BSD implementations of XCAST6

Yuji IMAI, Takahiro KUROSAWA, Koichi SUZUKI, Eiichi MURAMOTO,

Katsuomi HAMAJIMA, Hajimu UMEMOTO, Nobuo KAWAGUTI

XCAST fan club, Japan.

Abstract: XCAST [RFC5058] is a complementary protocol of Multicast. In contrast with the group address of Multicast, XCAST specifies the destinations by the list of unicast addresses, explicitly. Multicast is very scalable in terms of the number of receivers because membership of the group-address destination is implicitly managed on the intermediate routers and the number of receivers is potentially infinite. On the other hand XCAST is very scalable with respect to the number of groups. It is necessary for bi-directional multi-party communication systems such as tele-presence to deal with the large number of groups. We implemented XCAST6, the IPv6 version of XCAST, to prove the concept. Using our implementation, we operate multi-party video conference systems both on experimental overlay network and on native IPv6 network. In this paper, we will describe detailed implementation of XCAST6 on FreeBSD and NetBSD. We will also discuss about simplicity not only of implementation but also of operation.

1. Introduction

XCAST (eXplicit Multi-Unicast) is a protocol to deliver one datagram for small number of destinations simultaneously. It is considered as a complementary mechanism of Multicast. While Multicast is suitable to send a datagram for very large number of receivers, XCAST is good for delivering a datagram to small number of receivers, but capable of dealing with the large number of groups.

We implement XCAST6, the IPv6 version of XCAST, on the various flavor of BSDs to validate usefulness of the protocol. XCAST can be implemented so simply because the protocol itself is designed simply based on existing unicast mechanism. Using our implementation, communities can easily operate multi-party conference systems both on experimental overlay network and on native IPv6 network.

2. XCAST: eXplicit Multi-Unicast

Ordinary, “Multicast” is a system defined by [STD 5]. An IP Addresses in the special IP address range, formerly called Class D of IPv4, is assigned as identifier of the group of hosts or interfaces connected to the Internet. Potentially, any host can join to and leave from the group any time. The embership of the groups are maintained on the routers with distributed manner. A datagram transmitted for the multicast address is relayed and forwarded to all member of the group, duplicated on the routers.

It is a buried history that in a very early discussion of the Internet community,Multicast was not for a virtual group address but the list of unicast addresses [AGUILAR]. In its design, destinations were explicitly embedded in the option header of the IP datagram. Group address extension was introduced as an improvement from Aguilar's. The main point of the improvement was a scalability of the number of receivers. Maximum number of the original Multicast was limited by the length of IP option header. The Internet community considered it must be too small. So, they choose to introduce the special addresses for the destinations. By keeping membership on routers, datagrams can be transmitted for the special addresses as same as for unicast destination. By this extension “Multicast” got very good scalability with respect to the number of members per group, potentially unlimited.

Based on this design choice, “Multicast” deployment was started and difficulties appeared. One of the points is how to maintain the Multicast routing information. [RFC2902] Because the membership of group-multicast is dynamic,

routing information changes so frequently and looks difficult to be aggregated. Some insist it is impossible [SOLA]. Recently, it becomes consensus that scalability of group-multicast with respect to number of groups is not so good.

For some types of multi-party communication, the lack of the scalability of the number of groups is critical. Multi-party video conference is typical example. The participants, the source and the destinations of multicast", are sparsely distributed over the Internet. The system needs the Multicast routes as many as the participants. From the viewpoint of network operators, routers must maintain the multicast routes as many as the transmitters. It is easy and natural to consider the potential multicast transmitters are over hundred-millions, according to the number of users of instant messages or softphones like Skype. Today full routes of the unicast of the Internet exceed 250,000 and IAB considers it is facing serious scaling problems. [RFC4984] That means we need other mechanism than Multicast to realize this type of communication.

XCAST is a re-invention of primitive work of Aguilar's [RFC5058]. One of the improvements from Aguilar's is the way to store the list of destinations. For IPv4, they are encoded in the newer defined option header and for IPv6, the routing header. The number of destinations is up to 124 for IPv6 version. For typical usage of human multi-party communication, this limit would not be problem because it would be difficult for people to make conversation with more than 100 people simultaneously.

As XCAST datagram has an explicit list of unicast addresses, routers can duplicate and forward them using existing unicast routing information without any other like those of Multicast.

One big problem to deploy XCAST in the existing Internet is how to make XCAST datagram pass over the non-XCAST routers. For this purpose, XCAST6 prepares the mechanism called semi-permeable tunnel. The raw XCAST6 datagram starts with IPv6 header with special destination address ALL_XCAST_NODE, the group-multicast address specially assigned. It indicates that the datagram is XCAST6 one and a list of destinations is following in the routing header. The semi-permeable tunnel is encapsulation trick like IP over IP. A semi-permeable XCAST datagram is covered with an additional IPv6 header and a hop-by-hop options header. In the outer IPv6 header, a temporal address is embedded in the destination field that is one of the list of XCAST destinations the datagram has not been reached yet. The hop-by-hop options header marks the need for XCAST routing process. With this preamble headers, a semi-permeable datagram looks like an ordinal IPv6 datagram with a unicast destination. So, XCAST6 datagram can travel through the IPv6 network even that include non-XCAST6 routers. Only when the datagram passes on the XCAST6 routers, it detects the hop-by-hop options header, then checks the list of destinations in the routing header and duplicates the datagram if needed.

3. Implementation

We implemented XCAST6 on NetBSD and FreeBSD kernels. The set of codes consists of the following components:

- interface for user processes
- routing header processing
- the xcst network interface

When a user process issues an XCAST6 packet with the sendmsg(2) system call, the kernel needs to handle the request and send it to the network by processing the XCAST6 routing header. The packet is forwarded by routers and then reaches a node with XCAST6 support. The node should parse the packet, process the routing header, and forward it to the destinations listed in the routing header. If the list of the destinations contains the address of the node, the packet should be passed to the upper layer (UDP or ICMPv6).

The XCAST6 packets are usually sent in the semi-permeable tunneling format. When the kernel sends them to the network, it should encapsulate them. Also, it should be ready to receive the encapsulated packets. The xcst network interface is used for handling encapsulation.

The following sections describe how those components act on processing XCAST6 packets.

3.1. Interface for user processes

User processes can send XCAST6 packets by sendmsg(2). With sendmsg(2), IPv6 extension headers are added to the packets using the msg_control field of the msghdr structure as described in [RFC3542]. On sending XCAST6 packets, ALL_XCAST6_NODES is specified as the destination address in msg_name (regardless of whether semi-permeable tunneling is required or not) and the XCAST6 routing header in msg_control. Here we implemented codes so that the routing header is not rejected as unknown but is stored and associated with the message for later processing; we don't process the routing header itself here but simply pass it to the packet output routine.

On receiving side, user processes can receive XCAST6 packets just the same way as unicast packets. There is no need to change receiver code of socket interface. Also, we don't need to change code for joining/leaving multicast groups since XCAST6 doesn't require keeping track of joining/leaving unlike group multicast or source specific multicast.

3.2. Sender side

The ip6_output() function builds IPv6 packets and outputs them to network interfaces. The XCAST6 routing header associated with the message is also placed in the packet by the function, but is not processed yet.

At the end of ip6_output(), the function outputs the packet to the link layer according to the routing table. By routing packets destined for ALL_XCAST6_NODES to the xcst network interface, we can pick up them there with keeping changes to the ip6_output() function as few as possible.

The output function of the xcst interface checks whether the packet has a XCAST6 routing header or not. Packets with XCAST6 routing headers are passed to the routine of XCAST6 routing header processing. Packets that don't have XCAST6 routing headers are dropped.

3.3. Routing header processing

The XCAST6 routing header contains a destination address list and a bitmap. The bit in the bitmap indicates whether the packet should be delivered to the corresponding address or not. The address can be classified from the viewpoint of reachability as follows:

- unreachable
- assigned to the local node
- directly reachable from the local node (ex. on the same link)
- reachable via routers

As for addresses reachable via routers, there may be addresses that next hop routers are the same. We should group addresses by the next hop and send a packet for each group in order to minimize the number of copies of the packet in the network. For this reason we need to look up the routing table first for all the addresses that the packet needs to be sent. Once the grouping of the addresses is done, the packet can be sent for each next hop, normally encapsulated in the semi-permeable tunneling format. Addresses directly reachable from the local node are handled as no other addresses shares the same next hop. If the address list in the routing header contains the address

assigned to the local node, the local node is also expected to receive the packet. Such a packet is passed to the upper protocol layer (UDP or ICMPv6) in addition to being forwarded to next hops.

3.4. Forwarding XCAST6 packets

The node that supports XCAST6 should forward the incoming XCAST6 packets sent by other nodes in addition to sending packets requested by user processes. This section describes how incoming XCAST6 packets are processed.

The XCAST6 packets are usually encapsulated in the semi-permeable tunneling format. Packets without encapsulation can easily be detected as the XCAST6 packets because the destination addresses are ALL_XCAST6_NODES. The destination address of encapsulated packets may not be the local node address, but the hop-by-hop options header in the packet indicates that each router on the path should inspect the header deeper. We needed to add the code to process hop-by-hop options header with the type of XCAST6 and to pass the packet to the extension headers routines.

Also, packets with encapsulation need to be decapsulated. We implemented the decapsulation in the xcst interface by using encap_* functions provided by netinet/ip_encap.c just like gif(4) or gre(4).

The extension headers of the incoming XCAST6 packets are then processed as normal IPv6 packets. We added the codes where the routing header is handled in order to pass the packet to the routine of XCAST6 routing header processing. The same routine that is described on sending the packets is also used here.

3.5. The xcst network interface

The xcst network interface is a pseudo device that is introduced for picking up XCAST6 packets. As described above, packets that are going to be sent by the local node are picked up at the output routine of the xcst interface with keeping changes to the ip6_output() function as few as possible. Also, the encapsulated packets is picked up by the xcst interface and then decapsulated. The xcst interface contributes for localizing the changes to the existing IPv6 implementation and simplify the XCAST6 implementation.

3.6. ICMPv6 support

It became apparent that diagnostic packets were necessary during the deployment of XCAST6. The path of an XCAST6 packet between a sender and one of receivers is usually not the same as the path of a unicast packet between the sender and the receiver. On IPv6 unicasts, the sender can check reachability to the receiver by the ICMPv6 Echo Request message. On XCAST6, the ICMPv6 Echo Request message is unusable since the destination address field in IPv6 header is a multicast address.

In order to check the reachability from the sender to one of the receivers, we experimentally introduced a sub-option in Hop-by-hop options header. The sub-option specifies which destination should respond with the ICMPv6 Echo Reply message. We have slightly modified the receiver routine of ICMPv6 to implement this functionality. The “ping6x” userland program has also been implemented.

4. Related work

In this section, we describe the related implementations on XCAST6 and the operational activities called “X6bone”.

4.1. Group management

The group management for XCAST6 is completely separated from the function of forwarding or routing. That enables application developers to utilize their own application-specific group management functions free from group as well as group multicast address schemes, There are several implementations that help XCAST

application developers to make the group formation and management.

4.1.1. Xcgroup

The application of XCAST6 requires the list of destinations. We created a CGI script for httpd and a client program called "xcgroup". The application user invoke “xcgroupsrv” with a URL of CGI for managing the information of group name and its membership (i.e., participating nodes). Client “xcgroup” periodically sends a query to the “xcgroupsrv” CGI script by http and the CGI script acquires the IPv6 source address of the http query and records it in a list. Then it retrieves all the IPv6 source addresses recorded in the list and provides the client program that information. As a result, consistent membership information among the participants can be provided. Xcgroup announces the list of acquired unicast addresses for Mbone tools via mbus [RFC3259] so that XCAST applications on the same host can share the address list.

4.1.2. Group Management using SIP

The method how to arrange the multiple IP address for XCAST session had been discussed in [SIPSDP]. It proposed the method to exchange IP addresses and port numbers between participants using SIP and the method to carry multiple port numbers in XCAST header.

4.2. Testbed and deployment activities

To prove the concept of XCAST, WIDE XCAST WG started operating the testbed called X6Bone.It consists of a virtual overlay IPv6 network over IPv4 network by tunnels connecting many SOHO routers to a HUB XCAST router via ADSL and FTTH. The NAT traversal tunnel is made by the implementation of DTCP: Dynamic Tunneling Configuration Protocol[DTCP] and L2TP[L2TP] which invoke DHCPv6 PD over PPP over L2TP over UDP over IPv4. The XCAST6 datagrams traveling on the X6Bone could be branched at the HUB routers, thus eliminating the problem of inefficient daisy-chained connections by semi-permeable tunnel trick.

4.2.1. XCAST6 enabled VIC and RAT

Well-known Mbone tools VIC and RAT have been modified and integrated with XCAST6. On the sender side, original VIC and RAT bind the transmission socket to the multicast group address. We modified the procedures to call library function, XCASTAddMember() to append unicast addresses of the participated node that is provided by group management mechanism. On the receiver side, no modification is necessary since the payload of an XCAST6 datagram can be acquired by calling ordinary recv() function. The total amount of additional code is less than 200 lines for vic and 150 lines for RAT.

We have utilized the modified VIC and RAT on the X6bone. Next section explains the deployment activities with them.

4.2.2. Deployment activities

We envision that typical target usage of XCAST as casual conversations to exchange emotional atmosphere among private community like families and friends. Based on this assumption, we made various trials on the X6bone including the open source developer BoFs, the wedding ceremony, conversations with person in transportation, moving bicycle[ROMAIN] and boarding on the airplane flying over north pole. By January of 2008, the number of groups joining the meeting has increased to 10 organizations including BSD User groups in Asia and France.

In order to make these kind of conversation easily and fruitful, many related tools were developed utilizing *BSD variants as follows.

- Fukidashi-kun: Tools to display words on his/her VIC screen just like mumbling by Yasushi Oshima of Nagoya BSD user club.

- The device driver modifications of Web cam for FreeBSD/NetBSD by Takafumi Mizuno of Nagoya BSD user club.

- Instant XCAST6 CD: Bootable *BSD environments with XCAST6 packages. “Ebifurya” based on NetBSD. “FreeSBIE” based were contributed by Daichi Goto.

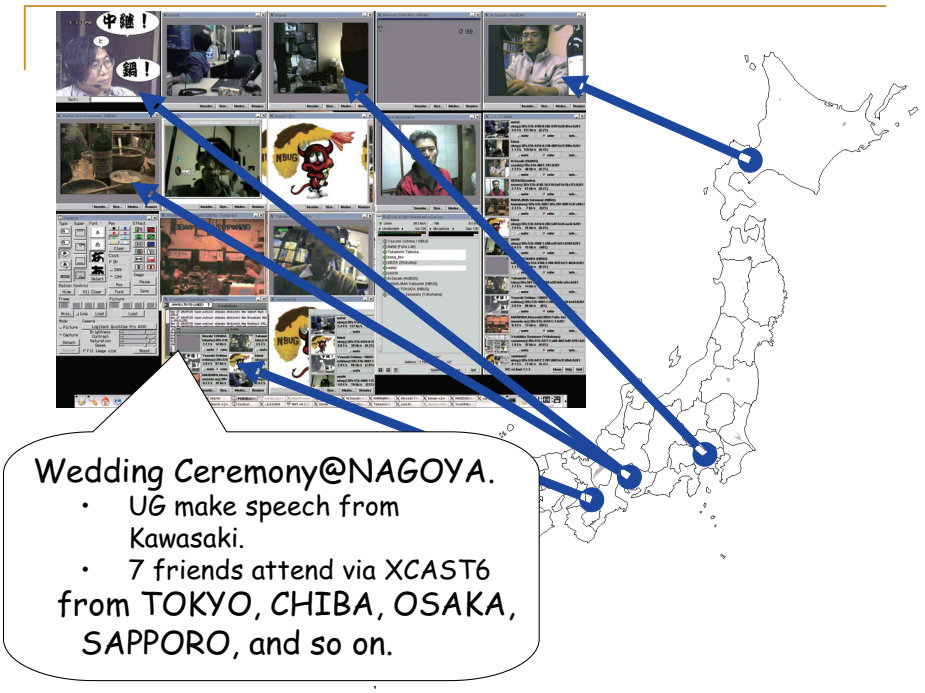

Figure 1: example screen shot of XCAST meeting on X6Bone in Japan.

5. Conclusion

We implemented XCAST6 protocol onto FreeBSD and NetBSD. In order to realize XCAST logic, we utilize existing unicast basis *BSD intrinsically equipped. As a transmission node, XCAST packet is just formulated and triggered to send out in the way of extended socket API, [RFC3542]. Kernel determines directions of the datagram to be forwarded looking up the next hop information of unicast routing basis and branches if needed. Tunneling pseudo device helps the datagram passing over the non-XCAST networks.

To deploy the XCAST6 network, we can use many tools to establish IPv6 connectivity without any modification. We operate X6Bone, the experimental XCAST6 network, using l2tp, dtcp, PPPoE and Ethernet emulation of Packetix. We can use such tools if they exchange ordinal IPv6 unicast datagrams.

Style of the interface for application keep similar with one of group multicast so that MBone tools can be easily modified to handle XCAST6.

We made the extended version of unicast reachability diagnosis tools such as ping6, traceroute6 with small modification with ICMP6 functions. Using these tools, it become very easy to check XCAST6 reachability compared with the group multicast specifics like mtrace.

With our X6Bone experience, we discussed with IETF community and convinced them

that there should be other class of multicast than group-address one. For the topic, SAM RG: Scalable Adaptive Multicast Research Group was made in IRTF. We keep reporting XCAST operational experiments including implementing status for various OS, Linux, Windows and *BSD of course.

Acknowledgment

Yoichi SHINODA of JAIST, Jun-ichiro "itojun" Hagino of KAME project and Hideaki YOSHIFUJI of Usagi project gave us great advice for our XCAST6 implementation. Takamichi TATEOKA, Sohgo TAKEUCHI and Tomo TATSUMI has been contributing the operation of X6bone. Rick Boivie of IBM and John Buford of Avaya kept encouraging us to make XCAST concept as an experimental RFC. And we express our greatest gratitude to WIDE Project, Asian Internet Interconnection Initiatives(AI3), Korean IPv6 community, ETRI and IRISA for the cooperation of our experiments.

Reference

[STD 5] S. Deering, "Host Extensions for IP Multicasting", STD 5, RFC 1112, September 1989.

[AGUILAR] L. Aguilar, "Datagram Routing for Internet Multicast- ing", Sigcomm84, March 1984.

[SOLA] M. Sola, M. Ohta, T. Maeno. “Scalability of Internet Multicast Protocols”,INET'98, http://www.isoc.org/inet98/proceedings/6d/6d_3.htm

[RFC2902] S. Deering, S. Hares, C. Perkins, and R. Perlman, "Overview of the 1998 IAB Routing Workshop", RFC 2902, August 2000.

[RFC3259] J. Ott, et al., "A Message Bus for Local Coordination", RFC3259, April 2002

[RFC3542] W. Stevens, M. Thomas, E. Nordmark, T. Jinmei. “Advanced Sockets Application Program Interface (API) for IPv6”. RFC3542, May 2003.

[RFC4984] D. Meyer, L. Zhang, K. Fall, “Report from the IAB Workshop on Routing and Addressing”, RFC4984, September 2007

[RFC5058] Boivie,R.,N. Feldman, Y. Imai,W. Livens,D. Ooms, “Explicit Multicast (Xcast) Concepts and Options”, RFC5058, November 2007.

[SAINT] Y. Imai, H. Kishimoto, M. Shin, Y. Kim, “XCAST6: eXplicit Multicast on IPv6”, IEEE Symposium on Applications and Internet Workshops, January 2003.

[DTCP] H.Umemoto, DTCP homepage, http://www.imasy.or.jp/~ume/published/dtcp/

[L2TP] H.Umemoto, L2TP document, http://www.imasy.or.jp/~ume/presentation/CBUG-20070421/l2tp-pd.odp

[ROMAIN] R. KUNTZ,”The E-Bicycle Demonstration Setup on Tour de France 2006”,http://member.wide.ad.jp/tr/wide-tr-nautilus6-ebicycle-tour-de-france-00.pdf

[SIPSDP] B. Van Doorselaer, “SIP for the establishment of xcast-based multiparty conferences”, http://www.tools.ietf.org/html/draft-van-doorselaer-sip-xcast-00, July 2000.