Quantitative Simulation of XCAST6 Performance Using OMNeT++

Odira Elisha Abade
Graduate School of Engineering, Nagoya University, Japan
abade@ucl.nuee.nagoya-u.ac.jp

Katsuhiko Kaji
Graduate School of Engineering, Nagoya University, Japan
kaji@nuee.nagoya-u.ac.jp

Nobuo Kawaguchi
Graduate School of Engineering, Nagoya University, Japan
kawaguti@nagoya-u. jp

ABSTRACT
Explicit multiunicast (XCAST) is a form of small group multicast in which the sender embeds IP addresses of all receiving nodes in a special header and sends it with all data packets. On sending, each XCAST-aware router in the path examines the packet and partitions the destinations into various sets based on their next hops. For each set, the router uses a bitmap in the packet header to ensure that each copy of the packet is delivered only to destinations whose corresponding bits in the bitmap are set to 1. Sometimes real world implementation of XCAST6 can take a long time to deploy and resource constraints can limit the scale of testing that can be done. We have therefore integrated XCAST6 into the IPv6 stack of INET Framework for OMNeT++ to allow for large-scale, exhaustive and realistic investigation of XCAST6 characteristics and to complement our current research into real world deployment of XCAST6. The contribution of this paper is two fold. First, it describes our implementation of XCAST6 in OMNeT++'s INET framework. Secondly, it experimentally shows the effectiveness of XCAST6 with regards to various multicast performance metrics such as stress, end-to-end delay, efficiency and packet processing overhead rate.

Categories and Subject Descriptors
C.2.2 [**Computer-Communication Networks**]: Network Protocols.

C.4 [**Performance of Systems**]: Performance attributes.

General Terms
Performance, Quantitative.

Keywords
XCAST6, Simulation, INET framework, OMNeT++.

1. INTRODUCTION
Numerous multicast technologies have been proposed to ensure efficient utilization of Internet bandwidth for group communication. Even though IP Multicast is conceptually able to provide efficient group communication and utilize the available bandwidth efficiently[6], its global deployment has not been effectively realized. This has majorly been due to security issues, efficient congestion control scheme and need for special support in the network devices[11]. However, there exist local networks that are multicast capable[16] and are used to deliver TV and Telephony services. Multicast was originally targeted at very large groups but there also exist group communication applications such as video conferencing, IP telephony and online multiplayer games which only require a large number of distinct, small groups and cannot be served well with the traditional IP multicast. Small group, multi-destination multicast variants have therefore been proposed specifically to serve this class of applications.

Explicit multiunicast (XCAST)[4] is a form of small group multicast in which the sender embeds IP addresses of all receiving nodes in a special header which it then sends with every data packet. During transmission, each XCAST-aware router in the path examines the packet, looks up the next hop routers for each of the embedded destination addresses and partitions the destinations into various sets based on their next hops. For each set, the router uses a bitmap in the packet header to ensure that each copy of the packet is delivered only to destinations whose corresponding bits in the bitmap are set to 1.

Deployment of XCAST in the real world has been a subject of research[12,3]. In [2,3] we have proposed a simple routing engine for realizing this but we are also cognizant of the fact that sometimes real world implementations can take a very long time to realize and resource constraints can limit the scale of testing that can be done. A simulation environment for XCAST6 is therefore necessary. However most of the existing simulators do not have multicast routing models required by small group multicast protocols such as XCAST.

Kolberg and Burford[14], in their project of extending OMNeT++ for Scalable Adaptive Multicast simulations, have come up with XCAST and AMT implementations for the Oversim simulator[1]. Their work however currently supports only simulation of XCAST on the IPv4 stack. On the other hand, our current research on XCAST6 deployment in the real world is based on XCAST6 version 2.0[13] hence we found it necessary to extend the IPv6

4INTEC'11, November 9–11, 2011, Bangkok, Thailand

stack of the INET Framework[17] in OMNeT++[18,19] to allow for simulation of our current research.

In the next section we briefly describe XCAST6, OMNeT++ and the INET Framework. In section three we describe the implementation of XCAST6 on OMNeT++ while in section four we describe our simulation of XCAST6 using OMNeT++ and discuss the results in section five. In the last section we give our conclusion and future work on XCAST6.

2. XCAST6 OVERVIEW

Conventional IP multicast routing protocols[5, 7] require maintenance of state-information relating to groups in the on-tree routers, usually referred to as Multicast Forwarding Tables (MFT) so as to forward multicast packets correctly. This approach is usually not advisable due to the possibility of the tree growing and in such cases, the state maintenance adds unnecessary load to the network devices such routers. Small group multicast variants solve this problem together with the scalability issue of the conventional IP multicast. XCAST6 refers to the implementation of explicit multiunicast on IPv6[4,13]. In contrast to the conventional IP Multicast and other multicast variants, in XCAST6, the sender explicitly specifies the destination addresses of all the receivers as a list of unicast addresses embedded in the IPv6 packet header. Succinctly, the sender embeds a list of destinations in the routing extension header of the IPv6 packet, puts the nearest destination in the destination address field of the IPv6 packet 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 and when there is only one destination left in the list, the XCAST6 packet can be transmitted as an ordinary unicast packet.

Together with list of destinations, the XCAST6 header also comprises of a bitmap that the routers use to determine which of the embedded destinations the packet needs to be delivered and to which ones the packet has already been delivered[13]. The bits in the bitmap correspond to each of the embedded destinations. 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 figure 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.

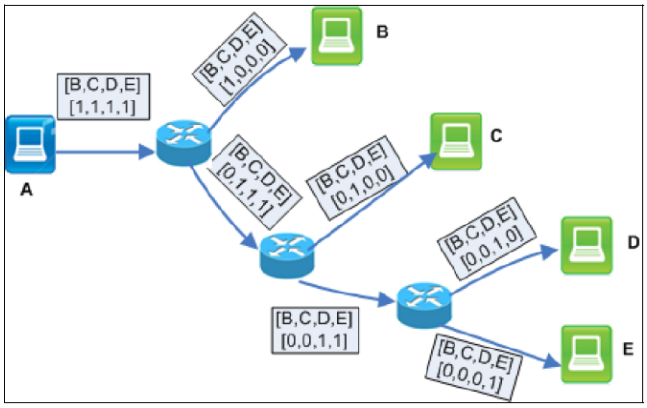

Figure 1. XCAST6 Overview

2.1 OMNET++

Object Modular Network Testbed in C++ (OMNeT++)[18,19] is an open source, extensible, modular, component-based simulation library and framework. Because of its generic architecture, OMNeT++ is used in simulating a wide range of networks. Domain-specific functionality such as support for sensor networks, wireless ad-hoc networks, Internet protocols, performance modeling, photonic networks, etc., is provided by model frameworks, developed as independent projects such as the INET Framework, INETMANET, xMIPV6 and MiXiM among others. OMNeT++ is a collection of hierarchically nested modules which communicate with each other using message passing and messages may carry arbitrary data structures. The depth of module nesting is unlimited. Modules can be connected with each other via gates (other systems would call them ports) and channels. Channels can posses certain bandwidth, delay and loss characteristics. The modules can also be combined to form *compound modules*. Modules at the lowest level of the hierarchy are called *simple modules* and they encapsulate the model behavior. Simple modules are programmed in C++ and make use of the simulation library. In the hierarchy, the top-most module is called the *System Module* or *Network* and contains one or more sub-modules each of which could contain other sub-modules.

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

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

2.2 The INET Framework

The INET Framework[17] builds upon OMNeT++ and uses the same concept of modules and messages whereby modules communicate with each other by message passing. It is organized into protocol layers that nearly mirror the OSI reference model. Specifically, the INET Framework contains models for several wired and wireless networking protocols in the Link, Network and Transport layers of the protocol stack. Support for mobility and wireless communication has been derived from the Mobility Framework project[9]. Some of the implemented protocols include: UDP, TCP, SCTP, IP, IPv6, Ethernet, PPP, 802.11, MPLS, OSPF, and many others. It implements the MPLS model with RSVP-TE and LDP signaling.

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

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

Modules in the INET Framework can be freely combined to form hosts and other network devices with the NED language without C++ code or the need for recompilation whatsoever. Therefore, there are some common modules that appear in all (or many) host, router and device models such as *InterfaceTable*, *RoutingTable* and *NotificationBoard* modules.

Modules communicate by message passing and so are the protocol layers. When an upper-layer protocol wants to send a data packet over a lower-layer protocol, the upper-layer module sends the message object representing the packet to the lower-layer module. The lower layer protocol module then *encapsulates* the message and sends it out. When a lower layer protocol wants to send a packet to an upper layer protocol, it removes lower layer information from the packet *(decapsulation)* then sends the message to an upper protocol layer module.

Extra information such as connection identifiers, destination addresses, and parameters like the packet TTL are often needed to be transmitted together with the packets between the protocol layers. This extra information is attached to the message object as *Controlinfo*. *Controlinfo* are small value objects, which are attached to packets (message objects) with its *setControlInfo()* member function. *Controlinfo* only holds auxiliary information for the next protocol layer, and is not supposed to be sent over the network to other hosts and routers.

3. IMPLEMENTING XCAST6 IN OMNeT++

XCAST and AMT are some of the Scalable Adaptive Multicast protocols currently not supported by most of the existing network simulators, especially when analyzing hybrid multicast protocols[14]. In [14], Kolberg et al have proposed an implementation of XCAST for Oversim and OMNeT++ which in the current state, is only on the IPv4 protocol stack of INET Framework. Additionally, their implementation is an XCAST model whereby an XCAST packet gets duplicated at branching nodes and fewer addresses are attached to each copy of the packet until each copy reverts back to a unicast IP/UDP message as the packet traverses the network. This complex header reconstruction at every branch can be reduced to a simple operation of updating a bitmap[4, 13] in the XCAST6 header and leaving the destinations intact in all copies of the XCAST6 packets. Furthermore, the use of bitmaps instead of the header reconstruction has the advantage that it allows for the possibility of using XCAST with IPSec since only the bitmap is changed and not the entire IPv6 header.

Since our main research on XCAST is in the area of real world deployment of XCAST6 version 2.0[13], our motivation is to realize an XCAST6 simulation environment which can complement our real world XCAST6 deployment when faced with time and resource constraints. Therefore, our target in OMNeT++ was the implementation of XCAST simulation on the IPv6 stack. In our work, in addition to targeting the IPv6 stack of OMNeT++'s INET Framework, we also implement the XCAST6 concept where the bitmaps are used instead of the complex header reconstruction[4, 13]process. Our approach also utilizes much of the functionalities in the INET Framework's network layer modules with significant

changes done only in the network layer modules. Since the key nodes such as *Router6* and *StandardHost6* modules inherit from most of the network layer modules like *IPv6* and *IPv6ControlInfo* modules, our implementation does not require any adaptation whatsoever to the existing *StandardHost6* and *Router6* modules. Therefore sample networks can be generated using any existing tools such as OMNeT++ IDE or ReaSE[10] without any other need for adaptation. The subsequent subsections describe our implementation in each of the layers of INET Framework.

3.1 Network Layer

The greatest impact of XCAST6 is in the network layer of the INET Framework. We derived a number of classes to implement XCAST6 functionality while some other classes were only slightly altered by adding a few lines for XCAST6 specific functionality. As specified in [4, 13], XCAST6 destinations and the bitmap are contained in the IPv6 routing extension headers. We modified the *IPv6ExtensionHeader.msg* file to include these XCAST6 specific attributes in the INET Framework's IPv6 routing extension header. This in effect adds these two data containers into the generated *IPv6ExtensionHeader* class. To facilitate inter-protocol layer transmission of XCAST6 controls, we added the XCAST6 traffic class, destination addresses list and the bitmap in the *IPv6ControlInfo* Module. Traffic class of XCAST6 header is *"010111XX"*, which is *"23"* in decimal notation, so any IPv6 traffic in the simulation with traffic class of 23 (*"010111XX"*) is recognized as XCAST6 traffic and this information is shared between protocol layers using the *IPv6ControlInfo* module. The *IPv6Datagram* module was also amended to be able to handle the new XCAST6-capable *extension header* modules.

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

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

3.2 Transport Layer

In INET, the source and destination addresses of a UDP datagram are defined in the *UDPControlInfo* module. We therefore modified this module to be able to handle multiple destinations as required by XCAST6. In [4, 13], the XCAST6 header is defined such that the destination address of the outer IPv6 header is that of *ALL_XCAST_NODES*, given in the range of IPv6 multicast addresses as *"ff0e::114"*. The destination address used by the *UDPControlInfo* module in our implementation also uses the *ALL_XCAST_NODES* while the list of unicast destinations and the bitmap are contained in our newly defined containers in this module.

There is no further modifications and derivations needed in the implementation of UDP protocol since we use *UDPControlInfo* and *IPv6ControlInfo* to exchange the XCAST6 information between protocol layers in the stack and just before transmitting the message across the network, we add the XCAST6 information into the *IPv6Datagram* message.

3.3 Application Layer

We developed a simple UDP application which can be used to test XCAST6 in OMNeT++. It is based on the *UDPBasicApp* that comes with INET Framework. However we implemented in it the ability to specify a parameter with the list of IP addresses to be used as XCAST6 receivers from the omnet.ini file. In simulation models with several nodes and only a set of them need to form an XCAST6 group, it is possible to specify a number of groups and the IP addresses of the members of the group. The application then delivers data to all members of the specified group. In the INET's *UDPBasicApp* example the application randomly picks one address and sends data to it. We improved on this such that when simulating on XCAST6, it is possible to specify a list of groups, of course each group also has a list of its members. Our application can randomly select a group and deliver a UDP packet to all members of that group.

3.4 Statistics Collection

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

4. SIMULATIONS

We used OMNeT++ IDE to define a model network with a topology similar to the one shown in figure 2 comprising of hosts, routers and switches. The model network had seven subnetworks with six routers connected to each subnetwork

by an Ethernet switch. In each subnet, we defined ten IPv6 hosts connected to the Ethernet switches by bidirectional links. We considered a simple case where there is only one XCAST6 sender, the host connected to router R1, while all the remaining 70 hosts were receivers. All the links were assumed to have the same bandwidth and equal chances of packet loss. The sender's message frequency was assigned to 0.9s in the omnet.ini file.

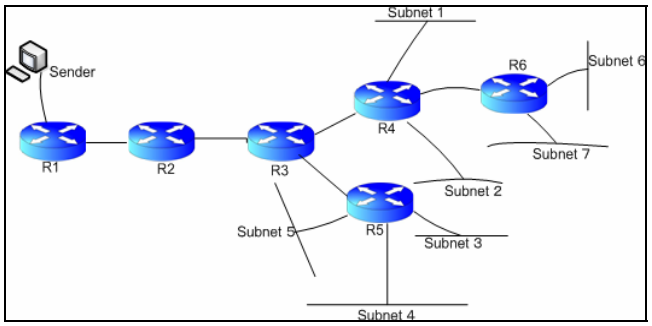

Figure 2. Testbed

4.1 Simulation scenario

We are interested in investigating XCAST behavior at different points in the network, namely the *sender side, the core and the edge of the network*. We therefore investigate the number of XCAST6 packets circulating in the network at each of these 3 distinct points in different simulation scenarios. We defined test scenarios by changing the number of receivers in each case. This changes the length of XCAST6 header and the number of locally attached hosts for the routers each time. We then monitored the impact of the number of receivers on XCAST6 on each of the routers at various points in the network. Table 1 summarizes each of these test scenarios.

5. PERORMANCE EVALUATION

Several performance metrics have been defined to characterize the multicast communication service and its impact on the network[8, 15].

Table 1. Summary of the simulation scenarios

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

The most important ones being stress (both link and node stress), link stretch, time to first packet, control overheads and efficiency. We measured some of these metrics for XCAST6 as discussed in the subsections that follow.

5.1 Stress (Node Stress)

This is defined in terms of the number of identical packets a physical link (link stress) or a node (node stress) carries in a network. Clearly for XCAST6, the link stress is 1 because no packet is sent repeatedly over the same link. However we measured the node stress since at each branching point, the XCAST algorithm states that the router duplicates the packet according to the number of the next hops. For our model network routers R1 and R2 have no branches hence the stress is minimal and incurred only in destination lookups for the embedded XCAST6 destinations.

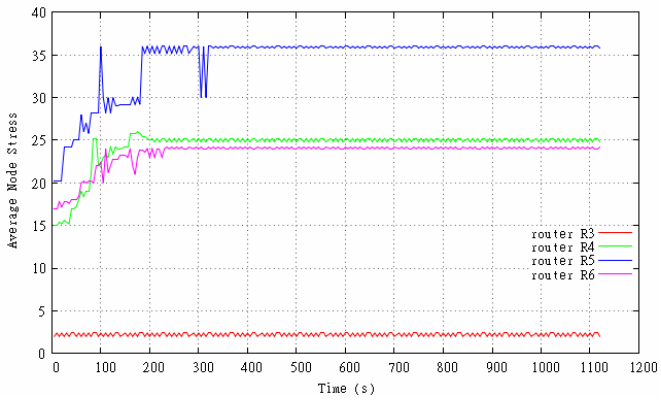

Figure 3. Average stress per router for a group of 70 nodes

To calculate the average stress at the branching routers, we define a number of variables. We define K to be the number of packets sent by the sender, L as the number of locally attached receivers of the packet, N the number of unique next hops for each of the destinations in the XCAST6 header and T the time interval for each measurement split into (i) time slots. For each of the routing node (h), we used the variables K, L, N and T to formulate the simple equation (1) which we used in calculating the average stress (λ).

$$\lambda_h = \sum_{i=0}^{T} K(L_h(i) + N_h(i)) \qquad (1)$$

In all scenarios, we observe that it takes sometime before the stress level stabilizes. We observed that a lot of Neighbor Discovery and Router Advertisement Packets are exchanged in the network over the same period of time so we attribute the shape of graph at those points to these kind of exchange messages. Using equation (1) we calculated the average node stress at intervals of 50 seconds based on emitted XCAST6 statistical signals. Figure 3 shows the results for the first scenario with 70 receivers.

Router R3 registers a constant low average stress of approximately 2 which corresponds to the expected unique next hops while router R5 registers the highest stress level averaging to approximately 36. This is because of the several receivers that are locally attached to it. Routers R4 and R6 with the same number of locally attached receivers register nearly similar level of stress but that of router R4 is slightly higher because router R4 also has router R6 as the next hop for all XCAST6 packets destined to subnetworks 6 and 7.

We reduced the receivers to 60 but conducted two tests by changing the number of locally attached receivers on routers R4 and R6. Figure 4 shows the results when the number of receivers attached to router R4 were reduced to 10 *(5 in each subnet for subnets 1 and 2)* while retaining locally attached receivers on router R6 at 20, *(10 in each subnet for subnets 6 and 7)*.

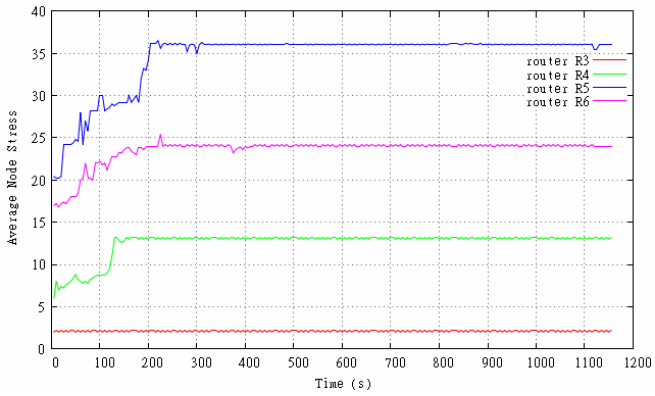

Figure 4. Average stress per router for a group of 60 nodes (case one)

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

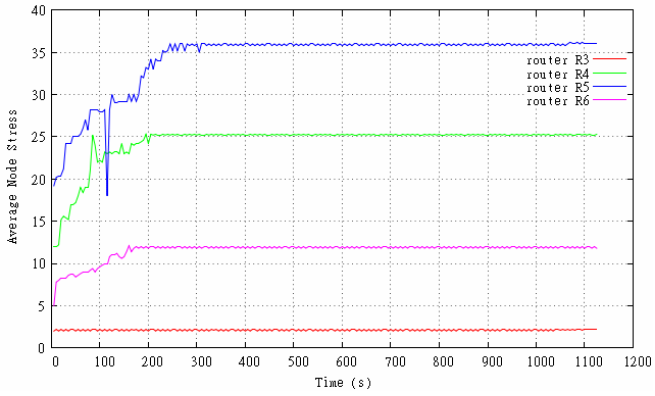

Figure 5. Average stress per router for a group of 60 nodes (case two)

Figure 5 shows that the impact of changing locally attached hosts is noted on routers R6 and R4 while the rest of the routers in the model are unaffected despite the fact that the total number of receivers has been reduced to 60.

The fourth scenario had a group size of 50 receivers only and host distributed as summarized in table 1. A considerable reduction in stress on router R5 is seen as indicated in Figure 6.

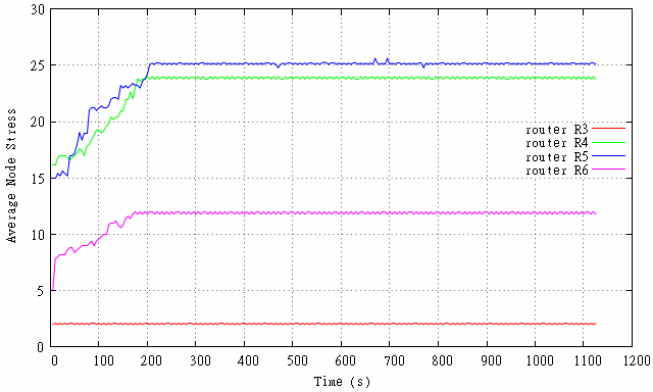

Figure 6. Average stress per router for a group size of 50 nodes

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

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

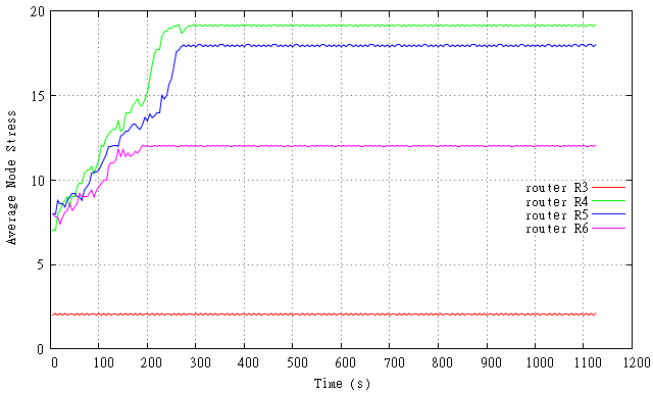

Figure 7. Average stress per router for a group size of 40 nodes

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

5.2 End-to-End delay

We also measured the average end-to-end delay during each of the above scenarios. Using the NICE overlay protocol implemented in OverSim[1], we used the same network model to compare the time delays in XCAST6 and Application Layer Multicast (ALM) and results are shown in figure 8. This also acts as an indicator of the processing overhead incurred by XCAST6 due to the increased packet header complexity as compared to ALM. However, XCAST6 is targeting small groups and the difference in end-to-end delay noted is also very small considering it is in the range of very minor fractions of a second.

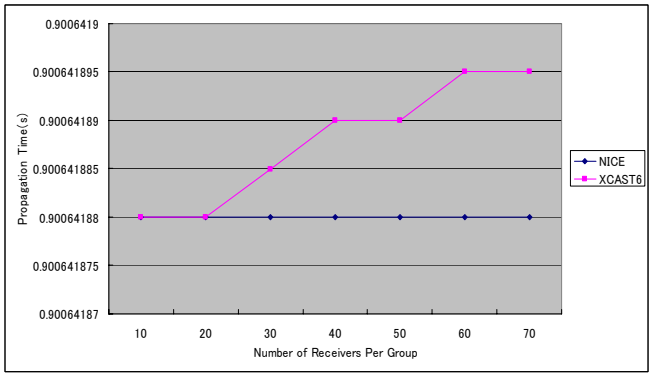

Figure 8. End-to-End Delay

5.3 Cost overhead rate

Considering that the result of section 5.2 shows a likely impact of the XCAST6 header complexity as a processing overhead, we compare XCAT6 and multicast based on a factor we define as the cost overhead rate. This is calculated using the number of packets that XCAST6 can generate if used in networks of hosts with varying MTUs such as the Internet. We define the following variables for this comparison: HDR to be the size of IPv6 header, ADR to be the size of an IPv6 address, N to the number of XCAST6 receivers. THDR to be the length of the transport protocol header and P to be the size of the payload data to be delivered in an XCAST6 message. We assume that each bit of the bitmap requires only 1 unit of data to store.

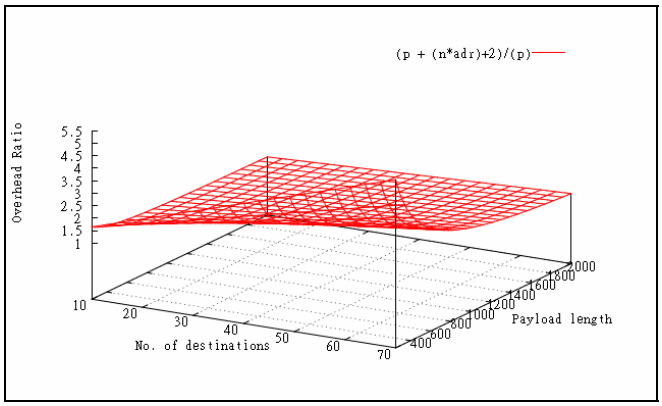

Figure 9. XCAST6/Multicast Cost overhead ratio

The number of XCAST6 packets generated to transmit a data of payload length P over an link with a given MTU can be calculated as:

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

Where 2 is added to represent the bytes for storing the size of the bitmap and also the length of the header in the IPv6 routing extension header. For the same data, the number of packets required by Multicast will be:

$$f_m(N,P) = \frac{HDR + THDR + P}{MTU} \quad (3)$$

The reduced ratio of equation (2) and (3) can be defined as the XCAST6 gain on Multicast and it comes to:

$$\frac{f_x(N,P)}{f_m(N,P)} = \frac{P + N(ADR + 1) + 2}{P} \quad (4)$$

We note that for IPv6, the ADR will be 16 bytes and when we plot equation 4 above for upto 70 destinations against the payload size of between 250 bytes and 2000 bytes we find the resulting comparison as shown in figure 9. The value of the cost overhead rate goes as high 5.5 when there are many destinations and the payload length is shorter but for a majority of cases the rate is between 2 and 2.5 especially for fewer destinations and payload length of between 1000 bytes and 1400 bytes. Considering that the MTU for common Network Interface cards is at 1500 bytes, it implies that XCAST6 bears a low overhead for the typical communication scenarios.

6. CONCLUSION AND FUTURE WORK

This paper presents a work in progress in implementing XCAST6 on OMNeT++ simulation environment. We have shown how to implement XCAST6 on OMNeT++ and we chose an approach that would ensure that all existing host modules of OMNeT++ can work without any need for alteration. We have also implemented various signaling information for statistical data collection in the model and used these to measure the stress node due to XCAST packet replication. We have confirmed that the simulation works correctly. As a further work we intend to implement XCAST6 QoS using this simulation tool and compare various multicast QoS policies against XCAST6. We shall also implement additional performance metrics in the model. We intend to use the XCAST6 simulation to complement our current research on real world deployment especially in projects where time and resource constraints can affect delivery when trying to test an XCAST6 characteristic on a large scale in a real network.

7. REFERENCES

[1] The overlay simulator http://www.oversim.org/wiki.

[2] O. E. Abade, K. Kaji, and N. Kawaguchi. Design, implementation and evaluation of a routing engine for a multipoint communication protocol: Xcast6. International Journal of Computer Science and Network Security, 11(5):200–209, May 2011.

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

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

[5] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C. Liu, and L. Wei. The pim architecture for wide-area multicast routing. IEEE/ACM Transaction on Networking, 4(2), April 1996.

[6] C. Diot, B. N. Leivine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the ip multicast service and architecture. IEEE Network, 14(1):78–88, January 2000.

[7] D.Waitzman, C.Partridge, and S.Deering. Distance vector multicast routing protocol. RFC1075, November 1998.

[8] A. El-Sayed, V. Roca, and L. Mathy. A survey of proposals for an alternative group communication service. IEEE Networks, 17(1):46–51, 2003.

[9] Mobility Framework. Mobility framework, http://mobility-fw.sourceforge.net

[10] Gamer and M. Scharf. Realistic simulation environments for ip-based networks. Proceedings of 1st International Workshop on OMNeT++, ICST, Marseille, France, March 2008.

[11] T. Hardjono. and G. Tsudik. Ip multicast security: Issues and directions. Annales de Tlcommunications, 55(1):324–340, January 2000.

[12] Y. Imai. Bsd implementations of xcast6. Proceedings of ASiaBSDCon2008 Tokyo,, March 2008.

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

[14] M. Kolberg and J. Burford. An xcast multicast implementation for the oversim simulator. Proceedings of Consumer Communications and Networking Conference (CCNC), 2010 7th IEEE, January 2010.

[15] A. Popescu, D. Constantinescu, D. Erman, and D. Ilie. A survey of reliable multicast communication. Proceedings of the 3rd Euro-NGI conference on Next Generation Internet Networks (NGI 2007), Trondheim, Norway, 2007.

[16] New York Times. Cheap, ultrafast broadband? Hong Kong has it. New York Times, http: // www. nytimes. com/ 2011/ 03/ 06/business/06digi. html?_ r= 1s, November 2007.

[17] A. Varga. The inet framework project site. http: //inet. omnetpp.org/

[18] A. Vareg, OMNeT++ community site. http: //www. omnetpp. org.

[19] [A. Varga.. The omnet++ discrete event simulation system. Proceedings of the European Simulation Multiconference, pages 319− 324, June 200 1