Design, Implementation and Evaluation of a Routing Engine for a multipoint communication protocol: XCAST6

Odira Elisha Abade[†, ††, †††] **Katsuhiko Kaji**[†,††] **and Nobuo Kawaguchi**[†,††],

[†]Graduate School of Engineering, Nagoya University, Japan
[††]WIDE Project, Japan
[†††]School of Computing & Informatics, University of Nairobi, Kenya

Summary
Multipoint communication has passed from research to deployment and back to a research issue. For instance, 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. XCAST (explicit multiunicast) solves this scalability problem by using unicast routes thereby eliminating multicast routes and complex distribution tree construction algorithms. When combined with mobile IPv6, XCAST can simplify node migration and its efficiency can be enhanced by using sender initiated congestion control. However the custom header structure of XCAST has also created obstacles in its deployment in the real-world. In this paper we therefore propose an "XCAST6 Routing Engine", an out-of-the-box solution that simplifies gradual deployment of XCAST in the real-world. The contribution of this paper is two fold. We not only provide a simple solution that can hasten deployment of XCAST on the real Internet but we also exemplify through experimental performance evaluation of our solution with respect to a number of performance and resource utilization metrics, that contrary to other perceptions, XCAST does not actually add an extra ordinary load to the routing resources.

Key words:
XCAST6, Multipoint communication, Routing Engine, Scalability, IPv6, Performance..

1. 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 [15]. Therefore different multicast variants exist but most of multicast applications implement multicast at the application level.

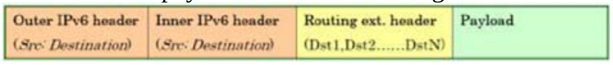
XCAST6 (explicit multiunicast on IPv6) solves multicast's group scalability problem by using unicast routes to deliver point-to-multipoint packets. It thus eliminates multicast routing tables, per-flow signaling and complex distribution tree construction algorithms. XCAST6 can also simplify migration problems in multipoint communication when combined with mobile IPv6. Its efficiency can also be enhanced using Sender Initiated Congestion Control protocol [13]. 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[7] hence its use in XCAST6 was a drawback which has since been resolved by its total elimination in XCAST6 version 2.0 [12] all but with new deployment challenges.

In this paper, we propose an out-of-the-box component; we call an "XCAST6 Routing Engine" that will simplify gradual deployment of XCAST6 in the real-world. We highlight its design and implementation then evaluate its performance based on various metrics and its utilization of CPU and Memory resources. This paper is organized into six sections. In the next section, we discuss the header structure of XCAST6 version 2.0 then define and show the need for the XCAST6 Routing Engine. The third section addresses the engine design while in section four we show how it can be implemented using FreeBSD. In section five, we show performance evaluation of the Routing Engine then related research works in section six and finally we provide our conclusion and future work in this area.

Manuscript received May 5, 2011
Manuscript revised May 20, 2011

2. XCAST6 Header structure

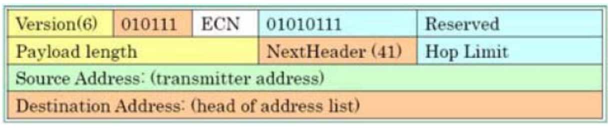
A simple XCAST6 version 2.0 packet comprises of two IPv6 headers, a routing extension header, a transport header and the payload as illustrated in figure 1.



Outer IPv6 header (Src: Destination)	Inner IPv6 header (Src: Destination)	Routing ext. header (Dst1,Dst2......DstN)	Payload

Fig. 1 XCAST6 version 2.0 header summary

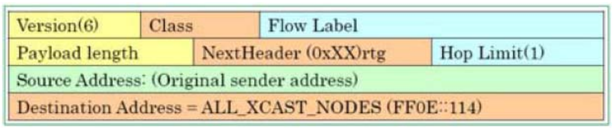
The outer IPv6 header is used to prepare a semi-permeable tunnel [16]. Semi-permeable tunneling is a trick like IP over IP tunneling that XCAST6 uses to make the XCAST6 datagram pass over non XCAST-aware nodes. The traffic class of the outer IPv6 header is “010111XX”. The first four bits of the traffic 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 and RFC4727 [5]. The remaining two bits, “XX” are Explicit Congestion Notification (ECN) bits as specified in RFC3168[6]. 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 2 shows a detailed view of the outer IPv6 header.



Version(6)	010111	ECN	01010111	Reserved
Payload length			NextHeader (41)	Hop Limit
Source Address: (transmitter address)				
Destination Address: (head of address list)				

Fig. 2 Outer IPv6 header

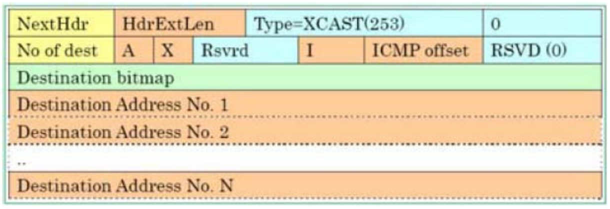
The inner IPv6 header shown in figure 3 is processed by the node or the router specified by the destination address of the semi-permeable header. Its source address is set to the unicast address of the original XCAST sender and its destination address set to ALL_XCAST_NODES. If a node is XCAST-aware, 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.



Version(6)	Class	Flow Label
Payload length	NextHeader (0xXX)rtg	Hop Limit(1)
Source Address: (Original sender address)		
Destination Address = ALL_XCAST_NODES (FF0E::114)		

Fig. 3 Inner IPv6 header

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 values defined in RFC4727 [5]. 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.



NextHdr	HdrExtLen		Type=XCAST(253)			0
No of dest	A	X	Rsvrd	I	ICMP offset	RSVD (0)
Destination bitmap						
Destination Address No. 1						
Destination Address No. 2						
..						
Destination Address No. N						

Fig. 4 XCAST6 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.

2.1 XCAST packet processing in the 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 packets so that there is only one copy of the packet for each of the next hops.
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 packets to each of these next hops.
vi. If there is only one destination left for a given next hop, the router can optimize delivery by sending the XCAST packet as an ordinary unicast packet.

3. The XCAST6 Routing Engine Design

XCAST6 Routing Engine design has been prompted by the obstacles faced while deploying XCAST6 in the real-world.

3.1 XCAST6 Deployment challenge

Experiments and small scale video conferencing have been used to prove the advantages of XCAST6 especially in terms of group scalability [14]. 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, there is a need to consider simpler, cost effective method that can be used to realize the deployment of XCAST6 in the real-world as we escalate research on how it will be incorporated into the future routers. 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 routers.

3.2 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. It is connected side-by-side to the core router as shown in the figure 5 and acts as a "software-router" for XCAST6 packets. As in figure 5, 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 step 4 above.

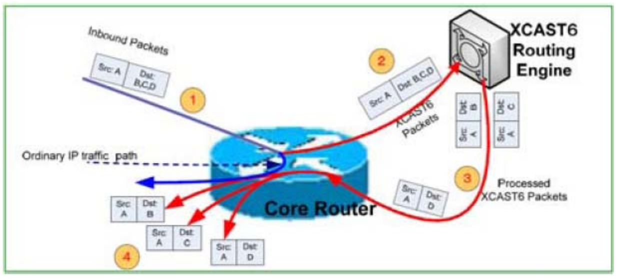


Fig. 5 XCAST6 Routing Engine Concept

3.3 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.
ii. How to process the XCAST6 packets in the routing
iii. engine and still realize the same next hops as if processing was done in the core router.
iv. How to forward XCAST6 packets correctly from the XCAST6 Routing Engine.

3.4 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;
            }
        }
   }
}
```

Listing 1. 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.

```
routing−instances {
  FBF−nxt−hdr {
    instance−type forwarding;
      routing−options {
        rib FBF−nxt−hdr.inet6.0 {
         static {
          route ::/0 next−hop 2001:200:30::1;
         }
        }
      }
    }
}
```

Listing 2. Routing instance for XCAST6 packets

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 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.

3.5 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.

3.5.1 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 [9]. 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.

3.5.2 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 [8]. 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. On 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 zeroes 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.

3.6 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 6. For example, the core router in our testbed has 4 Gigabit interfaces as shown below.

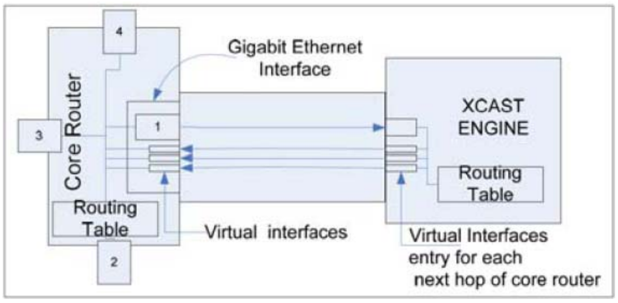

4
Gigabit Ethernet Interface
Core Router
3
1
XCAST ENGINE
Routing Table
Routing Table
Virtual interfaces
Virtual Interfaces entry for each next hop of core router
2

Fig. 6Virtual 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.

4. 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 [17] network using the WIDE connection at Nagoya University so as to ensure it operates in a real Internet setting.

5. Performance Evaluation

Our aim was to evaluate the routing engine in a real Internet environment. We therefore deployed the routing engine in the WIDE [17] network at Nagoya University and used a DV format based videoconferencing application to evaluate its packet processing capability. Digital Video (DV) format [DV ref] is a packet based video/audio format with each format specifying a standard digital interface media to exchange the digital stream 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), a middleware for multipoint communication we have developed in our laboratory [18].

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).

5.1 Bandwidth Utilization

Full DV stream consumes over 30Mbps when using standard NTSC quality video at 525 lines and 29.97 picture frames per second. With four 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 processing 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. The table below shows the varying results at each transmission rate.

Table 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

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.

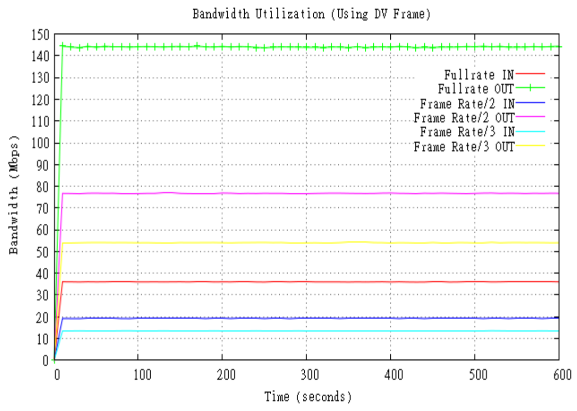


Fig. 7 Inbound and Outbound Bandwidth for Full frame rate to Frame rate/3

The graphs in figure 7 and figure 8 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 7 and figure 8 below.

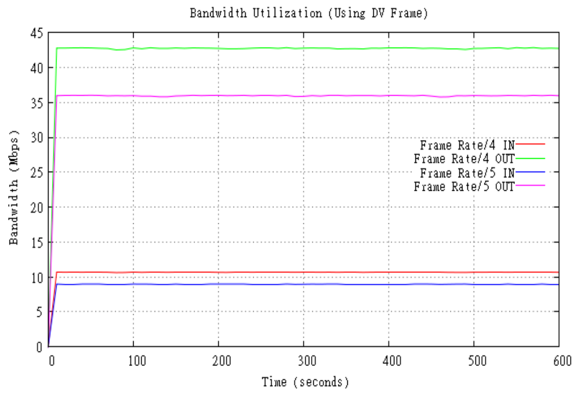


Fig. 8 Inbound and Outbound Bandwidth for Frame rate/4 and Frame rate/5

5.2 Latency and Latency Distribution

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.

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 9 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. 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 bandwidth parameter. Then the entire data set is summed and normalized by the number of measurements included. Figure 10 shows this estimation. We note that over 50% of the latency values observed are slightly above 110 microseconds as shown below.

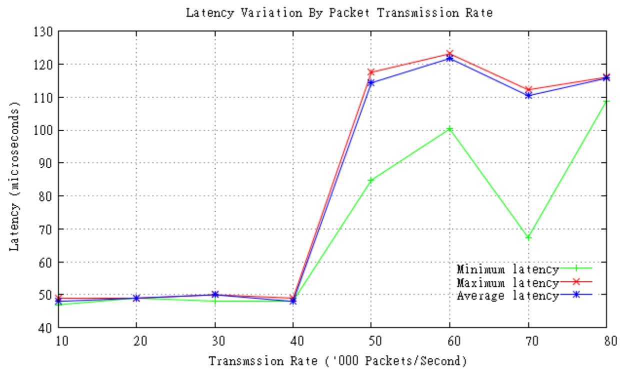


Fig. 9. Latency variation by packet transmission rate

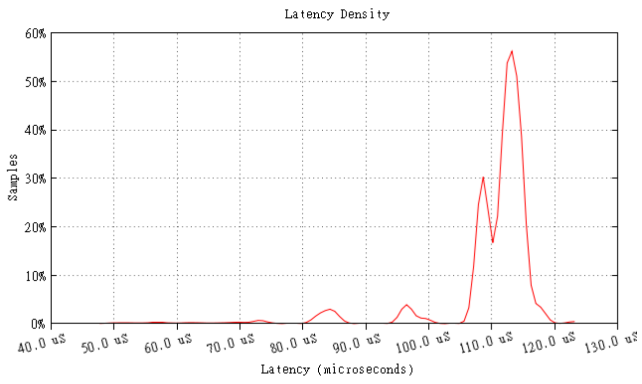


Fig. 10 Kernel Density Estimate of the latency distribution

5.3 Packet loss

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.

5.4 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.

5.4.1 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 [19], 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 ICMP6 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 (swi net) in FreeBSD. The results are presented as shown in figure 11 below.

We observe that processing of both XCAST6 and ICMP6 require almost equivalent counts of context switches due to CPU bus I/O activities. 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 ICMP6 registers fluctuations that on the lower bound are almost equivalent to 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.

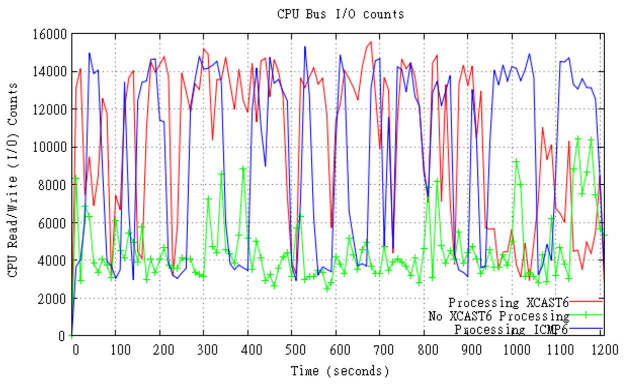


Fig. 11. Comparative count of Context Switches due to I/O requests on CPU bus

5.4.2 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 [20] 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 12. The counts of "system" memory (not userland memory) transactions for XCAST6 and ICMP6 processing and also when no active packet processing is done show clearly that XCAST6 processing registers higher counts while ICMP6 maps nearly equally to the system idle state. This is possibly attributed to the XCAST6 packet length which is certainly longer than that of ICMP6.

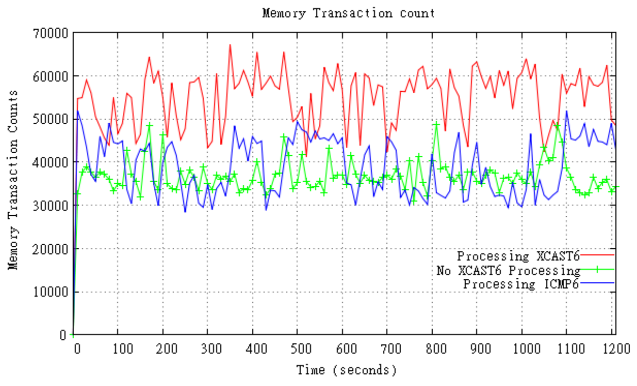


Fig. 12 Comparative count of Context Switches due to memory transaction requests

5.4.3 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 with several XCAST destinations in the header as shown

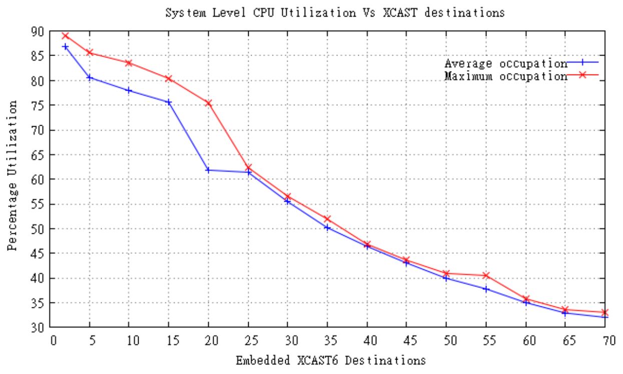


Fig. 13. XCAST6 CPU Utilization

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.

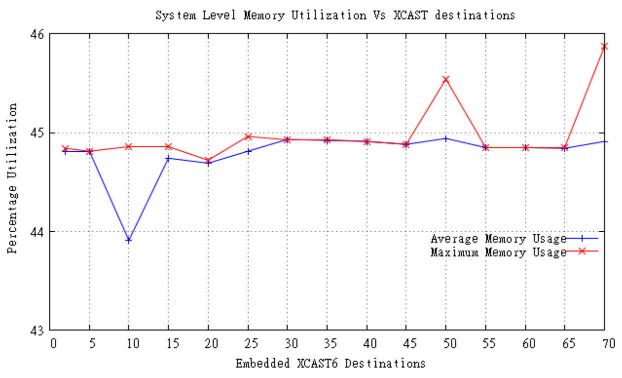


Fig. 14 XCAST6 Memory Utilization

5.4.4 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[13].

6. Related Works

The SAMTK project[1] [18] 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[3] 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[2], 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.

7. Conclusion and Future work

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.

Performance evaluation of the XCAST6 Routing Engine has shown favorable results. XCAST6 Routing Engine has also been deployed in the WIDE network at Nagoya University and we are seeking to investigate O0S implementation on XCAST6 protocol and avail the information for other academic research.

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.

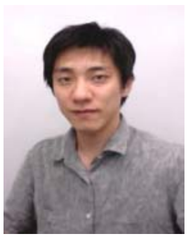
Acknowledgments

We thank all members of the XCAST fans club for their contribution in the XCAST6 research. We also thank the WIDE project, Japan, that has offered venues for consultation and sharing of ideas and experiment sessions as we work on XCAST6 Routing Engine. Finally we thank the SAMTK research team and many other people who have given us feedback on many aspects of the XCASt6 Routing Engine.

References

[1] Kawaguchi N., Nishiura S., Abade E.O, Kurosawa T.,
[2] Jinmei T., Muramoto E., "NAT Free Open Source 3D
[3] Video Conferencing using SAMTK and Application Layer Router", IEEE, CCNC January 2009.
[4] Xuemin Shen, Heather Yu, John Buford, Mursalin Akon,"Multicast Routing in Structured Overlays and Hybrid Networks",In: Handbook of Peer-to-Peer Networking, Berlin Heidelberg:Springer Verlag, December 2009.
[5] MatthiasW., Thomas C. S., GeorgW.,"OASIS: An Overlay Abstraction for Re-Architecting Large Scale Internet Group Services", Lecture Notes in Computer Science, Vol. 5630, pp. 95–106, Berlin Heidelberg:Springer-Verlag, June 2009.
[6] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.
[7] B. Fenner, "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.
[8] K. Ramakrishnan et al, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.
[9] The case against Hop-by-Hop options, Internet Draft, draftkrishnan-ipv6-hopbyhop-02.txt, February 2008, Work in Progress.
[10] R. Enns, Ed, NETCONF Configuration Protocol, RFC4741, Juniper Networks Inc, December 2006.
[11] D. Harrington, R. Presuhn, B. Wijnen, An Architecture for Describing SNMP Management Frameworks, Cabletron Systems Inc, April 1999.
[12] JUNOS Feature guide, Release 9.1, Juniper Networks Inc.
[13] R. Boivie, et al., "Explicit Multicast (Xcast) Concepts and Options", RFC 5058, November 2007.
[14] XCAST6 (version 2.0) Protocol Specification, Internet Draft, draft-ug-xcast20-protocol-spec-00.txt, February 2008, Work in Progress.
[15] Yoneda T., Muramoto E., Chih-Chang H., Konishi K, MatsumotoT, "Evaluation of Congestion Control Method using Multiple-Constant Bit Rate Streams over XCAST6"
[16] Y. Imai et al, BSD implementations of XCAST6, in proceedings of ASiaBSDCon2008, March 2008.
[17] Lorenzo Aguilar, Datagram Routing For Internet Multicasting, IGCOMM '84, March 1984.
[18] A. Conta, S. Deering, Generic Packet Tunneling in IPv6 Specification,RFC2473, December 1998.
[19] Widely Integrated Distributed Environment (WIDE) project (www.wide.ad.jp)
[20] Scalable Adaptive Multicast Toolkit (SAMTK) project (www.samtk.org)
[21] The FreeBSD Project PMC tools (http://wiki.freebsd.org/PmcTools)
[22] Hyok Kim et al, Peformance Analysis of the TCP/IP Protocol Under Unix Operating System for High Performance Computing and Communications, Dec 2002.

Odira Elisha Abade received Bsc in Computer Science and 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 Intelligent Networks division. He later joined the University of Nairobi in Software technology services. He is currently a PhD 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 the 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 research associate from 2007 to 2010. From 2010, he has been an assistant professor in Graduate School of Engineering, Nagoya University