

PAPER

A Fast and Efficient Explicit Multicast Routing Protocol*Mozafar BAG-MOHAMMADI^{†(a)} and Nasser YAZDANI^{†(b)}, *Nonmembers*

SUMMARY The state-oriented design of IP multicast may lead to the scalability problem, especially when there is a very large number of concurrent multicast groups in the network. Motivated by this problem, explicit multicast offers a stateless design using header space of multicast data packets. In this paper, we propose a novel stateless scheme called Linkcast that efficiently eliminates processing overhead of explicit multicast protocols. In Linkcast, the multicast sender encodes the tree listing its links in a proper way. The tree code is sent with every multicast data packet. Simulation results and experiments with real-trees show that Linkcast completely eliminates processing overhead of other explicit multicast protocols such as Xcast with comparable header size overhead.

key words: explicit multicast, small-size multicast

1. Introduction

Multicast can be considered as an efficient mechanism that decouples senders from receivers' information. Multicast sources can send their data packets on the network without worrying about information such as the number and location of receivers in the network. It also significantly enhances bandwidth utilization eliminating redundant packet replication. Traditional IP multicast routing protocols like DVMRP [18] and PIM-SM [13] require state maintenance in the on-tree routers, commonly known as Multicast Forwarding Table (MFT), in order to forward multicast packets correctly. This behavior is inconsistent with stateless philosophy of IP design, which declares that network devices (i.e. routers) should keep minimum state for routing purposes and complexities must be moved to the edges of the network. In IP multicast, state maintenance is performed in a per-group basis at on-tree routers. Therefore, the routers may easily run out of memory when there are very large number of low to moderate size multicast groups.

Small size multi-party applications such as video and audio conferencing, IP telephony and network games can not be well serviced at large scales by the current multicast model. Xcast [5], [6] and its variations Xcast+ [16], Bcast [1] and ERM [4] are designed to serve this class of applications in a scalable manner. Basic idea in explicit multicast is to provide sender with receivers' information such as

their IP addresses or their paths towards the sender. Then, the sender generates a proper header and attaches it to every multicast data packet. The header is processed by intermediate routers to forward multicast data packets. Therefore, group size is inherently limited to a small number in order to reduce processing overhead in intermediate routers and size overhead of generated header.

In Xcast and Xcast+, a list of destination IP addresses is sent with each multicast data packet. Hence, each router along the way forwards the packet based on the remaining destination addresses in the list. In Bcast and ERM, IP addresses of the receivers and branching points of multicast tree are sent with each packet. However, these protocols use unicast routing to forward multicast packets. They trade off the header size and processing power in favor of scalability and simplicity. The processing overhead consists of two parts: 1- number of required unicast lookups, 2-header decoding.

In [2], we presented Linkcast, which completely eliminates the unicast table lookups performed in forwarding process of other explicit multicast protocols. In Linkcast, the sender encodes the multicast distribution tree in terms of its link components. Interpreting the tree code, every on-tree router can easily determine next-hop link(s). Hence, the processing overhead is reduced to decoding a small portion of the multicast tree code rather than performing costly unicast lookups. Moreover, the tree code is generally small thanks to the basic assumption of explicit multicast about the size of multicast group. Also, simulation experiments for randomly generated networks and analysis of real multicast trees show that the header size overhead of Linkcast is comparable to both Bcast and Xcast.

In our first proposal for Linkcast [2], we introduced two encoding schemes. The first scheme, i.e. SBM (Sparse Branching Mode), is appropriate when there is a large number of relay links in the tree. The second one named DBM (Dense Branching Mode) was suitable for a tree with high average nodes degree. In this paper, we introduce a novel encoding scheme to further reduce the header size overhead of Linkcast. We analytically show that the code size of the new method is less than both previous methods. We have changed the multicast simulation method thoroughly to allow large-size network simulation. Also, we have evaluated performance of Linkcast for actual multicast tree based on real-data presented in [11] and [10].

In Section 2, we briefly review the related work. The key components of Linkcast design are discussed in Section

Manuscript received October 25, 2004.

Manuscript revised March 18, 2005.

[†]The authors are with Router Lab., University of Tehran, Tehran, Iran.

*A short and different version of this paper has been appeared in Networking 2004.

a) E-mail: mozafarb@ece.ut.ac.ir

b) E-mail: yazdani@ut.ac.ir

DOI: 10.1093/ietcom/e88-b.10.4000

3. Section 4 presents simulation results and Section 5 concludes the paper.

2. Related Work

Xcast [5], [6] was originally designed to overcome scalability and deployment problems of the current IP multicast routing protocols like PIM-SM [13] and DVMRP [18] for supporting very large number of low to moderate size multicast groups. In Xcast, multicast sender simply encodes the list of receiver IP addresses in a special header and sends it with all data packets. Each router along the way partitions the destinations' addresses into different sets based on their next hops. Then, for each set, it forwards a packet with appropriate Xcast header, which contains all destinations in that set. Therefore, the router should perform a unicast table lookup for every destination in the Xcast header of the packet. This processing overhead limits the number of supportable receivers to a very small number (around 10 [15]).

Xcast+ [16] introduces a simple modification to Xcast by combining it with the well-known IGMP (Internet Group Management Protocol) [9] protocol. The original Xcast does not define any mechanism for the receivers' join model for the sake of simplicity and generality. Hence, in Xcast, two or more receivers in the same LAN are treated as separate receivers. However, in Xcast+, they are represented by a LAN Designated Router (DR). This is achieved by adding the IGMP (Source, Group) join at receivers' side and sending join requests toward sender, and by encoding the address of DRs instead of the receivers' addresses in the Xcast+ header. Therefore, Xcast+ is more efficient than Xcast when some receivers reside in the same LAN.

Explicit Route Multicast (ERM) [4] is another explicit multicast scheme, which encodes the tree in terms of its final destinations and branching routers in each multicast data packet. Unlike Xcast in which destination nodes are end system hosts, ERM uses designated router directly attached to the receiver(s) as final destination. The information to build the delivery tree is acquired by collecting trace messages from the receivers. Bcast [1] is proposed as a modification to ERM. It benefits from smaller tree code while eliminating some noticeable design flaws of ERM. The main difference between Bcast and ERM is the place where tree encoding occurs. In Bcast, the sender host gathers tree information and generates the tree code(s). In contrast, ERM employs nearest ERM capable router to encode the delivery tree as well as sending multicast packet down the tree. Bcast and ERM efficiently reduce the number of unicast lookups that take place in Xcast and Xcast+ forwarding mechanism. Therefore, they can support larger group sizes due to their superior forwarding method.

Sender Initiated Multicast (SIM) [17] is another Xcast like scheme, which reduces the Xcast forwarding cost. SIM packets are forwarded between each SIM router pair in unicast. Basically, SIM has two forwarding modes: list mode and preset mode. In the list mode, the SIM sender always attaches the receivers' list to the packet. In the preset mode,

which is the prevalent SIM mode, the SIM sender periodically attaches the receivers' list to the packet. SIM capable routers construct an MFT-like table to forward the packet in the preset mode. This state maintenance is the main drawback of SIM, which severely limits its scalability.

Generalized Xcast (GXcast) [7] is basically intended to avoid packet fragmentation in Xcast. To cope with large set of receivers, GXcast regroups them into smaller sets and sends an Xcast packet to each set. Clearly, this approach is also applicable to other explicit multicast protocols with minor modifications.

Simple Explicit Multicast (SEM) [8] uses the receivers' list to construct the multicast distribution tree similar to SIM. SEM packets are also forwarded according to the unicast forwarding paradigm between SEM router pairs. The main difference between SEM and SIM is the position of SEM/SIM router pairs. An SEM router pair may consist of a branching point of the tree, the sender or a receiver. However, SIM router pairs can contain any on-tree router. SEM suffers from scalability issues due to the permanent state maintenance in branching routers of the multicast tree.

3. Linkcast

The main objective of the tree encoding method is to minimize the size of the generated code. Our new encoding method, which hereafter is called DMC (Dual Mode Coding) categorizes on-tree links into three types:

Member link: Incoming link of a receiver node (or a node that has a degree of 1 in the distribution tree). It is worth noting that by the term "receiver" we mean designated router directly attached to the receiver host. When end node of a member link receives a multicast data packet, it must deliver the packet to its local receiver(s).

Relay link: Incoming link of a node that has a degree of 2 in the distribution tree. The end node of such a link relays incoming multicast data packet onto the outgoing link, which is the next link ID in the DMC code.

Branching link: Incoming link of a node that has a degree of 3 or more in the distribution tree. The end of a branching link should find multiple next-hop links decoding the DMC code and forward a modified copy of the received packet on each link.

According to tree characteristic measurement in [11], [14], when multicast receivers are distributed sparsely in the network, the data delivery tree is likely to have a large number of relay and member links. Also, there are some multicast trees, such as binary trees, where branching links are dominant. However, since the size of explicit multicast groups are assumed to be small, we expect that sparse multicast trees be prevalent ones. DMC is designed and optimized to act better in the sparse mode. Moreover, our analytical results indicate that DMC performs better than both previous encoding methods i.e. DBM and SBM.

3.1 Tree Encoding

DMC scheme assigns a link ID to each on-tree link according to aforementioned link classification. The link ID consists of two parts: 1- Global Part: the first n ($n = 2$ in DMC) bits determine the type of link 2- Local Part: remaining $(8 - n)$ bits is a local ID which is assigned by on-tree routers during the join process (see Sect. 3.2.1). As described earlier, outgoing link of a relay link is placed just after the relay link in the tree code. Therefore, the end node of a relay link just relays the received packet on the next link in the code. If during decoding process, a router found that the next-hop link is a member link, it disables Linkcast header to allow normal delivery of the multicast packet by the member node connected to the other end of the member link. The member node will deliver multicast data packets to its local receiver(s) using IGMP protocol.

The most critical part of DMC scheme is coding of a branching link. End node of a branching link must be able to figure out the number of outgoing links and their IDs. Suppose that the end node of a branching link has m outgoing member links and n outgoing relay or branching links such that $m \geq 0, n \geq 0$ and $m + n \geq 2$. For such a branching link, DMC first lists all m member links followed by $n - 1$ pointers to the corresponding relay or branching links in the code. Finally, it places the last non-member link. Doing so, the end node will be able to determine outgoing links if there is at least a relay or a branching link (panel a,b,c of Fig. 1). In rare cases, if all outgoing links of the end node are member links, DMC must put a dummy pointer just before the last member link. For example, if it only has m (by definition $m \geq 2$) member links, Linkcast first places $m - 1$ member links followed by a pointer to the last member link. (panel d of Fig. 1). However, our approach for encoding a branching link implies that the link pointers must be distinguishable from the link IDs around them (see Sect. 3.1.2).

The tree code has a general pointer (P_g), which always points to the current link in the code. When a router receives a Linkcast packet, it examines the value of P_g . Then, it finds the next link(s) by interpreting the tree code. Finally,

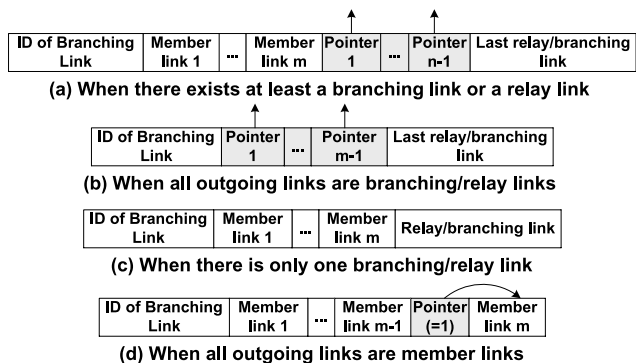


Fig. 1 Different possibilities for encoding of outgoing links of a branching link.

it modifies P_g value accordingly and forwards the packet. If P_g points to a link pointer, then something wrong has happened. Therefore, the receiving node will discard the received packet and inform the source about it. We can use another method to discover the current link in the tree code. In this method, the router removes the unnecessary part of the tree code when forwarding Linkcast packet. So, when a router receives a Linkcast packet, it finds the next link(s) by interpreting the tree code from the beginning. It then removes the unnecessary part of the tree code and forwards the packet. This eliminates the need for storing P_g value in the tree code.

3.1.1 An Example

We explain the new encoding method through an example. The example tree and its corresponding code are shown in Fig. 2. The number on each link is the link ID and is determined as discussed in Sect. 3.2. Each link pointer is denoted by pi_j , where j is the index of the link pointer in the tree code. The incoming link of a relay node has only one outgoing link. For example, the next link of $b1$ is $c1$. For every branching node that has two or more non-member links, Linkcast chooses one of its non-member links and continues the tree code from it. It then comes back to other non-member links. For example, for link $a1$ (or equivalently node b), we first encode $b1$ sub-branch completely and then turn back to $b2$ case. Link $c1$ is a branching link that its outgoing links are all member links. Here, we placed a dummy pointer of size 1 (pi_2) between $e1$ and $e2$. If we assume B ($=8$) bits is used to encode each link ID or pointer, the size of the tree code in that example will be $19B$.

3.1.2 Coding Details

We use 8 bits to represent link IDs or link pointers in the tree code. For a link ID, the first two bits of the link ID determine the type of the link. The remaining six bits determine the value of the link ID. If the ID is of the form $110*****$, then

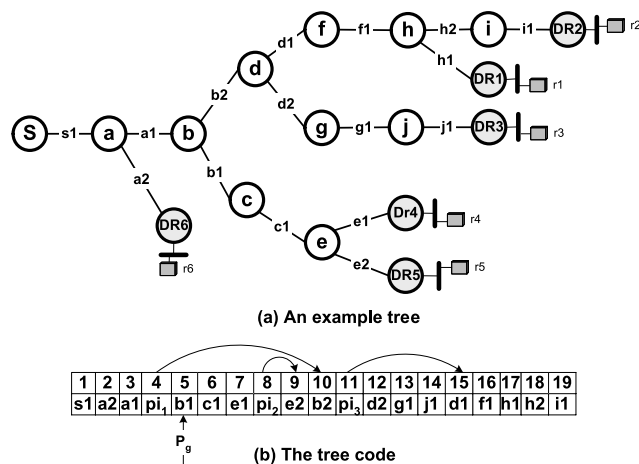


Fig. 2 An example tree and the resulting tree code.

Table 1 Link types/pointer semantic in Linkcast.

First two or three bits	Next six or five bits	Description	Action
00: member link	Link ID	-	Deliver the packet to local receiver(s)
01: relay link	Link ID	The next byte determines the ID of next link.	Forward the packet on the next link.*
10: branching link	Link ID	Next bytes are member link(s) and/or pointer to non-member link(s) and/or a non-member link.	Forward the packet on the next links.*
110: link pointer	Pointer value	-	Forward the packet on the pointed link.
111: extension	First five bits of a link ID or a pointer	The next byte determines type of ID (link ID or pointer) and remaining bits of ID.	The action is identified by next byte, which is one of the above actions.

it is a pointer to another link ID in the code and the remaining five bits determine the value of the pointer. Finally, value 111***** indicates that the ID is extended. In that case, next byte in the code determines the type of ID. Therefore, the link ID and the pointer can be as large as 2^{5+6} and 2^{5+5} respectively. Table 1 summarizes this discussion. Here, we have assumed that an on-tree node has received a Linkcast packet and wants to discover the next steps based on the value of the current link ID.

3.2 More Protocol Details

3.2.1 Gathering the Tree Information

The sender collects the tree information from the receivers' *Join* messages. Each new member sends a *Join* message toward the sender when it joins the multicast session. All routers on the path examine the *Join* message and append the incoming link ID, the outgoing link ID and their IP addresses to the message. Having the path information from receivers to the source, the source can construct the reverse shortest path tree. Our previous simulations in [2] showed that using reverse path instead of direct path has negligible impact on the size of the tree code.

The receivers must repeat their *Join* messages periodically in order to refresh their state in the sender. Thus, the sender can repair the multicast distribution tree against temporary route changes. If the sender misses three consecutive *Join* messages from a receiver, it will remove the receiver from the tree code. A receiver can also immediately depart the multicast session by sending a *Leave* message to the sender.

3.2.2 Multi-Access Links

In a multi-access link, the broadcast nature of the link is a source of ambiguity. Since the end node of the link is not unique, it is not possible to determine the end node and the next link(s) when decoding the tree code. To solve this ambiguity, we decompose a multi-access link into $n * (n - 1)/2$ virtual links, where n is the number of routers on the link. Each virtual link has a unique ID and virtually connects two routers. All routers must know the ID of each virtual link. Therefore, the routers must run a simple protocol to agree on the ID of the virtual links. Figure 3 shows the virtual links resulted from the decomposition process of a multi-access link for router a .

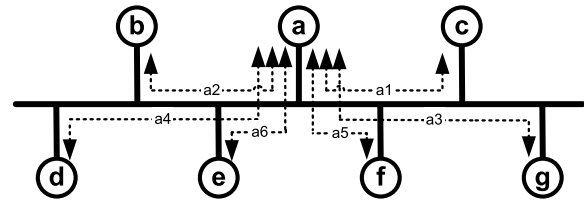


Fig. 3 Virtual links resulted from decomposition of multi-access link for node a .

Table 2 Structure of Linkcast header.

1	3	4	8	9	15	16	31
VERSION		NEXT_PROT		SIZE		POINTER	
ENCODED_TREE							

3.2.3 Linkcast Header

The Linkcast header is inserted between the IP header and the transport layer header. In addition, the *Protocol* field of the IP header must be set to a pre-known value namely LINKAST_PROT. This allows routers to easily distinguish Linkcast packets among others. The Linkcast header for IPv4 is depicted in Table 2. It is composed of two parts: a fixed part (first 4 octets) and a variable length part. The NEXT_PROT field specifies the protocol of the subsequent header (transport layer header). The SIZE field indicates the length of the Linkcast header in 4-octet words. This field forces an upper bound on the size of the Linkcast header, which is 2^8 words (4-octets). The POINTER field points to the current link ID in the code. The pointer was introduced beforehand in Sect. 3.1 as P_g . In addition, it is possible to identify the current link (node) by removing unnecessary parts of the code. If so, the value of POINTER field must be NULL. The ENCODED_TREE field is filled as discussed earlier.

3.3 Analysis

3.3.1 Code Size

The DMC code consists of all link IDs and some link pointers in which the number of link pointers is proportional to the number of non-member children of all branching nodes. Suppose that we use 1 byte for encoding every link or pointer in the tree. Therefore, we can obtain the total size of DMC tree code as follows. In these equations, N_M ,

N_R and N_B are the number of member links, relay links and branching links respectively. n_i and m_i are the number of non-member links and member links of the branching node i respectively:

$$DMC = N_M + N_R + N_B + \sum_{i=1}^{N_B} f(n_i - 1) \quad (1)$$

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 1 & \text{if } x = -1 \end{cases} \quad (2)$$

Furthermore, suppose that all branching nodes have at least one non-member link ($n_i \geq 1$ for all branching links). We can rewrite Eq. (1) as:

$$DMC = N_M + N_R + \sum_{i=1}^{N_B} n_i \quad (3)$$

We also analyzed the previous encoding methods as proposed in [2]. In the SBM (Sparse Branching Mode), each member link or relay link is determined by one byte. For a branching link, the branching factor of the end node must be stored in the tree code. In addition, SBM stores the ID of the branching link and a pointer to every outgoing branch of the branching link. Therefore, the SBM code size is:

$$SBM = N_M + N_R + 2N_B + \sum_{i=1}^{N_B} (n_i + m_i) \quad (4)$$

In the DBM (Dense Branching Mode), each non-member link has a pointer, which points to the end node of the link. Therefore, DBM stores two bytes for each non-member link. In addition, DBM stores $n - 1$ pointers to the non-member outgoing links of each branching node. The tree code size of DBM is determined as follow:

$$\begin{aligned} DBM &= N_M + 2N_R + 2N_B - 1 + \sum_{i=1}^{N_B} (n_i - 1) \\ &= N_M + 2N_R + N_B - 1 + \sum_{i=1}^{N_B} n_i \end{aligned} \quad (5)$$

The result indicates that DMC always results in smaller tree code size. As one can see, SBM is more efficient than DBM when the number of relay nodes is higher than the number of branching nodes.

3.3.2 Chuang-Sirbu Scaling Law

Chuang and Sirbu discovered that over a wide range of networks, multicast tree cost is dependent only on the number of receivers [12]. They counted total number of links that make up the multicast distribution tree (L_m). Then, they defined the multicast cost as the ratio between L_m and the average path length in the network (L_u). The simulation results for both real and generated topologies showed that the multicast tree cost obeys a power law. They provided the following formula:

$$L_m/L_u = m^{0.8} \quad (6)$$

Where, m is the number of multicast receivers. The Chuang-Sirbu formula states that the cost of multicast tree is more reasonable when the number of receivers increases. The multicast cost is relatively high for primary receivers. The multicast tree cost per member is sharply reduced as the number of receivers increases. By definition, the tree cost is the normalized number of links in the multicast distribution tree. Therefore, Chuang-Sirbu scaling law predicts that the branch length is probably shorter for new receivers as the number of receivers increases, i.e., the likelihood of choosing average path length branches decreases for new receivers.

Since the size of Linkcast header is dependent on the number of multicast links, Linkcast header size obeys the same scaling law as the multicast tree cost. Therefore, the Linkcast header size is more reasonable for larger groups. It is worth noting that by word ‘‘larger group’’ here we mean explicit multicast groups of sizes around 50. The tree branch for new receivers probably consists of fewer links compared to tree branches of previously joined members. Simulation results also show that the Linkcast header size grows slowly with group size. We can use Chuang-Sirbu scaling law to estimate the Linkcast header size for any group size. First, the number of multicast links can easily be derived from Eq. (3) as $L_u * m^{0.8}$. Therefore, Linkcast header size (in Bytes) can be estimated as:

$$L_u * C * m_b^{0.8} \quad (7)$$

Where, C and m_b are Linkcast coding overhead and the number of receivers in largest source branch respectively.

3.4 Discussion

3.4.1 IPv6

In IPv6, the header size overhead of all explicit multicast protocols except Linkcast increases due to the length of IPv6 addresses. Therefore, they must be fully re-examined in case of the IPv6 extension. For those protocols, the header size is multiplied by 4, which in turn limits the number of supportable receivers. On the other hand, since Linkcast uses link IDs to encode the multicast distribution tree, it naturally has no problem with IPv6. This fact combined with Chuang-Sirbu scaling law makes Linkcast a superior candidate for supporting larger groups.

3.4.2 Source Branching

Since the multicast sender generates the encoded tree, it is possible to produce different codes for each sub-tree rooted at the sender when the sender itself is a branching point. This significantly reduces the size of the encoded tree. Bcast has the same property as well. This allows them to support larger number of receivers compared to Xcast+. It is worth noting that Eqs. (1) through (6) in Sect. 3.3 are applied to the largest sub-tree rooted at the sender.

3.4.3 Mapping Table

As stated earlier when a Linkcast-capable router process a *Join* message, it attaches the incoming and outgoing link IDs to the message. The assigned Link IDs are not the same as physical addresses of the corresponding links. Therefore, the router needs to maintain a *mapping* table, which shows the correspondence between physical link addresses and link IDs. Later when the router decides to forward a received multicast packet on an outgoing link, it uses the outgoing Link ID as a key to find the physical address of the link in the mapping table. The mapping table contains all physical link addresses and their corresponding link IDs. The size of this table is orders of magnitude smaller than a typical unicast forwarding table. Furthermore, the table is identical for all multicast groups or trees that pass the router and it is independent from the sender identity or the multicast group address. The table could be filled by the router in the start-up process.

3.4.4 Link Failure

The failure of an on-tree link partitions the tree and disturbs the multicast service for a subset of receivers. In that case, the Linkcast code is erroneous for all receivers behind the failed link. Since each receiver repeats its *Join* message periodically, the failure effects will be eliminated at most after one period for affected receivers. The router that is directly attached to the failed link could detect the link failure more quickly. When a router receives a multicast data packet, it normally interprets Linkcast header to find the ID of the next link. It can check the status of the next link before forwarding the packet. In case of a link failure, the router can report the failure to the sender. After reception of failure report, the sender excludes the affected receivers from the tree code. Also, it could ask the excluded receivers to join the tree again.

4. Performance Evaluation

To evaluate the effectiveness of the proposed mechanism, we have compared the Linkcast header overhead with Xcast+, ERM and Bcast. We chose Xcast+ for our comparisons since Xcast+ is slightly more efficient than Xcast. We performed two sets of experiments, one for random networks and another for real trees obtained from Internet [10]. Although it is possible to calculate header size overhead directly looking at network topology, we have implemented Linkcast, Xcast+, ERM and Bcast in “myns” [3] packet level simulator to verify correctness of protocols massaging, encoding and decoding. It also helps us to collect the required information easily.

The network topologies used in our simulations are generated based on Transit-Stub graph model, using GT-ITM topology generator [19]. We fixed the network size at 10100 and performed simulations with various group sizes

ranging from 5 to 50. The average node degree is fixed approximately at 3.5. Each point in each graph is resulted from 5 simulation runs for 10 different random topologies. This creates 50 different simulation runs for each point. The identities of the group members, i.e. the sender and receivers, are selected randomly in each simulation run. We chose a single node randomly to act as a multicast sender. Then members will join the multicast session of the sender at different random times. Having path information of all receivers, the sender encodes multicast tree using ERM, Bcast or Linkcast encoding schemes.

In [11], many different-size real-trees is generated executing trace-route command from different receivers sites toward a single and fixed sender. These real trees are available at [10]. We used the largest set, which has 1950 receivers. Then, we chose smaller subset of the receivers and reconstructed the multicast tree and measured its performance. We repeated the process for various group sizes between 5 and 50. Also, for each group size, we repeated the process 100 times. The identity of receivers is selected randomly in each run.

4.1 Header Size Comparison

We experimented with two different encoding methods of ERM as discussed in [5]. ERM-1 uses plain unicast tunnelling between ERM router pairs, while ERM-2 uses minimal encapsulation within the IP header. For Xcast+, we also presented the result for an artificial version of Xcast+ named Xcast+-M. We augmented Xcast+ with route information in order to enable Xcast+ to gain benefit from the source branching possibility. Figure 4 shows the header size overhead of the four methods in randomly generated graphs. As can be seen, the header size for Xcast+-M is slightly better than Xcast+ in our simulations. However, source branching was fairly rare event both in the generated random networks and real trees.

As expected, the header size increases with the group

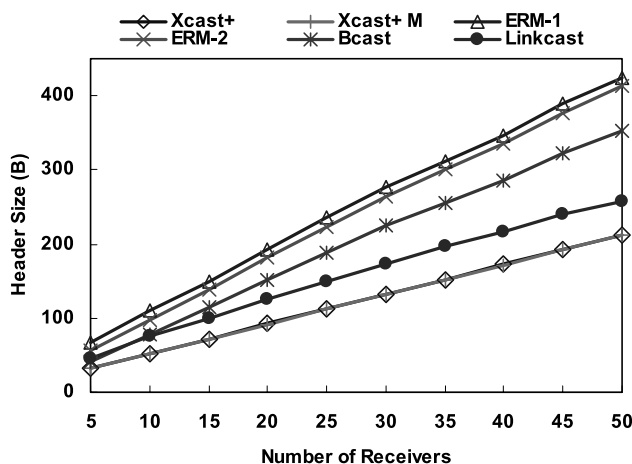


Fig. 4 The header size vs. group size in random topology graphs of size 10100.

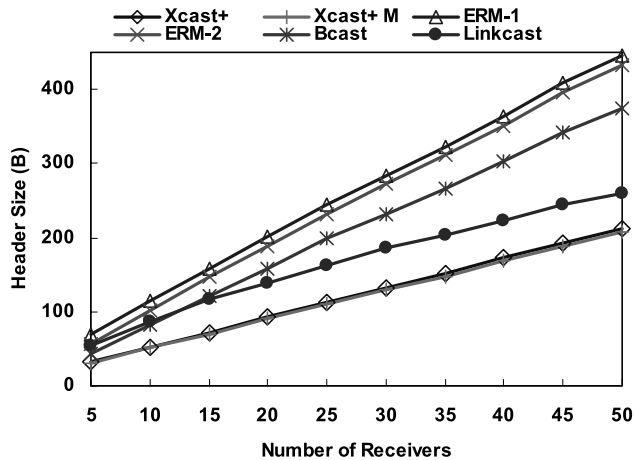


Fig. 5 The header size vs. group size in real trees generated by trace-route tool.

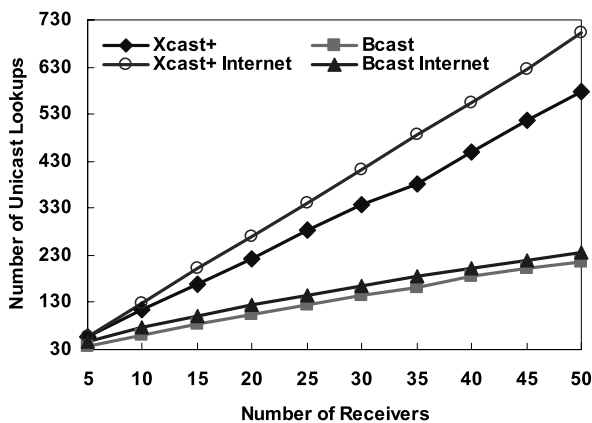


Fig. 6 The number of required unicast lookups changing the group size from 5 to 50.

size. Linkcast header size is larger than Xcast+ for all groups sizes. Also, it is always smaller than ERM and Bcast header sizes except for multicast groups of size 5 and 10. The relative overhead of Linkcast to Xcast+ decreases when the group size increases as predicted by Chuang-Sirbu scaling law. The Linkcast relative overhead is 1.44 for groups of size 5 and decreased toward 1.2 for groups of size 50. Figure 5 shows the header size comparison for real trees. As the figure suggests, the Linkcast header size overhead is less than Bcast and ERM. Moreover, simulation results almost fully conform to the real network experiments.

4.2 Processing Overhead

Figure 6 shows the comparison between the required unicast lookups in Bcast (ERM) and Xcast+. In this figure, "Xcast+ Internet" and "Bcast Internet" stand for Xcast+ and Bcast+ simulation results in the Internet respectively (real trees). The number of performed unicast lookups in the Internet experiments is more due to the higher average path length. As the figure suggests, Bcast (ERM) reduces the number of required unicast lookups efficiently. It is worth noting that

table lookups in explicit multicast methods are more expensive than unicast ones, because the multicast engine issues them to the unicast engine. Linkcast processing overhead is the smallest among all methods and it does not require any type of unicast table lookups in its forwarding plane. Instead, it looks up the next link ID in much smaller *mapping* table as stated in last section.

5. Conclusion

Traditional IP multicast protocols can not efficiently support very large number of any size multicast groups mainly due to their state-full design. Xcast is proposed to support very large number of small size multicast groups (around 10 [15]). The main drawback of Xcast is presence of excessive lookups in its multicast data forwarding. Trying to overcome the lookups problem, ERM and Bcast support larger group sizes than Xcast. Although Xcast, ERM and Bcast benefit from stateless design, they have two main difficulties to support moderate size multicast groups. First, their header size grows rapidly. Second, they need more unicast lookups in intermediate on-tree nodes when the number of multicast members increase. Linkcast solves the header size problem of ERM and Bcast without having to perform any form of table lookup. We believe that Linkcast is more appropriate than other explicit multicast protocols in supporting huge number of moderate to fairly large size multicast groups (less than 50). Simulation result shows that Linkcast header size is more reasonable when supporting larger multicast groups. As a main weakness, if some routers on the path between a receiver and the sender do not support Linkcast, the receiver could not be serviced by Linkcast. However, the sender still can use the normal unicast delivery scheme for such a receiver. The same is true for Xcast and Xcast+. On the other hand, ERM and Bcast are gradually deployable, i.e. they could establish the multicast service even in the presence of ERM/Bcast unaware routers but with possible performance degradation.

References

- [1] M. Bag-Mohammadi, S. Samadian-Barzoki, and N. Yazdani, "On the efficiency of explicit multicast routing protocols," Proc. IEEE ISCC 2005, pp.679-685, Cartagena, Spain, June 2005.
- [2] M. Bag-Mohammadi, S. Samadian-Barzoki, and N. Yazdani, "Linkcast: Fast and scalable multicast routing protocol," Proc. Networking 2004, pp.1282-1287, Athens, Greece, 2004.
- [3] S. Banerjee, myns Simulator, <http://www.cs.umd.edu/~suman/research/myns/index.html>
- [4] J. Bion, D. Farinacci, M. Shand, and A. Tweedly, "Explicit route multicast (ERM)," draft-shand-erm-00.txt, June 2000.
- [5] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens, "Explicit multicast (Xcast) basic specification," IETF Internet-Draft, draft-ooms-xcast-basic-spec-07.txt, July 2005.
- [6] R. Boivie, N. Feldman, and C. Metz, "Small group multicast: A new solution for multicasting on the Internet," Internet Computing, vol.4, no.3, pp.75-79, May/June 2000.
- [7] A. Boudani, A. Guitton, and B. Cousin. "GXcast: Generalized explicit multicast routing protocols," Proc. IEEE ISCC 2004, pp.1012-1017, Alexandria, Egypt, 2004.

- [8] A. Boudani and B. Cousin, "SEM: A new small group multicast routing protocol," Proc. IEEE ICT 2003, pp.450–455, Tahiti, Feb. 2003.
- [9] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet group management protocol, version 3" RFC 3376, Oct. 2002.
- [10] R. Chalmers and K. Almeroth, <http://www.nmsl.cs.ucsb.edu/mwalk/>
- [11] R. Chalmers and K. Almeroth, "On the topology of multicast trees," IEEE/ACM Trans. Netw., vol.11, no.1, pp.153–165, Feb. 2003.
- [12] J. Chuang and M. Sirbu, "Pricing multicast communication: A cost-based approach," Proc. INET'98, Geneva, Switzerland, July 1998.
- [13] S. Deering, D.L. Estrin, D. Farinacci, V. Jacobson, C.G. Liu, and L. Wei, "The PIM architecture for wide-area multicast routing," IEEE/ACM Trans. Netw., vol.4, no.2, pp.153–162, April 1996.
- [14] J. Pansiot and D. Grad, "On routes and multicast trees in the Internet," ACM Comput. Commun. Rev., vol.28, no.1, pp.41–50, Jan. 1998.
- [15] O. Paridaens and D. Ooms, "Security framework for explicit multicast," draft-paridaens-xcast-sec-framework-01.txt, Nov. 2000.
- [16] M.K. Shin, Y.J Kim, K.S Park, and S.H Kim, "Explicit multicast extension (Xcast+) for efficient multicast packet delivery," ETRI J., vol.23, no.4, pp.202–204, Dec. 2001.
- [17] V. Visoottiviseth, H. Kido, Y. Kadobayashi, and S. Yamaguchi, "Sender-initiated multicast forwarding scheme," Proc. IEEE ICT2003, pp.334–339, Tahiti, Feb. 2003.
- [18] D. Waitzman, C. Partridge, and S. Deering, "Distance vector multicast routing protocol," RFC 1075, Nov. 1988.
- [19] E.W. Zegura, K. Calvert, and S. Bhattacharjee, "How to model an Internetwork," Proc. IEEE Infocom '96, pp.594–602, San Francisco, CA, 1996.



Mozafar Bag-Mohammadi holds a B.S. degree in Electronic Engineering from Sharif University of Technology. He received the M.S. degree in Digital Electronic Design from University of Tehran in 2000. He is currently working toward the Ph.D. degree at the University of Tehran, Iran. His main research interests include multicast routing protocols, explicit multicast, routing protocols, multicast support in MPLS networks, and Internet.



Nasser Yazdani got his B.S. degree in Computer Engineering from Sharif University of Technology, Tehran, Iran. He worked in Iran Telecommunication Research Center (ITRC) as a researcher and developer for few years. To pursue his education, he entered to Case Western Reserve Univ., Cleveland, Ohio, USA, later and graduated as a Ph.D. in computer science and Engineering. Then, Dr. Yazdani worked in different companies and research institutes in USA. He joined the ECE Dept. of Univ. of

Tehran, Tehran, Iran, as an assistant professor at Sept. 2000. His research interest includes Networking, packet switching, access methods, Operating Systems and Database Systems.