An Efficient Routing Protocol with Group Management for Peer-to-Peer Multicast Applications

Ning Wang & George Pavlou
Centre for Communication Systems Research
University of Surrey
United Kingdom
{N.Wang, G.Pavlou}@eim.surrey.ac.uk

ABSTRACT

Despite the relevant efforts on IP multicast communication [4], not much progress has been made towards commercial deployment of applications that abide to the service model as originally proposed. This is due to the complicated architecture of IP multicast and, more important, its "open group" management strategy. Single-Source Multicast [8] has been recently proposed as a closed group service model for one-to-many applications such as Internet TV/radio, data distribution, etc. We notice that there still exist though many other applications based on interactive group communication, such as multi-party videoconferencing, distance learning, Collaborative Virtual Environment (*CVE*), etc. that are not well supported by the existing service models. A key aspect of such applications is that all the group members act both as senders and receivers simultaneously. In this paper we first propose a new multicast service model for peer-to-peer interactive applications with strict management mechanisms for controlling participation in a service session. An efficient and scalable routing protocol (*PROMPT* - PROtocol for Multicast in Peer-to-peer Transmissions) is introduced to support this service model on the Internet.

1. INTRODUCTION

Although some IP multicast applications have been available on the experimental Multicast Backbone (*MBone*) for several years, large-scale development has not been achieved until now. There are several possible reasons for this situation. First, the service model of IP multicast cannot apply to the majority of applications that are of immediate interest to people, e.g., Internet TV/radio, multimedia interactive conferencing systems etc. One basic characteristic of this model is its "open group" management strategy: an information source can send data to any multicast group without any control mechanisms, which conflicts with the nature of most multicast applications. The key problem is though that this may have disruptive effects to both group members and the network itself. Second, group management is not strong enough to control both senders and receivers. *IGMP* [7] is used to manage group members when they join or leave the session but there are no control mechanisms to avoid receiving data from particular information sources or prevent particular receivers to receive sensitive information. The third disadvantage comes from the scalability problem of multicast routing in large-scale networks such as the Internet: few routing protocols can work equally well in an inter-domain environment. It is very difficult to devise one or a set of protocols that can simultaneously satisfy requirements such as efficient address allocation, interdomain source discovery and core location (if shared tree is considered), etc.

Single-Source Multicast (*SSM*) [8] has been recently proposed as an alternative service model for IP multicast. From a commercial point of view, the new service model aims at one-to-many communications such as Internet TV/radio where a well-known source delivers data to all subscribers. From a technical point of view, this new model is relatively easy to support. Each group is identified by an address tuple (*S, G*) where *S* is the unicast address of information source and *G* is the "channel" destination address. A single multicast tree is built rooted at the source for delivering data to all the subscribers, so that problems such as address allocation and source discovery are not obstacles in its deployment. However, specifically designed for single source communication, *SSM* cannot cope efficiently with applications in which data streams are frequently flowing between peering hosts.

It should be noted that there still exist many types of applications that are based on interactive communications between multiple parties. Typical examples include videoconferencing, distance learning, *CVE, etc.* One significant characteristic of such applications is that each group member is both an information sender and a receiver at the same time. For example, in a multiparty videoconference, each participant is allowed to exchange messages with others in the same session and hence one cannot distinguish a *major* information source. We have devised a new service model for this type of interactive applications which we name *Peer-to-Peer Multicast* (*PPM*). Compared with the "open group" IP multicast service model which does not have control over senders, Peer-to-Peer Multicast can provide much sophisticated control mechanisms over group members due to the fact that both the sender and the receiver are integrated into a single entity. Nevertheless, *PPM* does not attempt to modify or

take the place of current IP multicast but rather tries to provide an easy and efficient solution specifically tailored for such interactive applications that rely on group communication.

2. PEER-TO-PEER MULTICAST SERVICE MODEL

2.1 Motivation

As mentioned above, Single-Source Multicast (*SSM*) has been proposed as an alternative service model for IP multicast. However, such a model cannot apply to all the current multicast-oriented applications. Although mechanisms have been provided in *EXPRESS* to deal with multiple senders, the protocol has been originally designed for a single source and hence results in many inefficiencies when multiple senders are concerned: data traffic from other potential sources always has to pass through the initial single source, resulting in traffic concentration, increased delays, etc. Considering applications such as videoconferencing and distance learning, every participant exchanges data with each other so that there is not a main information source. In such applications each group member acts both as an information sender and a receiver, and furthermore each message from a participant is meaningful to all the other group members and at the same time every participant is typically interested in data from all the other senders in the group. Another significant characteristic of this type of applications is that the communication mode is in reality "few-to-few" compared with "one-to-many" applications such as Internet TV/radio that could have up to millions of subscribers simultaneously. Based on this idea, we introduce *Peer-to-Peer Multicast* (*PPM*) Service Model for this type of interactive multicast applications. Being closed-group, *PPM* also requires control mechanisms such as registration for new members. Fig. 1 illustrates the difference in the service models of IP multicast, *SSM* and *PPM*.

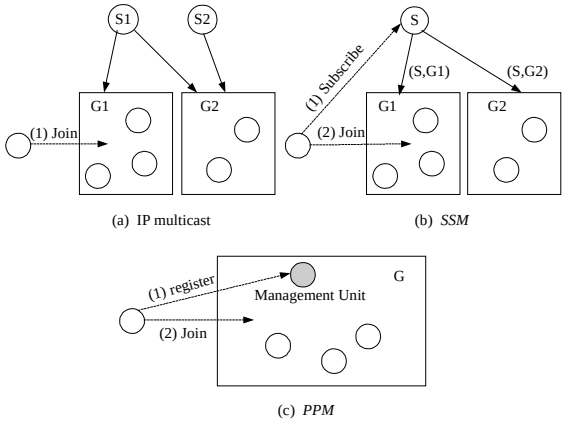

Fig. 1 Service model of IP multicast, *SSM* and *PPM*

2.2 Implementation Issues

Being a *closed group* service model, Peer-to-Peer Multicast requires that each group member be managed in a centralized fashion. Since every *session* has an initiator or organizer, e.g., there is always a chairman for organizing a videoconference, a lecturer is normally responsible for starting a session in distance learning, etc., such an organizer will be ideal to manage the group and we define it as *session manager* (corresponding to the "group management unit" in Fig. 1(c)). Each host that wants to join a session will have first to register with the session manager by providing some type of security key. After registration, the host becomes a *session member* and will be able to send/receive messages to/from other members. It should be noted that the session manager is also a session member with the common class D session address but it has additional functions for registering and managing other members.

Bi-directional data flow and centralized session management led us to decide that building a shared tree for delivering data between session members is the most efficient solution. The core of the tree should be located at the first hop router of the session manager so that centralized management becomes easy. This core is also known as the *Designated Router* (*DR*) of the session manager. Besides the session address, the session manager will also announce the "core" address for registering other members. The potential session members will have to register to the core address. When group members send messages to each other, they are identified by their individual unicast address so that all the other participants are able to know where the data come from. On the other hand, all of the session members use the common class D address to receive data from other peers. This aspect still abides to the fundamental architecture of classical IP multicast.

2.3 Advantages

From a group management point of view, additional level of control and subsequently security is achieved in comparison to the IP multicast service model. Both senders and receivers are under centralized control by the session manager. Such control mechanism is relatively easy to achieve since a sender and receiver is integrated into one *member* entity. *PPM*, being closed group, prevents external data sources from sending packets to members of a session, so both customers and service providers will be able to obtain associated benefits. Session members benefit since they will never receive packets from irrelevant external sources while ISPs benefit since their network itself will not be unnecessarily congested by multicast flows from irrelevant and potentially malicious sources.

From an implementation point of view, many difficult problems that exist in the IP multicast service model can be avoided. First, since all the information of a session, including the session and core address, can be obtained in advance, the problem of source discovery does no longer exist in

inter-domain routing. Session members can directly locate the session manager even if the latter is in a different domain. Moreover, every time a new member joins the session, the session manager will inform current members by sending a notification message so that all the members know all the participants in the session. Second, since all the senders are session members, no IP-in–IP encapsulation is needed.

3. *PROMPT*: PROTOCOL FOR MULTICAST IN PEER-TO-PEER TRANSMISSIONS

3.1 Protocol Overview

As we mentioned previously, in order to natively deliver messages between session members, a bi-directional shared tree (more precisely, bi-directional data flow, unidirectional control flow) is to be built rooted at the first hop router of the session manager. Since all the participants share the same class D session address, in order to identify each of them, their individual unicast address is used for identification when they are in the role of an information source and send data to the session address. *PROMPT* has a strict membership management mechanism: packets from any other non-member sender will be discarded once they arrive at the on-tree routers so that each member will not receive any data that are of no interest. Moreover, the session manager is able to control the behavior of all the members, e.g., preventing some of them from sending or receiving data.

In *PROMPT*, in order to detect the data from non-member sources and control the behavior of all the session members, each router has the entry for the session manager and all the *downstream* session members. The basic format of the entry is: <*SA*, *UA*, *Interface*, *Status*> where *SA* denotes class D session address; *UA* is the unicast address of each session member so that they can be identified when sending data to other peers; *Interface* indicates the one to which this session member is attached; and *Status* is used to control the behavior of the members. Possible values of *Status* include: *DISABLED*, *RECEIVE-ONLY*, *SEND-ONLY* and *ENABLED*.

3.2 Basic Descriptions

The basic working procedure of *PROMPT* can be described in the following four steps:

Step 1: Initialization. To initialize the session, the session manager will first publicly announce information including the core and session addresses. If any hosts are interested in the session, they will respond and apply for session membership to the core, which will then forward the application to the session manager itself. Upon receiving such applications the session manager will record the unicast address of each applicant and individually assign to them a security key for further registration. All this could be done by email or some other "out-of-band" mechanism in advance. How the multicast key is more efficiently delivered to each of session members is specified in [2] and is out of the scope of this paper. Initially, the session manager enters the group by obtaining the session address and performing a self-registration to the core, i.e. its first hop router. Once the router receives this self-registration packet, it will create an entry for the session manager at the interface from which the self-registration packet is received; the value of *Status* in the entry is set to *ENABLED* and will remain until the session is finished.

Step 2: Registration. After receiving all the necessary information, the potential session members will be able to register with the manager by sending registration packets to the core, which will then forward the packet to the session manager itself. Once an intermediate router receives a registration packet, if it does not yet have an interface with state for this session, it will create the session state at the interface that is used to deliver unicast messages to the core; we call this interface the *upstream interface*. (A certain interface has the state of a session if the router contains at least one entry, which indicates that this interface is attached to a session member.) The corresponding entry for the session manager is created with initial value of *Status* being *DISABLED*, and the unicast address of the session manager is left unfilled because the new member may not know the unicast address of the manager until it has received the join-notification from it. At the same time the interface from which the registration packet is received is known as the *downstream interface*, and the router will create an entry for this new session member with the value of *Status* set to *DISABLED* so that the new member is kept in a pending state. If an on-tree router already having the state of the session receives an additional registration packet, it uniquely forwards this packet on the existing upstream interface towards the session manager after the entry for the new member is created. Once the session manager receives the registration packet, it will check its corresponding source address (i.e., the value of *UA*) and security key, and then send a join-notification packet not only back to the new member but also to all the other downstream on-tree routers with active session members, so that all the current members will be able to know that a new host has joined the session. The major difference between *PROMPT* and *CBT* [1] in traveling route of registration and corresponding notification packets is shown in Fig. 2. The join-notification packet should also contain a separate list of the *UA*s for all the current session members so that the new member is able to know who is currently in the session. When a router receives a join-notification packet from its upstream interface, it will first check whether it has the entry with pending state for this new session member. If there exists such an entry, the router will reset the corresponding value of *Status* to *ENABLED* and then forward this notification to all its

downstream interfaces with the state of the session. Otherwise if the router cannot find any entry that matches the value of *UA* contained in the notification packet, it will simply forward the packet to all its downstream routers with session state. If this is the first notification packet from the session manager (i.e., the *Status* value in the manager's entry is *DISABLED*), the router will copy the unicast address of the session manager into to its entry from the notification packet and then reset its *Status* to *ENABLED*. Once the new member has received the notification packet from its Designated Router, it will be able to send / receive data packets according to its status within the session.

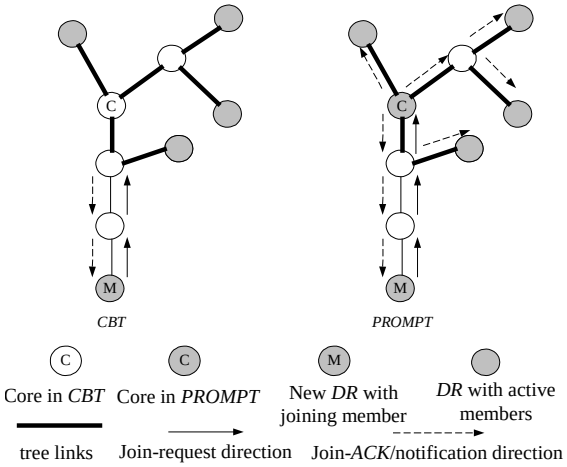

Fig. 2 Join Mechanism of *CBT* and *PROMPT*

Step 3: Session data transmission.
As we have mentioned above, all the session data must be transmitted along the bi-directional shared tree. This is very similar to the function of *CBT* [1]. However, since all the data sources are session members, there is no need to perform IP-in-IP encapsulations for non-member senders in *PROMPT*. In fact packets from any non-member sender will be discarded as they arrive at on-tree routers so that members only receive data from their peers in the session. Moreover, the session manager can restrict the behavior of each session member by resetting the corresponding value of *Status* in their upstream routers. The implementation of these control mechanisms will be described in section 3.4.
Step 4: Deregistration. Once a session member wants to leave, it just sends a deregistration packet towards the session manager using the core address. All the routers receiving this packet will forward it to their upstream interface such that the manager will finally receive it. The downstream interface of the routers along the corresponding path will set the value of *Status* to *DISABLED* in the entry for this member so that it will enter a pending state. When the router receives the leaving-notification packet from the session manager, the entry for this member is deleted. If there are no other session members attached to this interface, the router stops forwarding data to it. At the same time, the router will check if there exist other session members attached to any other interface. If there are none, it will delete the state for this session at the upstream interface by removing the entry for the session manager, and hence the router will break up from the shared tree. Upon receiving the deregistration packet, the core will multicast a notification packet to all the members to let them know that the particular member has left the session.

3.3. Packet Classification
There are four basic types of packets in *PROMPT*:
(1) *Registration/Deregistration packets*: This type of packet is used for session members to join/leave the session. Such packets are forwarded uniquely to the upstream interface once they are received from the downstream interface. If the router has not been on the tree, it will forward the registration packet from the interface that is used to deliver unicast packets to the session manager. If a registration is received from the upstream interface, it is dropped silently. These packets result in a member entering the pending state, with the real effect taking place only when the corresponding notification packet is received.
(2) *Notification packets*: This type of packet is used for the session manager to notify all the session members that a new member is joining or a current member is leaving. Since *PROMPT* does not perform *RPF* check but uses *hard state* to maintain the shared tree, a copy of the join-notification packet must be received via the same interface from where the original registration packet for this new member was sent, so that the new branch will be activated. The new host will not be able to send or receive messages until it has received the join-notification packet from the session manager indicating that it has already been approved to become a session member. Similarly, the intermediate router won't delete the entry for a leaving member until the corresponding leaving-notification packet is received from the session manager. When an intermediate router receives a join-notification packet for the first time, it will copy the session manager's unicast address into the corresponding entry (remember that the value of *UA* for the session manager was left empty when the registration packet was received) and set the *Status* value to *ENABLED*. In order to let the new member know about all the existing participants, the join-notification packet also contains the full list of current session member addresses.
(3) *Data packets*: This type of packet is used to carry messages from session members. If a router receives such a packet from its downstream interface, it will first authenticate whether the data source is a valid member and also check if the packet itself comes from the right interface; then the router will forward the data packet not only to the upstream interface but also to all the downstream interfaces except the one from which this data packet was received. If the data packet comes from the upstream interface, the router

will always forward them to all its downstream interfaces with the session state. All the forwarding behaviors are subject to the value of *Status* for each session member.

(4) *Marking packets*: This type of packet is used for the session manager to restrict the behavior of all the other members. Once a router receives a marking packet, it will reset the value of *Status* in the entry for the corresponding member according to the behavior code contained in the marking packet. In that way the session manager is able to control the behavior of any other member. Marking packets can either be unicast to any session member or multicast to all the members such that restriction for both an individual host and the whole group can be achieved.

3.4 Control Mechanisms

First we will describe how the whole session is immune to data from non-member senders in *PROMPT*. As we have mentioned, during the registration phase, the router creates entries for all the session members whose registration packet has passed through this router, i.e., each router has got the detailed information of all its downstream session members as well as the session manager. Moreover, it can be inferred that the core router has recorded the information for all the members because all the registration packets will be forwarded via the core to the session manager.

When a router receives a data packet from one of its downstream interfaces, it will first check if there exists such an entry for the data source and if the router cannot find a matching entry that contains the unicast address of the source, the data packet is discarded. Then the router will verify if this packet comes from the same interface with the one from which the registration packet of the data source was received. Only if the data packet has passed these two mechanisms of authentication, it will be forwarded to the upstream interface and the other downstream interfaces with the session state (if any). When a data packet comes from the upstream interface, the router will always forward it to all its downstream interfaces with the session state. Although the router cannot judge if this is a valid data packet, since it comes from its unique upstream router, there exist only two possibilities: either the upstream router has the entry for the data source or the upstream router has received the packet from its own parent router in the tree. The worst case is that none of the intermediate ancestral routers have such an entry and then we have to backtrack to the core. Since the core has recorded entries for all the session members and it never forwards any unauthenticated packet to its downstream interfaces, we can safely conclude that packets received from the upstream interface are always from valid session members. However, this scenario precludes the case of routers attached on multi-access networks, and we will discuss the corresponding operations in section 3.6.

As we have also mentioned, *PROMPT* can achieve not only data source authentication but also restrictions over the behavior of all the other session members. Such type of control can be performed on both individual members and on the whole group. For individual members, the session manager will send a marking packet containing the corresponding unicast address via the core to change the value of *Status* in the entry of all the routers on the path to this member. After changing the *Status* value contained in the entry, each intermediate router will only forward this marking packet to the downstream interface which this member is attached to. Hence this kind of individual marking packet is only "unicast" toward the destination session member. If the manager wants to restrict the behavior of all the other session members, the corresponding group marking packet will not contain any unicast address of individual member but only the session address. Once the router receives such a packet from its upstream interface, it will change the *Status* value in all the entries and then forward the marking packet to all the downstream interfaces with the state. If the session manager sends *RECEIVE-ONLY* marking packets down to all the members preventing them from sending, the *SSM* model is emulated.

3.5 An Example

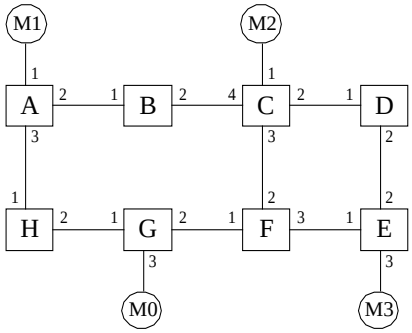

Fig. 3 An example of *PROMPT* routing

We will use the network model in Fig. 3 as an example to illustrate how *PROMPT* works. Suppose that *M0* is the session manager for session *M* and hence its first hop router *G* will become the core in the tree. Initially *M0* will perform a self-registration at the core and the core will create a corresponding entry <*M, M0, 3, ENABLED*> for the manager. When *M1* wants to join the session, it will send a registration packet via the shortest path *A*→*H*→*G* to the core. Every time *G* receives registration packets, it will always forward it to the session manager *M0* from interface 3. The routers along the path will create an entry for the new member *M1* and the session manager *M0* with the initial value of *Status* to *DISABLED*. The *UA* value of *M0* is left unfilled e.g. the entry in *H2* will be <*M*, _, 2, *DISABLED*>. When *M0* finishes authentication and sends out the join notification packet, the routers along the way will

reset the value of *Status* to *ENABLED* in the entries for the manager *M0* and the new member *M1*. Moreover the unicast address of *M0* contained in the notification packet will be copied into the entry for *H2* and *A3* as the packet travels along its way to *M1*. Similarly, after *M2* and *M3* join, each of the routers involved has created entries for *M0* and all their downstream session members. The resulting shared tree with the entries of each router is shown in Fig. 4. In this way valid data packets natively flow between session members, e.g., in Fig. 4, when the data packets from *M3* arrives at interface *F3*, since router *F* has found an entry for *M3*, it will forward the data on *F1* and *F2* to reach core *G* and *M2* respectively. By recording these entries, the session members will not receive data from non-member senders since the packets are discarded when intermediate routers fail to find a matching entry for the source.

During the session, if the manager *M0* wants to restrict the behavior of individual members or the whole session group, the core will unicast or multicast a marking packet to the necessary interface(s) after receiving it. For example, if *M0* wants to prevent *M2* from sending packets, it sends a corresponding marking packet containing the *UA* of *M2* and *RECEIVE-ONLY* code to the core. On receiving this packet the core router *G* finds that *M2* is attached to interface 2, hence the router will exclusively forward the marking packet to this interface after resetting the value of *Status* in the entry to *RECEIVE-ONLY* for *M2*. Since the entries for *M2* contained in router *G*, *F* and *C* have the *RECEIVE-ONLY Status*, any packet from this member will be discarded at router *C*.

When *M3* wants to leave the session, it sends a deregistration packet toward the core. The *Status* value for *M3* contained in router *G*, *F* and *E* will be reset to *DISABLED*. Once these routers sequentially receive the leaving-notification from *M0*, all the entries for *M3* are then deleted. Router *E* finds that no other session members are attached to any of its interfaces, so it will delete the entry for *M0* and hence breaks from the multicast tree.

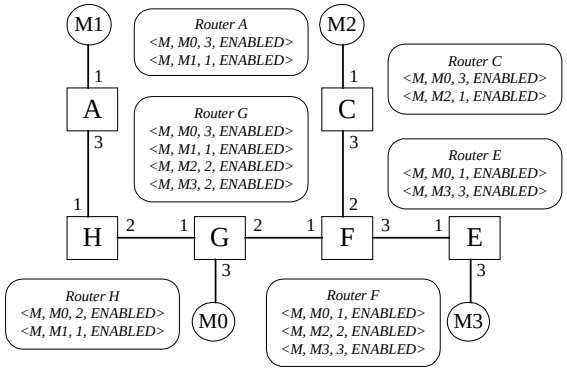

Fig. 4 Resulting shared tree with routers' entry information

3.6 Router Operations in Multi-access Networks

We need special consideration for protecting members from non-member sources attached to multi-access networks such as *LANs*. As we have mentioned, if an intermediate router receives data packets from its upstream interface, it will always forward them to all the downstream interfaces with the session state since these packets have been assumed to come from a valid session member. However this may not be the case if the upstream interface of a router is attached to a broadcast network. When a non-member host wants to send data with session address to the multi-access *LAN*, some mechanism must be provided to prevent these packets from being delivered to all the downstream members. To achieve this, once the Designated Router (*DR*) on the *LAN* receives such a packet from its downstream interface, if it cannot find a matching entry for the data source, it will discard it, and at the same time this *DR* will send a "forbidding" packet containing the address of the malicious source to the *LAN* from its downstream interface. Once the downstream router receives this packet in its upstream interface, it will stop forwarding the data with this unicast address which originates from a non-member host attached to the *LAN*. Hence all the downstream session members will only receive little amount of useless data for a short period of time. Fig. 5 gives a detailed example of this type of operation on a *LAN*. Router *A* is the *DR* leading to the core *C* and router *B* is one of the downstream routers attached with two session members *R1* and *R2*. Suppose that the non-member-sender *S* wants to send messages to the session address from the *LAN*, since router *B* receives the data from its upstream interface *B1*, it always assumes that the data comes from a valid session member and hence router *B* will forward the packets to its downstream interfaces. Once router *A* receives the data from its downstream interface *A2*, it can't find a matching entry for host *S*, so router *A* will send a forbidding packet containing the unicast address of *S* to the *LAN* via interface *A2*, and later on once router *B* receives this forbidding packet it will discard packets coming from this non-member host *S*.

In *PROMPT* the *DR* should not suppress additional *IGMP* membership reports from the *LAN* as it does in IP multicast, so individual members attached on the *LAN* will perform their own registration with the session manager. In a similar fashion to all the other multicast routing protocols, at any time there must be only one *DR* on a multi-access network towards the core. This guarantees that all the downstream members use a unique common path to send packets upstream towards the session manager. In order to avoid the formulation of parallel paths or loops, the router with the best path to the session manager should be elected as the *DR* on the *LAN*, however this requires that all the routers on the *LAN* have consistent view of the network metric. If the *DR*

loses reachability to the session manager, a new *DR* election will have to start. Once the election finishes, all the session members attached on the *LAN* will send registration packets to the new *DR*; besides, if there are any downstream routers attached on the *LAN* with active members, they will also respectively send a special type of sub-group registration packet containing all the information for the downstream members it has recorded.

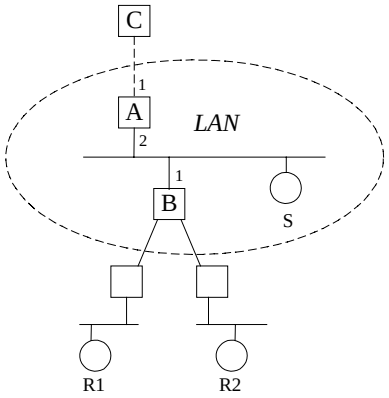

Fig. 5 Sending forbidding packet on *LAN*

4. SCALABILITY ANALYSIS

One of the most significant scaling issues faced by any multicast routing scheme is the amount of memory consumed by the routing table as the number of sources increases. In *PPM*, the maximum memory space needed by *PROMPT* is *O(km)* where *k* is the number of concurrent sessions and *m* is the number of members per session. This is the same with that needed in other multicast routing protocols based on source specific trees. It should be noted that applications such as videoconference are in reality "narrowcast" with few session members compared with Internet TV/radio applications that could have up to millions of subscribers simultaneously. Hence the value of *m* cannot be very large and it will not be a huge overhead for a router to keep the entry for all its downstream members.

In order to evaluate the average memory consumption for recording necessary entries in different types of routing protocols, we performed a simulation study on the average number of entries per on-tree router needed in Source Specific Tree (*SST*, e.g. *DVMRP*, *PIM-SM* shortest path routing etc.), *PROMPT* and *CBT* respectively. [3] presents an extensive performance comparison between *CBT* and *PIM* covering many features, including routing table size. In our simulation, we first build trees for a given session group and then calculate the routing table size per on-tree router. In *source specific tree* routing, one shortest path tree is formed rooted at the first hop router (*DR*) of each session member and is spanned to all the other peers, while in *PROMPT* and *CBT* a unique shared tree is built with the core located at the first hop router (*DR*) of a session manager that is randomly selected. For generalization, we only consider the necessary entries involved in source specific trees in a stable state for all the *SST* protocols. In fact in *PIM-SM*, additional entries for the shared tree still exist allowing for new sources even when *DR*s of all the current members switch to the shortest paths from current sources, while in other protocols such as *DVMRP*, *MOSPF* and *PIM-DM*, due to the flooding of the data, entries are set up throughout the network initially and then the unnecessary entries are removed after the multicast tree becomes stable. Hence we regard the value obtained in our simulation as the lower bound for *SST* routing table size. For simplicity we only consider one session group in our simulation. The average number of entries needed in *SST* can be expressed as:

$$\overline{R}(SST) = \frac{\sum_{i=1}^{n}\sum_{j\in G} X_{ij}}{\sum_{i=1}^{n} Y_i}$$

where *n* is network size and *G* denotes the session group.

Besides,

$$X_{ij} = \begin{cases} 1 & \text{if router } i \text{ is included in the shortest path tree} \\ & \text{rooted at member } j \\ 0 & \text{otherwise} \end{cases}$$

and

$$Y_i = \begin{cases} 1 & \text{if router } i \text{ is included in at least one shortest} \\ & \text{path tree} \\ 0 & \text{otherwise} \end{cases}$$

In *PROMPT*, we can regard the shared tree as a hierarchical structure with the core at the top level, i.e. level 0. It is noted that besides all the downstream members, each individual router also has an entry for the session manager. Therefore the number of entries contained in router *i* in the multicast tree *T* is:

$$R(i) = \sum_{(i,j)\in T} (R(j) - 1) + 1$$

and the average number of entries per on-tree router in *PROMPT* is:

$$\overline{R}(PROMPT) = \frac{\sum_{i=0}^{H}\sum_{j=1}^{L_i} R(j)}{\sum_{i=1}^{n} Y'_i}$$

where *H* is the number of hops from the farthest session member (or maximum level) and L_i is the number of routers on level *i*, while

$$Y'_i = \begin{cases} 1 & \text{if router } i \text{ is included in the shared tree} \\ 0 & \text{otherwise} \end{cases}$$

In our simulation, we adopt the commonly used Waxman's random graph generator [11] to create network models. A random network with 100 routers is generated with the number of session members varying from 10 to 50 in step of 5. Table 1 shows the total number of entries needed in the three types of routing protocols (*S—SST*, *P—PROMPT*, *C—CBT*)

versus the size of session group. From the table we observe that the total number of entries in *SST* grows significantly with the increase of the group size, while the number of entries needed in *CBT* grows very slowly. This numerical result is consistent with that in [3], i.e., the number of entries needed in *SST* (e.g. *PIM-SM*) is approximately *m* times the one needed in *CBT* where *m* is the group size. Moreover, we find that *PROMPT* also needs small number of entries compared with *SST* and hence performs well in scalability regarding memory consumption.

	10	20	30	40	50
S	227	755	1394	2291	3346
P	43	70	126	168	207
C	22	37	46	56	65

Table 1. Total number of entries *vs.* group size

In Fig. 6 we show the ratios over *CBT* for both *SST* and *PROMPT* routing regarding the average number of entries *per on-tree router*. From the figure we can see that the average number of entries per router needed in *SST* grows sharply with the increase of the number of session members. The result is expected because each *DR* has to create entries for all the other session members that are not locally attached, which makes significant contribution to the large average table size. On the other hand, *PROMPT* results in very small number of entries per router on average. Moreover, the curve does not rise significantly with the increase of the number of session members. Different from *SST*, in *PROMPT* each on-tree router only records the entry for its downstream session members as well as the session manager, and in the extreme case, only the core (or possibly a few other routers which are the *unique* successors of the core) has the entries for all the session members. Hence, the most significant contrast between *SST* and *PROMPT* is that, in *SST* the more "marginal" the router is on the tree, the greater number of entries are needed, while in *PROMPT*, edge routers keep less entries compared with those near the core.

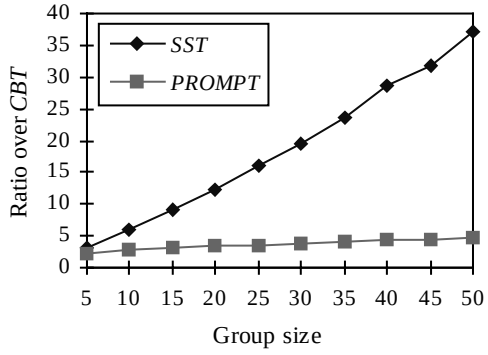

Fig. 6 Ratio comparison *vs*. number of session members

From the simulation above, we can conclude that *PROMPT* needs much smaller number of entries than *SST* on average and hence achieves superior scalability, though the maximum memory space needed by both protocols is *O(km)*.

5. SUMMARY

In this paper we introduced a new closed group service model called Peer-to-Peer Multicast (*PPM*). A simple but efficient routing protocol named *PROMPT* was proposed to support *PPM*, with the capability of data immunity from non-member senders and the possibility to restrict session members' behavior. While part of this functionality could be potentially achieved with application layer mechanisms, the network layer solution proposed here is more efficient, has sophisticated group management functionality which may not be possible in application-based solutions and will allow network providers to offer *SSM* and *PPM* based multicast services *within* their networks. Simulation results show that the proposed protocol *PROMPT* achieves good scalability with respect to memory consumption for routing entries in comparison to source specific trees, which is essential for deployment on the Internet. In our future work we plan to investigate robustness improvement, Quality-of-Service (*QoS*) issues, etc.

REFERENCES

[1] T. Ballardie, P. Francis, J. Crowcroft, "*Core Based Trees (CBT): An Architecture for Scalable Multicast routing*", Proc. *SIGCOMM'93*, pp85-95

[2] A. Ballardie, "*Scalable Multicast Key Distribution*", *RFC* 1949

[3] T. Billhartz *et al*, "*Performance and Resource Cost Comparisons for the CBT and PIM Multicast Routing Protocols*", *IEEE JSAC* Vol. 15, No. 3, 1997, pp304-315

[4] S. Deering, "*Multicast Routing in Internetworks and Extended LANs*", Proc. *ACM SIGCOMM*, 1988

[5] S. Deering *et al*, "*The PIM Architecture for Wide-Area Multicast Routing*", *IEEE/ACM* Transactions on Networking, Vol. 4, No. 2, Apr. 1996, pp 153-162

[6] C. Diot *et al*, "*Deployment Issues for the IP Multicast Service and Architecture*", *IEEE* Network, Jan./Feb. 2000, pp 78-88

[7] W. Fenner, "*Internet Group management Protocol, version 2*", *RFC* 2236, Nov. 1997

[8] H. W. Holbrook, D. R. Cheriton, "*IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications"*, Proc. *ACM SIGCOMM'99*

[9] B. N. Levine *et al*, "*Consideration of Receiver Interest for IP Multicast Delivery*", Proc. *IEEE INFOCOM* 2000, vol. 2, pp470-479

[10] R. Perlman *et al*, "*Simple Multicast: A design for simple, low-overhead multicast*" Internet Draft, draft-perlman-simple-mulitcast-*.txt, Feb. 1999, work in progress

[11] B. M. Waxman, "*Routing of multipoint connections*", IEEE *JSAC* 6(9) 1988, pp 1617-1622