INTERNET DRAFT
<draft-abade-xcast20-routing-engine-spec-00.txt>

Odira Elisha Abade
Nobuo Kawaguchi
Nagoya University

October 19, 2009
Expires April 19, 2010.

Design and Implementation of an XCAST6 Routing Engine

Status of this Memo

Copyright Notice

Abstract

XCAST6 (Explicit Multiunicast on IPv6) is a new protocol defined in RFC 5058. In XCAST, the list of destinations is explicitly encoded within the data packets instead of using a multicast group address. Research is currently ongoing on two versions of XCAST6 and this

document describes the design and implementation of a routing engine for the new version in which the use of hop-by-hop options header has been eliminated. This draft explains why there is a need for an XCAST6 routing engine, highlights the requirements for its implementation, the design process and how to eventually implement the routing engine to allow for deployment of XCAST6 protocol.

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC-2119 [RFC2119].

Table of Contents

1. Introduction

Explicit multiunicast (XCAST) protocol specified in RFC 5058[5] is a new multipoint communication scheme which supports a large number of small sessions. This property results from the fact that in XCAST, a list of destination addresses is explicitly encoded within the data packets instead of using a multicast group address when sending packets from one source to multiple receivers.

Implementation of XCAST on IPv6 is referred to as XCAST6. As specified in RFC 5058, in addition to two IPv6 headers and a routing extension header, XCAST6 utilizes hop-by-hop options header to ensure the XCAST6 packets are routed in the Internet. This implementation is referred to as XCAST6 version 1.0[5]. The contents of this header need to be processed by every node along the path of an IPv6 datagram[17]. For routers, it requires deeper packet inspection through the slow forwarding path[1]. This and other shortcomings make hop-by-hop options header unpopular with the commercial hardware router manufacturers since it substantially increases the router's CPU load[1]. This undue CPU overload can be exploited to launch a distributed denial of service attack[17].

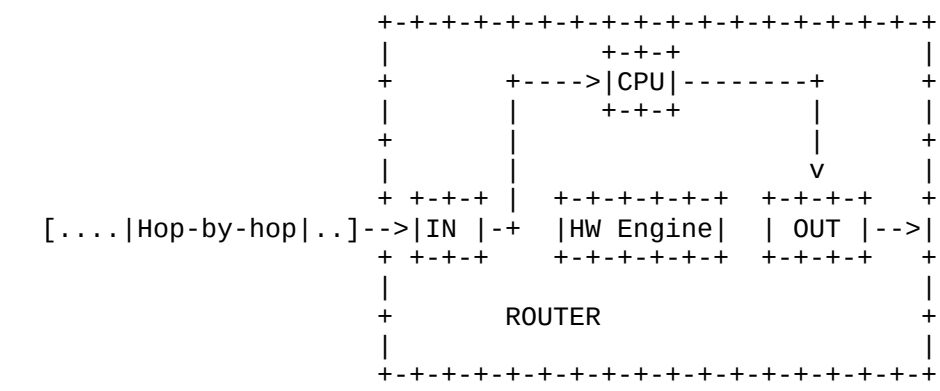

Figure 1. Hop-by-hop header is processed is processed in the CPU.

Due to these limitations associated with the use of hop-by-hop options headers, we have been researching on how to eliminate these headers in the implementation of XCAST6. This has resulted in us coming up with XCAST6 version 2.0 in which hop-by-hop options headers are not used in the routing process.

XCAST6 version 2.0 eliminates the use of hop-by-hop options header but still challenges exist in that most of the presently available commercial routers are not aware of the XCAST6 packet structure and its processing algorithm. An alternative method on how to route these packets is therefore of paramount importance. This document therefore describes this implementation, we call, an XCAST6 routing Engine.

2. XCAST6 version 2.0 headers

Before we describe the XCAST Engine, we seek to briefly describe the structure of an XCAST6 version 2.0 datagram. A detailed explanation and illustration of these headers are in RFC5058[5] and its associated upgrade Internet draft[7]. With hop-by-hop options header eliminated, XCAST datagram in XCAST6 version 2.0 will at minimum

comprise of: - Two IPv6 headers, - One routing extension header, - A transport header (usually UDP) and - The datagram payload. Basically in the header format should be as shown below:

[IPv6 (semi-permeable) header| IPv6 (inner header)| Routing header| Transport header| Payload]

2.1. The outer IPv6 header

The outer IPv6 header is used for semi-permeable tunneling. In this IPv6 header, the values of the source and destination fields are changed on each node in which the XCAST6 processing occurs. The source address will usually be the unicast address of the source node or address of the last branching router, while the destination address will be assigned to that of the host whose IP address appears first in the bitmap of destinations.

The traffic class field is assigned the value of "010111XX" which comprises a set of bits allocated for experimentation[14] by the IRTF SAM RG and those for explicit congestion notification (ECN) as specified in RFC 3168[16]. The flow label field is composed of a 20-bit, three parts allocated as follows: The first 8bits are "01010111" while the 9th bit to 13th bit default to 00000. The 14th to 20th bits are for the offset of the ICMP target that specified one of the destinations in the address list for which ICMP reflection, echo replies and errors are not ignored. The next header field points at "IPv6 header" (41) which is the inner IPv6 header in an XCAST6 datagram.

2.2. The inner IPv6 header

The inner IPv6 header maintains the source address of the original sender while its destination address is marked as ALL_XCAST_NODES. This header is processed by the node or router whose address is specified in the destination field of the first header. If the node is XCAST aware, then it knows how to process the datagram using the XCAST algorithm. However if the node is not XCAST-aware, it simply drops the datagram because the address, "ALL_XCAST_NODES" is within the range of multicast addresses and should be ignored without any ICMP notification as described in RFC2463.

2.3. Routing Extension header

The XCAST6 routing extension header is a variation of the IPV6 routing header specified in RFC2460 and encloses the complete list of unicast addresses of the destination nodes. The next header and the header extension length fields specify type of the next header and the length of the routing header respectively [13]. The type value in

an XCAST6 routing header is 253 from the experimental range as defined in RFC4727[14]. The fourth octet of an XCAST6 routing header must be set to zero. This ensures that non XCAST6-aware nodes or routers only drop the packets but send no ICMP errors to the datagram source hence eliminating possible misuse that might be exploited to launch DDOS spoofing attacks.

Because the length of the routing extension header is limited, (8 by 255 octets) the maximum number of destinations that an XCAST6 datagram can contain is thus 126. It must be noted that this is a limitation posed by the routing extension header but not at all emanating from the XCAST algorithm.

2.4. Transport header

This specifies the transport layer protocol for use. Most of the XCAST6 testing have been done with UDP for both data and multimedia content.

2.5. Payload

This refers to the usual payload data. XCAST has been tested using both data and multimedia payload content over UDP.

3. What is an XCAST Routing Engine

With the datagram structure described above, an XCAST6 datagram certainly needs a little care in handling to ensure that it shall be routed successfully over the Internet from the sender to the set of receivers in a multipoint communication session. However the challenge is that the commercially available routers still do not have this functionality inbuilt. This poses challenges to real world deployment of XCAST6 over the Internet.

To break this barrier, we choose to implement a scheme in which an XCAST6-aware node is connected to the network core-router such that all XCAST6 packets inbound to the network core router are forwarded to this XCAST-aware node for processing. The core network router examines the traffic class of the inbound packets and if they match those of an XCAST6 packet, it forwards them to the XCAST6-aware node. The XCAST6-aware node applies the XCAST packet processing algorithm as specified in RFC5058 and sends the packets back to the core network router which then forwards them as explained earlier. This XCAST6-aware node therefore acts an "XCAST6 software router" and ensures proper routing of XCAST6 packets even if the core network router is not XCAST6-capable. This "software router" is what we refer to as an "XCAST6 Routing Engine".

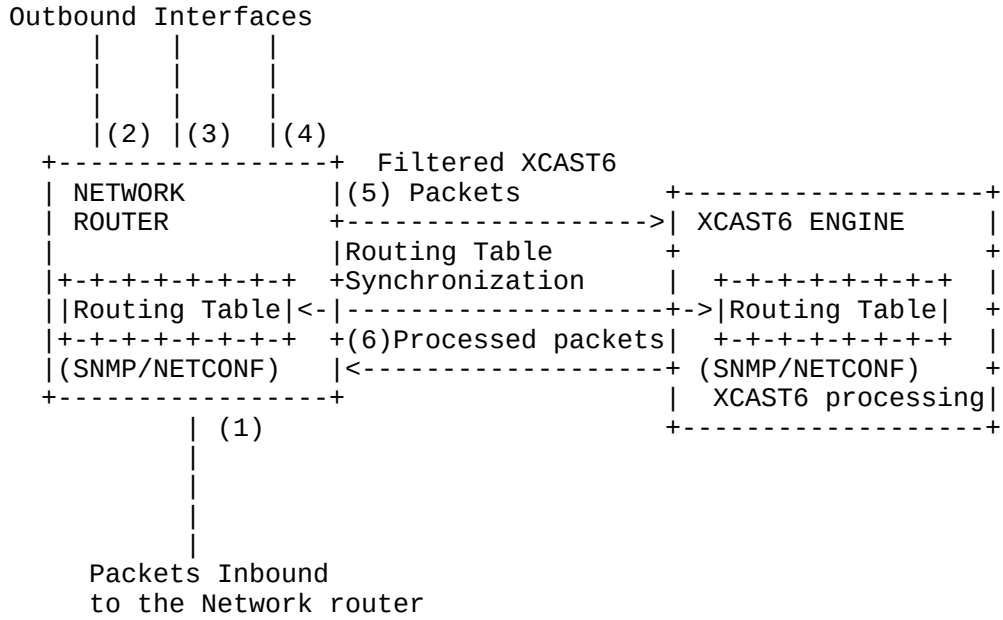

Figure 2

3.1. Why we need an XCAST6 Routing Engine

The reasons for implementing an XCAST6 Routing Engine therefore are: - A lot of investments are already in place in terms of existing commercial routers that are not XCAST6-aware and cannot be eliminated. - Hop by hop options header that we used in XCAST6 version 1.0 are susceptible to distributed denial of service attacks hence it is unpopular among commercial router vendors and we had to change that. - Currently, deployment of XCAST6 is not easy.

4. Requirements for implementing an XCAST Routing Engine

For the implementation of an XCAST6 Routing Engine we have a basic set of requirements namely: - An XCAST6 enabled computer. Currently XCAST6 implementations exist for LINUX, FreeBSD and other BSD operating systems. - The network core router should be able to support policy routing especially filter based forwarding scheme. - The XCAST Routing Engine and the network core router should be able to support SNMP and/or NETCONF protocols[2]. - The XCAST Engine should be running on a hardware platform that supports IEEE802.1Q (VLAN tagging).

Once the above requirements have been met, then the following considerations must be investigated: - How to filter XCAST6 packets

at the core network router to ensure that only XCAST6 packets are sent to the XCAST6 routing engine for processing in accordance with the XCAST algorithm. - How to synchronize the routing tables of the XCAST6 routing Engine and that of the network core router so as to ensure that when XCAST6 packets are processed, the most uptodate network structure is used. - How to forward the processed XCAST6 datagrams to their next hop routers or destinations in a manner that would be the same as if the processing was done at the core network router. - Performance considerations of the XCAST Engine must also be investigated because the objective of the XCAST Engine is to determine the feasibility of deployment of XCAST6 protocol in commercial routers.

4.1. Filtering of XCAST6 Packets

To identify XCAST6 packets, policy based bit matching should be done on inbound packets at each of the core router's interfaces except the one to which the XCAST Engine is connected. The matching is done against the traffic class of IPv6 packets and those with 010111XX class are identified as XCAST6 packets. Policy routing and filter based forwarding is therefore a required feature in the commercial routers to which the XCAST Engine are connected.

4.2. Synchronizing the routing tables

The routing table of the XCAST Engine must mirror as closely as possible that of the network core router. This is to ensure that XCAST6 packets processed appear as if they were actually processed at the core router. To realize this, mechanism must be in place that ensure that changes in the routing table of the network router are immediately effected in the XCAST6 routing Engine.

SNMP scripts can be defined that retrieve the IPv6 routing table MIB of the core router and passes it to a program that updates the routing table of the XCAST6 Routing Engine. It should be noted that some SNMP MIBs, including the routing table MIB are processor intensive hence an alternative implementation of this synchronization is currently under investigation. The alternative approach seeks to investigating the use of NETCONF[2] in realizing the same objective since it is hypothesized that when the routing table of the core router is too large, using NETCONF instead of SNMP would help reduce the router's CPU load considerably.

In the current implementation, both SNMP and NETCONF have been tested using a polling approach whereby the corresponding scripts regularly poll the core router over a specified unit of time. Other approaches are being considered preferably where the synchronization process will be initiated by the core router only when its routing table has

changed.

4.3. Forwarding of the processed XCAST6 datagram

To realize effective forwarding of the XCAST6 packets, a set of virtual interfaces are cloned between the XCAST Engine and the core router. Typically, this should reflect the number of interfaces on the core router. Each of the cloned interfaces is assigned a different subnetwork and when synchronizing the routing tables as described in 4.2 above, each of these interfaces will handle their packets in a similar manner to their corresponding interfaces on the core network router.

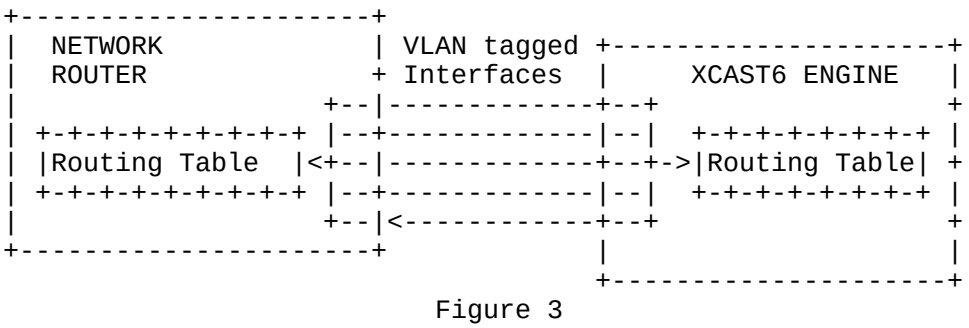

Figure 3

This way, XCAST6 packets shall be forwarded to their next-hop routers or destinations as if they were processed from the core network router.

4.4. Performance characteristics

The real objective of implementing an XCAST6 Routing Engine is to pave way for real world deployment of XCAST protocol in commercial routers. To achieve this, we should be able to understand the performance characteristics of XCAST protocol so as to seek for the feasibility of its deployment in commercial routers. The XCAST6 Routing Engine performance measurement seeks to benchmark the XCAST protocol with respect to the following performance metrics: -Throughput -Latency and latency distribution -Packet loss rate -CPU utilization -Memory utilization -Context switch and system call overheads -Average system load

5. XCAST Engine APIs

Implementing the XCAST6 Routing Engine should be fast and as simple as possible. To realize this, there is a need of creating a set of Application Programming Interfaces (APIs) that can be easily invoked.

While the current work is yet to finalize this, it is an area we are currently working on. Once completed, we shall seek to explore the full potential of an XCAST6 routing Engine.

6. IANA Consideration

XCAST6 version 2.0 uses the following IANA resources from experimental range. IANA should consider assigning the following resources to avoid the conflict with any other experiments similar to XCAST6 version 2.0, should such an experiment appear.

(1) DSCP (2) Multicast Address for ALL_XCAST_NODES (3) Routing Type of IPv6 Routing Header (4) Option Type of IPv6 Destination Option Header

7. Security Consideration

To counter measure the problem of unlimited repeat delivery (RH0 problem), XCAST6 version 2.0 specification defines the usage and handling of "hoplimit". When an XCAST6 packet reaches a node (or a router), whether the node is XCAST6 aware or not, it reduced the hoplimit value of the outer IPv6 header by 1. Additionally, the un-delivered mark ’1’ of the bitmap field always decreases when a packet is copied. It therefore means that, the edge of delivery tree of a single XCAST packet is 255(hoplimit) * 126(number of bitmap). The maximum stretch of the delivery tree is less than 256.

8. Informative References:

[1] IPv6 Extension Headers Review and Considerations, Cisco Systems, 2006

[2] R. Enns, Ed, NETCONF Configuration Protocol, RFC4741, Juniper Networks Inc, December 2006.

[3] JUNOS NETCONF API, Release 9.3, Juniper Networks Inc, 2008

[4] C. Partridge and A. Jackson, IPv6 Router Alert Option, RFC2711, October 1999

[5] R. Boivie, et al., "Explicit Multicast (Xcast) Concepts and Options", RFC 5058, November 2007

[6] S. Deering, "Multicast Routing in a datagram internetwork", PhD thesis, December 1991.

[7] XCAST6 (version 2.0) Protocol Specification, Internet Draft, draft-ug-xcast20-protocol-spec-00.txt, Feb 2008, Work in Progress.

[8] M. McKusick, George Neville-Neil: The Design and Implementation of the FreeBSD Operating System, Addison-Wesley, July 2004

[9] JUNOS Feature guide, Release 9.1, Juniper Networks Inc.

[10] Y. Imai et al, BSD implementations of XCAST6, in proceedings of ASiaBSDCon2008 (Mar. 2008).

[11] Dominique C, Rex Young, "Build a network router on Linux", 2003

[12] Andre Ben Hamou, Practical Ruby for Systems Administration, Apress, June 2007.

[13] S. Deering, R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998

[14] B. Fenner, "Experimental Values in IPv4, IPv6, ICMPv4, ICMPv6, UDP, and TCP Headers", RFC 4727, November 2006.

[15] K. Nichols et al, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, December 1998

[16] K. Ramakrishnan et al, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001

[17] The case against Hop-by-Hop options, Internet Draft, draft-krishnan-ipv6-hopbyhop-02.txt, February 2008, Work in Progress.

[18] S. Bradner, Harvard University, Key words for use in RFCs to Indicate Requirement Levels, March 1997.

9. Authors' Addresses

Odira Elisha Abade, Graduate School of Engineering, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, 464-8603, JAPAN Email: abade@ucl.nuee.nagoya-u.ac.jp

Nobuo Kawaguchi, Graduate School of Engineering, Nagoya University, Furo-cho, Chikusa-ku, Nagoya, 464-8603, JAPAN Email: kawaguti@nagoya.jp

10. Contributor Addresses

Eiichi Muramoto Matsushita Electric Industrial Co., Ltd. 4-12-4 Higashi-shinagawa, Shinagawa-ku, Tokyo 140-8587, Japan Phone : +81-3-6710-2031 E-mail: muramoto@xcast.jp

Yuji Imai Fujitsu LABORATORIES Ltd. 1-1, Kamikodanaka 4-Chome, Nakahara-ku, Kawasaki 211-8588, Japan Phone : +81-44-754-2628 Fax : +81-44-754-2793 E-mail: ug@xcast.jp

Takahiro Kurosawa E-mail: takahiro.kurosawa@gmail.com