GXcast: Generalized Explicit Multicast Routing Protocol

Ali BOUDANI, Alexandre GUITTON and Bernard COUSIN
Irisa/University of Rennes I
Campus de Beaulieu, 35 042 Rennes Cedex, France
Email: {ali.boudani,alexandre.guitton,bernard.cousin}@irisa.fr

***Abstract*— Recently several multicast mechanisms were proposed that scale better with the number of multicast groups than traditional multicast does. These proposals are known as small group multicast (SGM) or explicit multicast (Xcast). Explicit multicast protocols, such as the Xcast protocol, encode the list of group members in the Xcast header of every packet. If the number of members in a group increases, routers may need to fragment an Xcast packet. Fragmented packets may not be identified as Xcast packets by routers. In this paper, we show that the Xcast protocol does not support the IP fragmentation. We show also that avoiding fragmentation limits the group size that can be handled by the Xcast protocol. First, we describe the Xcast protocol, the Xcast+ protocol (which is an extension of Xcast) and we compare these two protocols with traditional multicast protocols. We propose then a generalized version of the Xcast protocol, called GXcast, intended to permit the Xcast packets fragmentation and to support the increasing number of members in a multicast group. The behavior of the GXcast protocol is analyzed according to several criteria. Finally, we present and evaluate with simulations an improvement to GXcast and we conclude that GXcast is a feasible and promising protocol.**

I. INTRODUCTION

Multicast, the ability to efficiently send data to a group of destinations, has become increasingly important with the emergence of network-based applications like IP telephony, video-conferencing, distributed interactive simulation and software upgrading. A multicast routing protocol should be simple to implement, scalable, robust, use minimal network overhead, consume minimal memory resources, and inter-operate with other multicast routing protocols.

Most of proposed multicast protocols like DVMRP [1] and MOSPF ([2], [3]) perform well if group members are densely packed. However, the fact that DVMRP periodically floods the network and the fact that MOSPF sends group membership information over the links, make these protocols not efficient in cases where group members are sparsely distributed among regions and the bandwidth is not plentiful.

To address these issues, the Protocol Independent Multicast (PIM) routing protocols are being developed by the Inter-Domain Multicast Routing (IDMR), working group of the IETF. PIM contains two protocols: PIM-Dense Mode (PIM-DM) [4] which is adapted to groups where members are densely distributed, and PIM-Sparse Mode (PIM-SM) [5] which is adapted to group where members are sparsely distributed. Although these two protocols share similar control messages, they are essentially proposed for two different kinds of applications.

Traditional multicast protocols [6] can be used to minimize bandwidth consumption by using a delivery tree. However, a router has to keep a forwarding state for every multicast tree passing through it. Thus, traditional multicast protocols suffer from a scalability problem with the number of concurrently active multicast groups. Indeed, the number of forwarding states grows with the number of groups.

There seem to be two kinds of multicast that are important: a broadcast-like multicast that sends data to a large number of destinations and a narrow-cast multicast that sends data to a fairly small group. An example of the first kind of multicast is the audio and video multicasting of a presentation to all employees in a corporate intranet. An example of the second kind of multicast is a video-conference involving three or four parties [6]. Thus, a one-size-fits-all protocol will be unable to meet the requirements of all applications [7]. Providing for many groups of small conferences (a small number of widely dispersed people) with global topological scope scales badly given the current multicast model [8].

Recently several multicast mechanisms were proposed that scale better with the number of multicast groups than traditional multicast does. These proposals are known as small group multicast (SGM) [9] or explicit multicast (Xcast) [10]. Explicit multicast protocols, such as the Xcast protocol, encode the list of group members in the Xcast header of every packet. Xcast assumes that there is no packet fragmentation. However, if fragmentation occurs (*e.g.* if the group size or the data is too large) the fragmented packets will not be identified as Xcast packets by routers. In this paper we propose a generalized Xcast protocol to support the group size increasing and to overcome the fragmentation problem.

In Section II, we describe the Xcast [10] protocol, the Xcast+ [11] protocol (which is an extension of Xcast) and we present the Xcast packets fragmentation problem. In Section III, we describe the GXcast protocol and we study the effect of its parameter on several criteria. In Section IV, we propose an improvement of the GXcast protocol and we evaluate the gain in performance. Finally, we conclude in Section V that the GXcast protocol is feasible and promising.

II. The Xcast and the Xcast+ protocols

To solve the problems of traditional multicast protocols, Boivie *et al.* propose the Explicit Multicast protocol (Xcast). In this section, we describe the Xcast protocol [10], the Xcast+ protocol [11] (which is an extension of Xcast) and we compare them with traditional multicast protocols.

A. The Xcast protocol

The Xcast protocol [10] is a newly proposed multicast protocol to support a very large number of small multicast groups. To send data to a given group, the source first explicitly encodes the list of destinations in the Xcast header of the packet. Then, the source parses the header, partitions the destinations based on each next unicast hop and forwards a packet with an appropriate header to each of the next hops. Each router along the path to destinations repeats the same processing on receiving an Xcast packet. If a router detects that there is only one destination in the destination list of a packet, the packet is converted to unicast. The algorithm realizing the conversion of an Xcast packet to a unicast packet is called Xcast-to-Unicast (X2U). This packet is then forwarded in unicast along the remainder of the route.

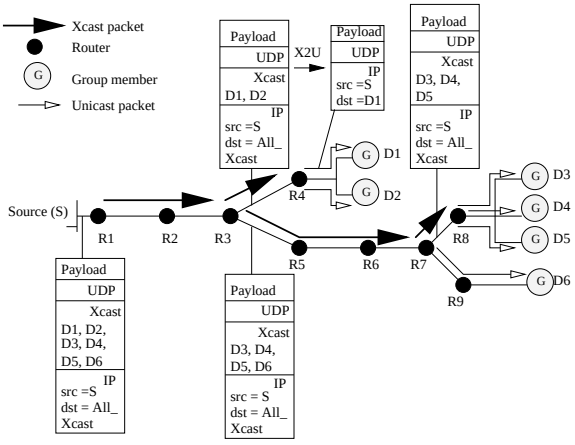

Fig. 1. The forwarding of data in the Xcast protocol.

Example: consider the network represented by Figure 1 and the group G formed from the source S and the six destinations D_1, D_2, D_3, D_4, D_5 and D_6. The source S generates an Xcast packet with the destination list $(D_1, D_2, D_3, D_4, D_5, D_6)$. S proceeds the packet and remarks that R_1 is the next unicast hop for all the destinations. Consequently, S sends the Xcast packet to R_1. R_1 receives the packet and proceeds it similarly. It forwards the packet to the R_2 router which also forwards it to R_3. While proceeding the packet, the R_3 router remarks that R_4 is the next unicast hop for the two destinations D_1 and D_2 and that R_5 is the next unicast hop for the remaining destinations D_3, D_4, D_5 and D_6. R_3 sends to R_4 an Xcast packet containing the destination list (D_1, D_2) and to R_5 an Xcast packet containing the destination list (D_3, D_4, D_5, D_6). Upon receiving the Xcast packet, the R_4 router detects that D_1 and D_2 are two separated stations. R_4 then generates two unicast packets using the X2U algorithm and sends them to D_1 and D_2. Upon receiving the packet, D_1 extracts from the packet the data. The process is similar for routers R_5 to R_9 and for the five remaining destinations.

B. The Xcast+ protocol

Xcast+ is an extension of Xcast for a more efficient delivery of multicast packets [11]. Every source or destination is associated to a Designated Router (DR). Instead of encoding in the Xcast packet header the set of group members, Xcast+ encodes the set of their DRs. When a new member wants to join the group G of source S, it sends an IGMP-join message [12] to its DR. The DR will send a join-request message to the source S. The DR of the source intercepts this message and analyzes it in order to keep track of all concerned DR addresses. When the source S wants to send a message to the group G, it sends a multicast packet. This packet is received by its DR and converted to an Xcast packet using the Multicast-to-Xcast algorithm (M2X). The packet is then forwarded as in Xcast to the DRs, since the destination list in the Xcast header contains the DR addresses instead of the member addresses. Then, each DR converts the Xcast packet to a multicast packet using the Xcast-to-Multicast protocol (X2M) and sends it in its subnetworks.

Example: consider the same network represented by Figure 2 and the group formed from the source S and the five destinations D_1, D_2, D_3, D_4 and D_5. The figure shows where the M2X and X2M algorithms are used. Between the DR of the source and the DRs of the destinations, the packets are forwarded as normal Xcast packets. Suppose that D_6 is a new member which wants to join the group G. D_6 initiates the join of the group by sending an IGMP message for the group (S, G). The DR of D_6, R_9, receives the join request and sends a registration request message toward S. When the DR of S, R_1, receives the registration request message, it

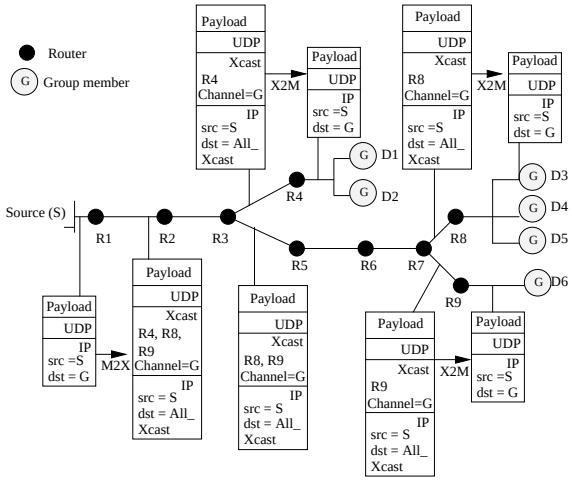

Fig. 2. The forwarding of data in the Xcast+ protocol.

sends back to R_9 a registration reply message and does not forward the registration request message to S. Thus, R_1 is able to know dynamically the set of DRs of the receivers and can fill the destination list of Xcast packets on receiving multicast packet from S.

C. The IP fragmentation mechanism

Due to physical reasons, every link can transfer only a limited volume of information in each packet. The Internet protocol (IP) [13] contains a mechanism called *fragmentation* which makes this limitation transparent for end stations.

The fragmentation mechanism allows a packet to be cut into fragments in order to be suitably transferred on a link. Suppose that a router receives a packet. After having decided on which link this packet should be forwarded, the router checks the maximum capacity of this link which is the Maximum Transmission Unit (MTU). If the packet is too large and unless it is explicitly forbidden, the router cuts out it in order to respect the following constraints: each resulting fragment is an autonomous IP packet, with a valid IP header, each resulting fragment has a size less than or equal to the MTU, and the data is distributed between the fragments.

The algorithm used to fragment IPv4 packets is explained in [13]. The IPv6 protocol also have a fragmentation mechanism, described in [14]. Note that one goal of IPv6 is to avoid fragmentation. This will be discussed later.

D. Xcast packet fragmentation

Let us consider the Xcast packet fragmentation in a router. Since the Xcast packet header may be large, two cases can be considered: either the whole Xcast packet header is short enough to be kept in the first fragment, or the Xcast header has to be cut out. In both cases, the second fragment is not a valid Xcast packet since it has no Xcast header. Thus, these packets cannot be forwarded to receivers and the data they contain is lost. Moreover, in the second case the first fragment contains only a subset of receivers and no data. The first fragment may however be forwarded up to the mentioned receivers, inducing meaningful traffic.

These problems show that the fragmentation of an Xcast packet should be forbidden. This can be done in IPv4 by setting the appropriate flag (Don't Fragment, DF) in the IP header. In order to reach the receivers, the source has to limit the size of its packets to 576 bytes which is the minimum MTU guaranteed by IPv4 on any link. This size limits the number of receivers in an Xcast group to 134 (see Section III-B.1). However, the number of members of the group in Xcast is stored using a 7 bits field, which leads to group containing no more than 127 members. We consider that these two limits are too restrictive. What we propose is a simple mechanism to cancel these limitations in the size of Xcast groups. The performance and the scalability of our proposition will be analyzed.

III. The GXcast protocol

As explained in the previous section, the Xcast protocol can not support large groups due to its incompatibility with the IP fragmentation mechanism. In this section, we propose a generalized Xcast routing protocol, the GXcast protocol, which is designed basically to avoid the fragmentation. Moreover, the GXcast protocol can be parameterized in order to improve the Xcast behavior.

A. The GXcast protocol

The GXcast protocol is a simple generalized version of the Xcast protocol: instead of sending a message to the n destinations, the source limits the number of destinations in a packet to n_M. Thus, the list of n destinations is cut into sub-lists of at most n_M destinations. Each sub-list corresponds to a destination list for an Xcast packet. Several packets may have to be sent in order to deliver data to all the n destinations.

n_M is the parameter of the GXcast protocol and it impacts the protocol performance in terms of several criteria. The choice of n_M is justified in section III-B. GXcast packets are similar to Xcast packets: they have the same header and are treated in the same way by intermediate routers, DR destinations and destinations. The only difference between the Xcast protocol and the GXcast protocol is the packet process at the source or at the DR of the source. The Xcast protocol and the GXcast protocol can therefore inter-operate easily.

Example: consider the same network represented by Figure 2 and the group formed from the source S and

the six members D_1, D_2, D_3, D_4, D_5 and D_6. As in the Xcast+ protocol, the DR of the source keeps track of only the three DRs representing the subnetworks that contain all the destinations: R_4, R_8 and R_9. For this example, n_M is fixed to 2[1]. The source sends a multicast packet to its DR, R_1. R_1 translates it from a multicast packet to an Xcast packet using the M2X algorithm. R_1 notices that there are three destinations in the list for the next hop. Since n_M equals to 2, this list is cut into two sub-lists: one contains the first two destinations R_4 and R_8 and the second contains the last destination, R_9[2]. Each generated packet is treated as a normal Xcast packet as shown on Figure 2.

B. Study of the GXcast parameter

The behavior of the GXcast protocol greatly depends on the value of the n_M parameter. Indeed, as we will see in this subsection, there is a number of criteria that are directly influenced by the chosen value. In the following, we denote by MTU the value of the minimum guaranteed MTU which depends on the IP version used, by E the size of the IP header plus the size of the Xcast header and by IP the size of an IP address. n will represent the number of destinations in the group and d the volume in bytes of data to transfer.

1) Simple behavior: As we have seen in subsection II-C, since a packet has to contain at least one byte of data, the maximum number of destinations n_{max} allowed in an Xcast packet is defined as: $n_{max} = \lfloor \frac{MTU - E - 1}{IP} \rfloor$.

The values $n_{max} = 134$ and $n_{max} = 76$ correspond respectively to the IPv4 and to the IPv6 specifications. The simplest behavior GXcast can have is to fix the n_M value to the n_{max} value. However, this is not efficient for groups having a lot of members (typically more than n_{max}). For example, suppose that IPv6 is used and suppose that $n = 70$ members have joined the group. Each message can contain only 104 bytes of information[3]. In order to send a volume of 10000 bytes, 97 packets are needed. However, less packets would be the result of a better choice of n_M. Choosing n_M equals to 38 allows 616 bytes per packet, which results into the emission of only 34 packets to reach the n destinations. This is approximately three times less.

2) The number of members influenced by a fault: If a drop occurred on a GXcast packet, every member having its address in the member list will be concerned by the drop. To reduce the number of destinations concerned by such errors, small values of n_M should be chosen.

3) Number of generated packets: Considering a group of n destinations and a volume of d bytes to transmit to these members, the number of packets $p(n, d, n_M)$ sent by the GXcast protocol with a parameter of n_M is defined as:

$$p(n, d, n_M) = \lceil \frac{n}{n_M} \rceil \lceil \frac{d}{MTU - E - IP. \min(n, n_M)} \rceil.$$

Recall that in the GXcast protocol, the list of n destinations is cut into sub-lists of size at most n_M. The left part of the expression of p represents the number of sub-lists that will be generated by the GXcast protocol. The right part of the expression of p represents the number of packets needed to transmit d bytes of data. In order to study the behavior of p in terms of n_M, we will consider two cases: $n < n_M$ and $n_M \leq n$. In the first case, we have: $p(n, d, n_M) = \lceil \frac{d}{MTU - E - n.IP} \rceil$.

This expression of p does not depend on n_M. The GXcast protocol behaves in this case in the same way than the Xcast protocol. In the second case where $n_M \leq n$, we assume the following approximation:

$$\bar{p}(n, d, n_M) = \frac{n}{n_M} \frac{d}{MTU - E - IP.n_M} \approx p(n, d, n_M).$$

The $\bar{p}$ function admits a minimum value for: $n_M = \frac{MTU - E}{2.IP} \approx \frac{n_{max}}{2}$.

Since this optimal value does not depend neither on n nor on d, it is very simple to calculate and provides good results in terms of the number of generated packets, we propose it as a default value for the GXcast protocol.

4) Global processing time: To send a fixed amount of data, several packets are generated. The global processing time $t_G(n_M)$ of the GXcast protocol is the sum of the header processing times of these packets. The global processing time for a GXcast packet having n_M destinations is approximately $t_{n_M} = \tau_1 + \tau_2 n_M$, where τ_1 is the processing time of the IP and the GXcast header and τ_2 is the processing time for an entry in the list of destinations (lookup in routing table, generation of packets per outgoing interface, *etc.*). We have then:

$$t_G(n_M) = \lceil \frac{n}{n_M} \rceil t_{n_M} \approx \frac{n\tau_1}{n_M} + n\tau_2.$$

The $t_G(n_M)$ function is strictly decreasing and admits a minimum for $n_M = n_{max}$. Meanwhile, choosing $n_M = n_{max}$ is not realistic as shown in section III-B.1. On the other side, choosing a small value for n_M greatly increases the global processing time. The default value of n_M, $\frac{n_{max}}{2}$, leads to a global processing time which is close to the optimal and is therefore a good compromise.

[1]This assumption is done to make the example easier. In real cases, n_M will usually have larger values.

[2]An improvement to better choose the sub-lists will be described in section IV.

[3]We consider that $E = 40 + 16$ for IPv6, $MTU = 1280$ and $IP = 16$. For IPv4, we would have taken $E = 20 + 16$, $MTU = 576$ and $IP = 4$.

5) Total delay added by the GXcast protocol: Packets generated at the source or at branching routers may be delayed. At the source, packets destinated to at most n_M members are sent successively. Therefore, the last generated packet experiences a larger delay than all previous packets. At a branching router, generated packets per outgoing interface are different and cannot be sent simultaneously. In the same way, latest packets are also delayed.

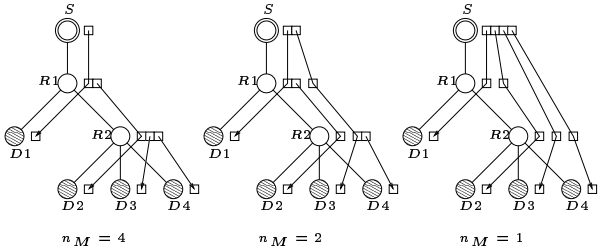

Fig. 3. Delay added by the GXcast protocol.

Let us consider the network represented by Figure 3, where the source is called S and the $n = 4$ members are respectively $D1$, $D2$, $D3$ and $D4$. Let $\delta(D)$ be the added delay experienced by a destination D. In a first time, we consider that $n_M = 4$. S sends a message to $(D1, D2, D3, D4)$. The branching router $R1$ sends a first message to $(D1)$ and a second message (slightly different) to $(D2, D3, D4)$. Thus, $\delta(D1) = 0$ and the three destinations $D2$, $D3$ and $D4$ will experience a small delay. Since the branching router $R2$ sends a first message to $(D2)$, a second to $(D3)$ and a third to $(D4)$. The added delay for the destination $D4$ will be the time for the router $R1$ to do a copy plus the time for the router $R2$ to do two copies, *i.e.*, three units of time ($\delta(D4) = 3$). The total added delay for $n_M = 4$ is $0 + 1 + 2 + 3 = 6$. For information, when $n_M = 2$, $\delta(D1) = 0$, $\delta(D2) = 1$, $\delta(D3) = 1$ and $\delta(D4) = 2$, which yields a total added delay of 4. When $n_M = 1$, $\delta(D1) = 0$, $\delta(D2) = 1$, $\delta(D3) = 2$ and $\delta(D4) = 3$, which yields a total added delay of 6.

The choice of n_M has an impact on the total delay added by the protocol. We have seen that $\delta(Di)$ is due to the time for the source to send several packets and to the time for branching node routers to send several packets. Destination Di is in the $\lfloor (i-1)/n_M + 1 \rfloor$-th packet sent by the source. The time for the source to send the previous packets is $\lfloor (i-1)/n_M \rfloor$ units of time. The number of branching nodes is harder to compute since this number depends on the topology, which is unknown by the protocol. However, in the worst-case, the packet to Di may be delayed $(i-1) \mathrm{mod} n_M$ times. Thus, we have: $\delta(Di) \leq \lfloor \frac{i-1}{n_M} \rfloor + (i-1) \mathrm{mod} n_M$.

Let k be such as $n_M = n/k$. For every member Di, we have: $\lfloor (i-1)/n_M \rfloor \leq k-1$ and $(i-1) \mathrm{mod} n_M \leq n/k - 1$. Finally, we obtain $\delta(Di) \leq k + n/k - 2$. This function admits a minimum value for $k = \sqrt{n}$, thus, for $n_M = n/k = \sqrt{n}$, the total added delay is limited to a minimum value interval.

We propose to choose $n_M = \sqrt{n_{max}}$ for applications that are delay-sensitive. For IPv4, the value for n_M is 11 and for IPv6, the value for n_M is 8.

C. Using Path MTU instead of minimum MTU

In Section III-B, we defined MTU as the minimum MTU guaranteed by IP. However, the value of the Path MTU (PMTU) can also be used since we do not make any assumptions on the stability of the MTU value in our study. The PMTU is the minimum value of the MTU on the links of a path. It can be noticed that the PMTU value is easy to obtain in GXcast, since unicast paths are used. Moreover, the IPv6 protocol stores the PMTU for unicast paths to every destinations.

IV. Proposed improvement

However, a certain locality can be deduced from the longest common prefix of two IP addresses [13]. What we propose is to sort the destination list in the source in order to increase the chance of having a good regrouping. In order to analyze the computation time induced by this sort, three operations have to be considered: the join of a new member to the group, the leave of a member of the group, and the send operation, *i.e.*, the cutting operation.

To reduce the complexity of the proposed sort, we choose to store the set of members as a red-black tree [15]. Inserting, searching or deleting a node in a red-black tree can be done in logarithmic time. In such a tree, the cutting operation can be realized in linear time with two steps: first, an infix walk of the tree produces a sorted list, and then, this sorted list is cut into sublists of n_M elements. Thus, managing a sorted list of members can be done in an efficient way.

To evaluate the impact of the members list sort, we simulate the GXcast protocol on the Abilene topology [16]. The Abilene topology is an experimental Internet2 backbone for educational and research purposes. It consists of 11 nodes and 14 edges.

Figure 4 shows the average cost of the trees for $n = 210$ destinations, divided in packets of $n_M = 70$ destinations, on 1000 simulations. The number of LAN per node varies from 1 to 20. Class C IP subnetworks were randomly attributed to each LAN, which ensures that two IPs in the same LAN are close. There is no relation between two IPs of different LANs, even if these two LANs are connected to the same node. Since the IP are randomly chosen and uniformly distributed, the cost of the trees is 30 (since $n/n_M = 3$, three trees are generated. Without sorting, each tree spans all the 11 nodes of the Abilene topology, covering 10 edges; the

total cost of the three trees is therefore 30) for every simulation. Indeed, for each of the three trees, every node is represented at least once in the destination list and therefore, the eleven edges of the Abilene topology are used by each tree. When there are few LANs per node, sorting the destination list greatly reduces the cost of the trees. Indeed, some locality can be deduced: for example, a node that contains LAN having only high IP adresses is not represented in the destination list for the first tree.

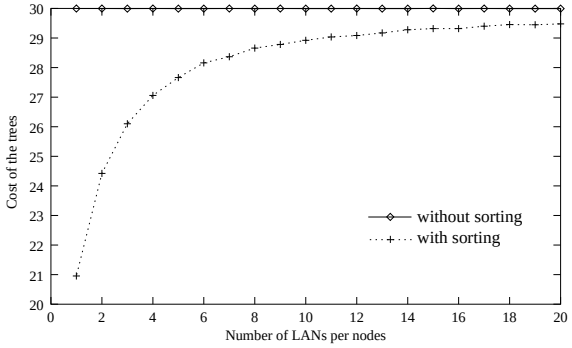

Fig. 4. The sorting performs better when there are few LANs per node.

Figure 5 shows the average cost of the trees on the same topology when the number of packets increases and when there are 5 LANs per node, for 1000 simulations. The number of packets varies from 1 to 20. As the number of packets increases, the cost of the trees without sorting grows linearly (each tree covers all nodes). The more packets are generated, the better the sorting performs. For 10 generated packets, the gain of sorting is close to 50%. Therefore, sorting the destination list is very important in the GXcast protocol.

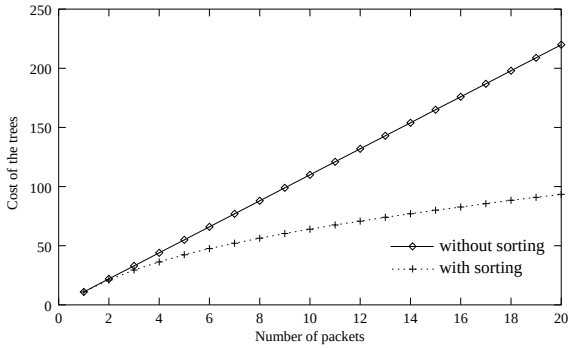

Fig. 5. The more packets are generated, the better the sorting performs.

V. Conclusions

The Xcast and Xcast+ protocols permit to manage efficiently a large number of small multicast groups. A major drawback of these protocols is that they are incapable to manage packet fragmentation. In addition, there is a limit for the multicast group size. In this paper, we proposed an extension to these protocols, named GXcast. GXcast solves the fragmentation problem of the Xcast protocol. We studied optimization criteria like sending less packets or minimizing the header processing time in routers. We showed that sorting the destination list using the IP adresses enhances the performance of the GXcast protocol. Finally, we deduced that the GXcast protocol could manage a large number of small size groups, with members in different sub-networks.

References

[1] D. Waitzman, C. Partridge, and S. Deering. Distance Vector Multicast Routing Protocol. IETF RFC 1075, 1988.

[2] J. Moy. Multicast Extensions to OSPF. IETF RFC 1584, 1994.

[3] J. Moy. MOSPF: Analysis and Experience. IETF RFC 1585, 1994.

[4] A. Adams J. Nicholas W. Siadak. Protocol Independent Multicast-Dense Mode (PIM-DM): Protocol Specification (Revised). IETF Internet Draft, 2003.

[5] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Weit. Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. IETF RFC 2362, 1998.

[6] M. Ramalho. Intra- and Inter-domain multicast routing protocols: A survey and taxonomy. *IEEE Communications Surveys and Tutorials*, 3(1):2–25, First Quarter 2000.

[7] Reliable Multicast Transport IETF Working Group Web Site. http://www.ietf.org/html.charters/rmt-charter.html, February 2003.

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

[9] D. Ooms. Taxonomy of Xcast/SGM proposals. IETF Internet draft, July 2000.

[10] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens. Explicit multicast (Xcast) basic specification. IETF Internet draft, January 2003.

[11] S. Myung-KI, K. Yong-Jin, P. Ki-Shik, and K. Sang-Ha. Explicit multicast extension (Xcast+) for efficient multicast packet delivery. *ETRI Journal*, 23(4), December 2001.

[12] B. Cain, S. Deering, and A. Thyagarajan. Internet Group Management Protocol, version 3. IETF RFC 3376, 2002.

[13] J. Postel. Internet Protocol. IETF RFC 791, 1981.

[14] S.Deering and R. Hinden. Internet Protocol, version 6 (IPv6) Specification. IETF RFC 2460, 1998.

[15] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. MIT Press/McGraw-Hill, 1990.

[16] Abilene Network. http://abilene.internet2.edu/.