model, the routing table of each node was initially con-

Table 3 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.



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

Fig. 8 Implementation of DiffServ in model routers.

structed using a NETCONF [31], [32] based XML file. We added methods in the "*RoutingTable6*" *module* of OMNeT++ which parse the XML file based on NETCONF XML DTDs [31]. 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.

4.1.3 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 [6], 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 1,500 Bytes. In our case, we embedded up to 75 destinations with a payload data of approximately 256 Bytes hence the total message length

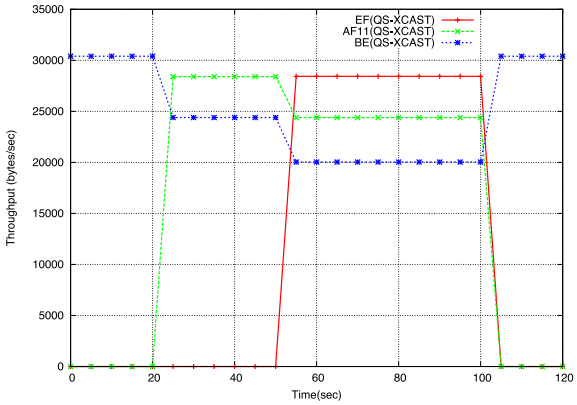


Fig. 9 Service differentiation verification using a group of 30 receivers.

was 1,450 Bytes.

4.1.4 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 **Fig. 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 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.

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 4.1.

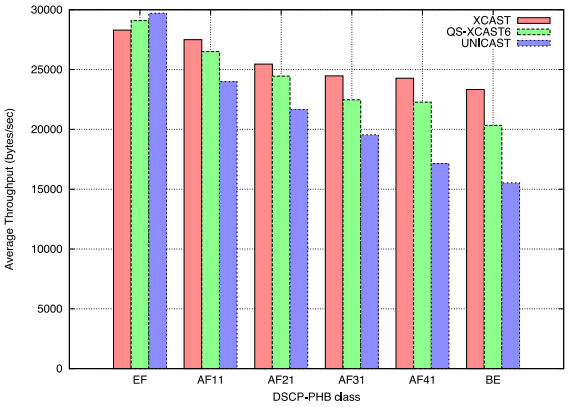


Fig. 10 Comparative average throughput.

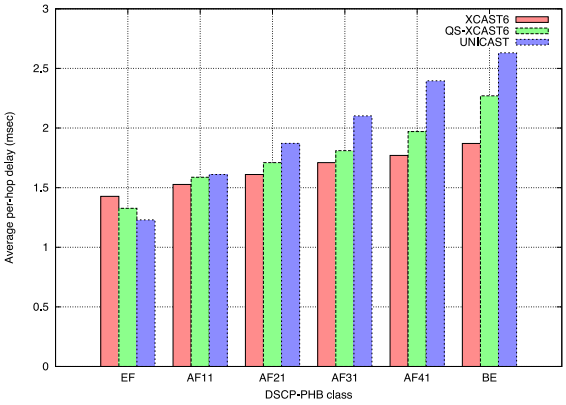


Fig. 11 Comparative average per-hop delay.

4.2 Average Throughput

The values of average throughput presented in **Fig. 10** and those of average per-hop delay (**Fig. 11** of Section 4.3) 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 Fig. 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.

In Fig. 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.

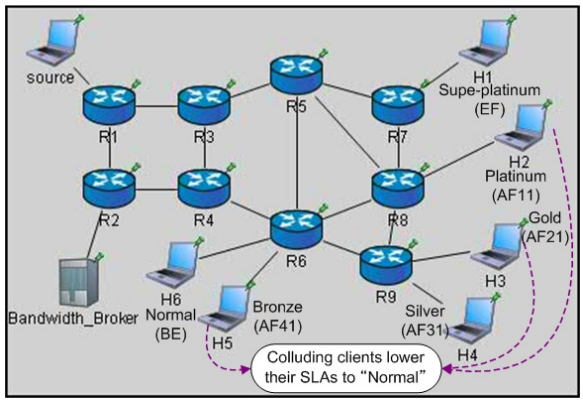


Fig. 12 Collusion attack example in a group of 6 hosts.

4.3 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 4.2 above. As illustrated by Fig. 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 11 shows that for XCAST6, delays of all 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.

4.3.1 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 **Fig. 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 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 **Fig. 13**.

As shown in Fig. 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. Fig-

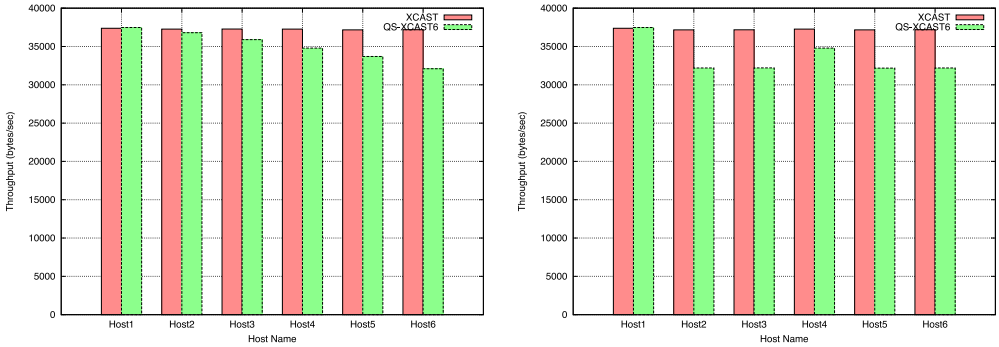


Fig. 13 Throughput values for a group of 6 hosts: a). Values with initial DSCP assignments. b). Values after H2, H3 and H5 have lowered their DSCP requirements to "Normal" (BE class).

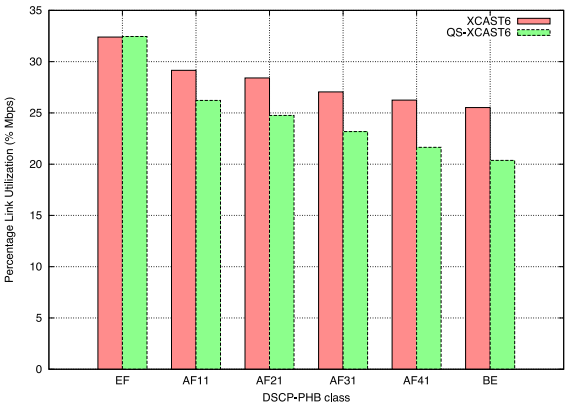


Fig. 14 Comparative average link utilization.

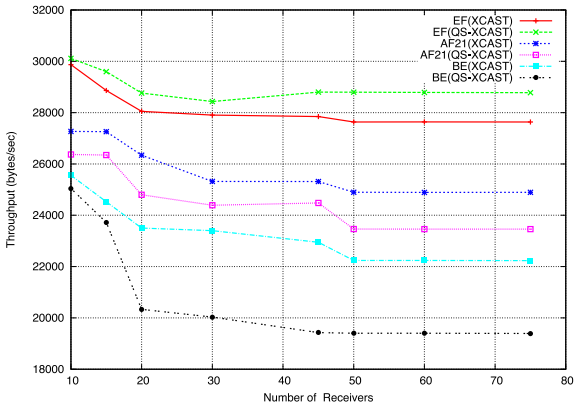


Fig. 15 Throughput for varying group sizes.

ure 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 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 *Superplatinum* services. For QS-XCAST6, Fig. 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.

4.4 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 **Fig. 14** were calculated from all group sizes (10 to 75) using the same approach explained at the beginning of Section 4.2. 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 to all receivers in only one packet. From Fig. 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.

4.5 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.

4.5.1 Effect on Throughput

For both protocols, as the group size increases, throughput decreases marginally. This is illustrated in **Fig. 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

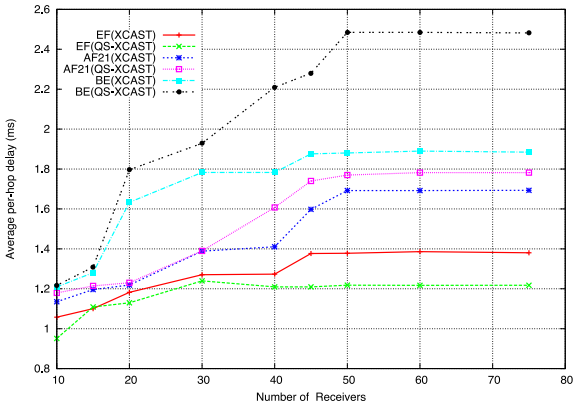


Fig. 16 Average per-hop delay for varying group sizes.

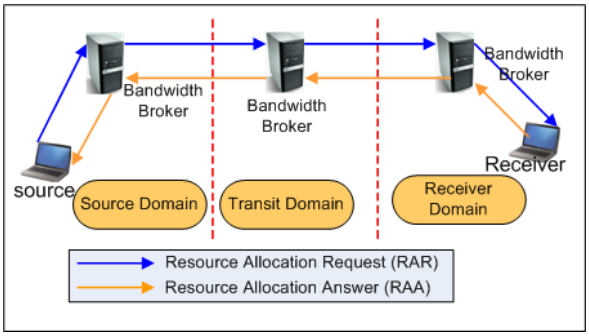


Fig. 17 QoS provisioning in multiple DiffServ domains.

corresponding treatment as defined by the Per-Hop-Behaviour hence it introduces QoS awareness to XCAST6.

4.5.2 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 **Fig. 16**.

4.6 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 **Fig. 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 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

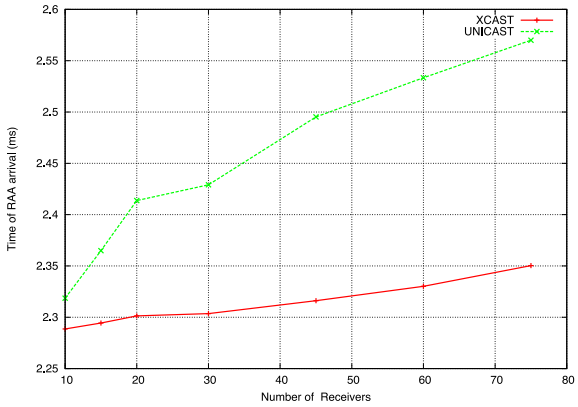


Fig. 18 Time taken to receive Resource Allocation Answer (RAA) message.

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

4.7 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 **Fig. 19**.

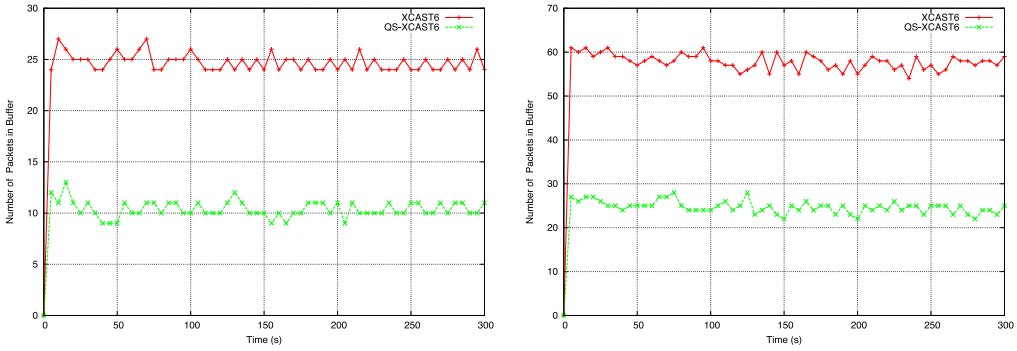
The number of buffered packets under XCAST6 remains higher than those under QS-XCAST6. For both protocols, buffering in edge routers is less than that of core routers. Interpretation of buffer evolution can be two fold in terms of: the impact on router's traffic load and that of forwarding fairness to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

4.7.1 Impact on Traffic Load

Traffic load can be defined as the ratio of incoming traffic to outgoing traffic in a router [33]. This way, buffering can be assumed to be directly proportional to a router's load size. Interpreting results of Fig. 19 in this context, we can conclude that QS-XCAST6 adds less traffic load than XCAST6 to a DiffServ router.

4.7.2 Impact on Forwarding Fairness to Other Protocols

If we interpret the buffer evolution pattern in Fig. 19 in conjunction with observations made in Figs. 10 and 11 where XCAST6 registers higher throughput and lower average per-



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

Fig. 19 The impact of XCAST6 and QS-XCAST6 on DiffServ routers.

Table 4 Comparative bandwidth allocation for XCAST6 and QS-XCAST6 using 100 MB

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

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

4.8 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 [6], 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 [34] 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 [35] 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.

4.9 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 4** 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 (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.

$$QXG = \frac{Bw\_XCAST6}{Bw\_QSXCAST6} \quad (1)$$

For the purpose of this illustration, we calculate *QS-XCAST Gain* as shown in Table 4 based on the link utilization results observed in Fig. 14 (Section 4.4).

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 bandwidth utilization of 90.68 MB saving the rest due to QS-XCAST Gain as shown in

Table 4. 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.

4.9.1 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 4.7, 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 [36] thereby ensuring that there is no forwarding penalty associated with DiffServ implementation.

5. Conclusion and Further Work

This paper proposes a model for providing Quality of Service (QoS) in XCAST using DiffServ. We explore the various challenges that complicate integration of XCAST and DiffServ. We then show how to overcome the challenges and test 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. In addition to this simulation framework, we have also implemented an experimental testbed for XCAST6 Routing Engine [14]. As further work, we shall be implementing the DiffServ architecture including Bandwidth Broker functionalities for use with the XCAST6 Routing Engine to allow for testing XCAST QoS in a real network environment.

References

[1] Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z. and Weiss, W.: An Architecture for Differentiated Services, RFC 2475 (1998).

[2] Nichols, K., Blake, S., Bakers, F. and Black, D.: Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers, RFC 2474 (1998).

[3] Davie, B., Charny, A., Bennett, J., Benson, K., Boudec, J.L., Courtney, W., Davari, S., PMC-Sierra, Firoiu, V. and Stiliadis, D.: An Expedited Forwarding PHB (Per-Hop Behavior), RFC 3246 (2002).

[4] Heinanen, J., Finland, T., Baker, F., Weiss, W. and Wroclawski, J.: Assured Forwarding PHB Group, RFC 2597 (1999).

[5] Grossman, D.: New Terminology and Clarifications for Diffserv, RFC 3260 (2002).

[6] Boivie, R., Feldman, N., Imai, Y., Livens, W. and Ooms, D.: Explicit Multicast (Xcast) Concepts and Options, RFC 5058 (2007).

[7] Imai, Y.: BSD implementations of XCAST6, *Proc. ASiaBSDCon2008 Tokyo* (2008).

[8] Abade, O.E., Kaji, K. and Kawaguchi, N.: Design, Implementation and Evaluation of a Routing Engine for a multipoint communication protocol: XCAST6, *International Journal of Computer Science and Network Security*, Vol.11, No.5, pp.200–209 (2011).

[9] Abade, O.E., Kaji, K. and Kawaguchi, N.: Quantitative Simulation of XCAST6 Performance Using OMNeT++, *Proc. Asian Internet Engineering Conference, AINTEC'11*, Bangkok, Thailand (2011).

[10] Bag-Mohammadi, M., Yazdani, N. and Samadian-Barzoki, S.: On the Efficiency of Explicit Multicast Routing Protocols, *Proc. 10th IEEE Symposium on Computers and Communications* (2005).

[11] Bag-Mohammadi, M. and Yazdani, N.: A Fast and Efficient Explicit Multicast Routing Protocol, *IEICE Trans. Communications*, Vol.E88, No.10, pp.4000–4007 (2005).

[12] Imai, Y., Kurosawa, T. and Muramoto, E.: XCAST6 (version 2.0) Protocol Specification, Internet Draft, draft-ug-xcast20-protocol-spec-00.txt (2008).

[13] Alzyoud, F.Y., Wan, T.-C. and Mohamad, I.J.: The Effect of Using XCAST Based Routing Protocol in Wireless Ad Hoc Network, *IEEE TENCON 2009* (2009).

[14] Abade, O.E., Kawaguchi, N., Imai, Y., Kurosawa, T. and Muramoto, E.: Design and Implementation of an XCAST6 Routing Engine, Internet Draft, draft-abade-xcast20-routing-engine-spec-00.txt (2009).

[15] Strigel, A. and Manimaran, G.: A Survey of QoS Multicasting Issues, *IEEE Communications Magazine*, pp.82–87 (2002).

[16] Varga, A.: Omnet++ community site, OMNeT++ (online), available from ⟨http://www.omnetpp.org⟩ (accessed 2011-07-05).

[17] Varga, A.: The OMNeT++ Discrete Event Simulation System, *Proc. European Simulation Multiconference*, pp.319–324 (2001).

[18] Deering, S., Estrin, D., Farinacci, D., Jacobson, V., Liu, C. and Wei, L.: The PIM architecture for wide-area multicast routing, *IEEE/ACM Trans. Networking*, Vol.4, No.2 (1996).

[19] Waitzman, D., Partridge, C. and Deering, S.: Distance Vector Multicast Routing Protocol, RFC 1075 (1998).

[20] Siregar, L., Budiarto, R., Omar, M. and Rosli, A.: Quality of Service Performance for Xcast in IPv6 Network, *Proc. DFmA 2008* (2008).

[21] Siregar, L., Aji, R., Hasibuan, Z. and Budiarto, R.: Quality of Service For IPTV Using Xcast in IPv6 Network, *Proc. NETAPPS 2010* (2010).

[22] Strigel, A. and Manimaran, G.: A scalable approach for DiffServ multicasting, *IEEE International Conference on Communications*, Vol.8, No.6, pp.2327–2331 (2001).

[23] Strigel, A. and Manimaran, G.: DSMCast: A Scalable Approach for DiffServ Multicasting, *Computer Networks*, Vol.44, No.6, pp.713–735 (2004).

[24] Cui, J.-H., Lao, L., Faloutsos, M. and Gerla, M.: AQoSM: Scalable QoS multicast provisioning in DiffServ networks, *Computer Networks*, Vol.50, pp.80–105 (2006).

[25] Tong, S.-R. and Chang, C.-C.: Harmonic DiffServ: Scalable support of IP multicast with QoS heterogeneity in DiffServ backbone networks, *Computer Communications*, Vol.29, pp.1780–1797 (2006).

[26] Yang, B. and Mohapatra, P.: Multicasting in Differentiated Service domains, *Proc. IEEE GLOBECOM* (2002).

[27] Li, Z. and Mohapatra, P.: QoS-aware multicasting in DiffServ domains, *Proc. Global Internet Symposium* (2002).

[28] Nichols, K., Jacobson, V. and Zhang, L.: A Two-bit Differentiated Services Architecture for the Internet, RFC 2638 (1997).

[29] Varga, A.: The INET Framework Project site, OMNeT++ (online), available from ⟨http://inet.omnetpp.org⟩ (accessed 2011-07-05).

[30] Evans, J. and Filsfils, C.: *Deploying IP and MPLS QoS for Multiservice Networks: Theory and Practice*, Morgan Kaufmann, San Francisco, CA 94111 USA (2007).

[31] Enns, R., Bjorklund, M., Schoenwaelder, J. and Bierman, A.: Network Configuration Protocol (NETCONF), RFC 6241 (2011).

[32] Shafer, P.: An Architecture for Network Management Using NETCONF and YANG, RFC 6244 (2011).

[33] Singh, K.: Router Buffer Traffic Load Calculation based on a TCP Congestion Control Algorithm, *International Journal of Computational Engineering & Management*, Vol.15, No.1, pp.20–23 (2012).

[34] Boudani, A., Guitton, A. and Cousin, B.: GXcast: Generalized Explicit Multicast Routing Protocol, *Proc. International Symposiym on*

Computers and Communications, (*IEEE ISCC 2004*) (2004).
[35] Nonnenmacher, J. and Biersack, E.W.: Scalable Feedback for Large Groups, *IEEE Trans. Networking*, Vol.7, No.3, pp.375–386 (1999).
[36] Filsfils, C. and Evans, J.: Deploying DiffServ in Backbone Networks for Tight SLA Control, *IEEE Internet Computing*, Vol.15, No.1, pp.58–65 (2005).

Odira Elisha Abade received a B.Sc. in Computer Science and a Master of Engineering in Information and Communication Engineering degrees from the University of Nairobi, Kenya and Nagoya University, Japan in 2005 and 2010 respectively. During 2005–2006, he worked with Huawei Technologies Co. Ltd in the Intelligent Networks division. He later joined the University of Nairobi in software technology services. He is currently a Ph.D. student at the Graduate school of Engineering, Nagoya University. His research interests include high availability networking, network security, mobile and wireless networks, mobile ad hoc networks and mobile IP communication and electronic money in e-commerce and m-commerce.

Katsuhiko Kaji received his B.S., M.S. and Ph.D. degrees in Information Science in 2002, 2004 and 2007 respectively from Nagoya University. He was with NTT Communication Science Laboratories, Japan, as a research associate from 2007 to 2010. From 2010, he has been an assistant professor in the Graduate School of Engineering, Nagoya University.

Nobuo Kawaguchi received his B.E, M.E. and Ph.D. in Computer Science from Nagoya University, Japan, in 1990, 1992, and 1995 respectively. From 1995 he was an associate professor in the Department of Electrical and Electronic Engineering and Information Engineering, School of Engineering, Nagoya University. Since 2009, he has been a professor in the department of Computational Science and Engineering, Graduate School of Engineering, Nagoya University. His research interests are in the areas of Human Activity Recognition, Smart Environmental System and Ubiquitous Communication Systems.