REUNITE: A Recursive Unicast Approach to Multicast

Ion Stoica, T. S. Eugene Ng, Hui Zhang
Carnegie Mellon University, PA 15213
{istoica, eugeneng, hzhang}@cs.cmu.edu

Abstract—**We propose a new multicast protocol called REUNITE. The key idea of REUNITE is to use recursive unicast trees to implement multicast service. REUNITE does not use class D IP addresses. Instead, both group identification and data forwarding are based on unicast IP addresses. Compared with existing IP multicast protocols, REUNITE has several unique properties. First, only routers that are acting as multicast tree branching points for a group need to keep multicast *forwarding* state of the group. All other non-branching-point routers simply forward data packets by unicast routing. In addition, REUNITE can be incrementally deployed in the sense that it works even if only a subset of the routers implement the protocol. Furthermore, REUNITE supports load balancing and graceful degradation such that when a router does not have resources (forwarding table entry, buffer space, processing power) to support additional multicast groups, the branching can be automatically migrated to other less loaded routers. Finally, sender access control can be easily supported in REUNITE.**

I. Introduction

IP multicast, which was proposed by Deering in 1988, has two important components: the service model and routing protocols [1]. In the IP multicast service model, a group of receiver hosts can be identified by a single class D IP *group* address. Any host can send to the group by setting the destination address in the IP header as the group address. Receivers can dynamically join and leave the group. Such a service model provides a powerful abstraction for applications as end hosts (senders and receivers) can utilize the service without having to keep track of the membership of the group. It is the responsibility of IP multicast routing protocols to maintain the membership information and to build multicast distribution trees to deliver packets from a sender to all the receivers in a group.

Despite a decade of research and development, there are still open technical issues that make it difficult to deploy IP multicast in the global Internet. From the point of view of routing, existing IP multicast routing protocols [1], [2], [3], [4], [5], [6] scale poorly with large number of groups. In particular, with current routing protocols, each router needs to maintain a multicast forwarding table entry for every group whose distribution tree passes through the router. Therefore, the size of the multicast *forwarding* table needs to grow with the number of active groups, which results in higher router cost and lower forwarding performance. From the point of view of the service model, the current model requires each new group to be allocated a globally unique address. This is difficult to do in a large-scale distributed environment [7]. In addition, the current model does not provide means to control who is allowed to send to the group – any host can send to any IP multicast address. While this is also the case for IP unicast, the waste of network resources, disruption or denial of service by unauthorized senders can be greatly amplified in the case of multicast due to the potentially large number of receivers in the group [6].

This research was sponsored by DARPA under contract numbers N66001-96-C-8528, F30602-99-1-0518, and E30602-97-2-0287, and by NSF under grant numbers Career Award NCR-9624979 ANI-9730105, and ANI-9814929. Additional support was provided by Intel, Lucent, and Ericsson. Views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of DARPA, NSF, Intel, Lucent, Ericsson or the U.S. government.

Several schemes (e.g., Simple Multicast [5] and EXPRESS [6]) have been proposed recently to tackle the address allocation and the sender access control problems. In these schemes, there is a special node (sender or core) associated with each group and the group is identified by a two-tuple <special node's unicast IP address, class D multicast address>. The allocation of group address becomes trivial as by locally enforcing the uniqueness of the class D addresses used at each node, the uniqueness of the two-tuples are enforced. In addition, access control of senders can be supported by forcing all packets to go to the special node to be authenticated before being multicasted to the receivers.

While these proposals address some important issues related to the service model of IP multicast, the scalability problem of IP multicast routing still remains. In this paper, we propose a novel multicast scheme called REUNITE (REcursive UNicast TreE) to address the scalability issues. Unlike all existing IP multicast protocols, REUNITE does not use class D IP addresses. Instead, both data forwarding and group identification are based on unicast IP addresses. Multicast data forwarding is implemented with a novel technique called recursive unicast. A group is identified by a two-tuple $< root\_IP\_address, root\_port\_number >$ where the root node can be either the sender or a special node. Compared with existing IP multicast solutions, REUNITE has several important advantages:

Enhanced Scalability by Reduction of Forwarding State With REUNITE, only routers that are acting as multicast tree branching points for a group need to keep multicast forwarding state of the group. Non-branching-point routers simply forward packets by unicast routing.

No Need for Class D IP Address With REUNITE, a multicast group is identified by a two tuple $< unicast\_IP\_address,$ $port\_number >$ and there is no need for a separate block of class D IP addresses. In this case, the allocation of unique group identification becomes trivial. In addition, the maximum number of simultaneously active multicast groups increases dramatically.

Native Support for Incremental Deployment Since unicast addresses are used as destination addresses in the IP header, a

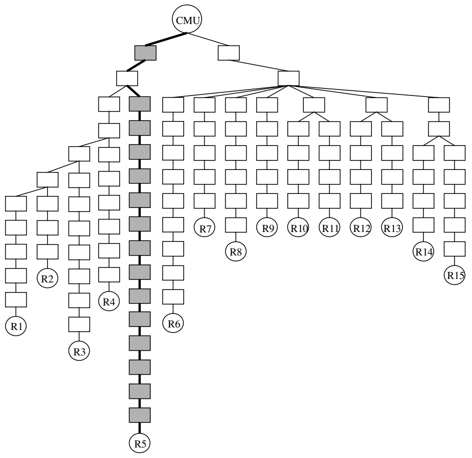

Fig. 1. Traceroute experiment from CMU to 15 U.S. sites.

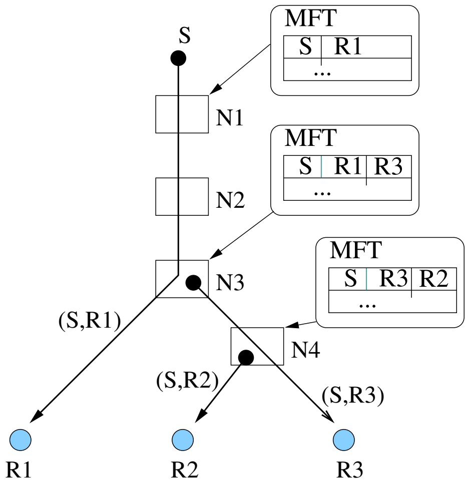

Fig. 2. Example of packet forwarding in REUNITE. Packets are sent via unicast and replicated at branching points.

router that does not implement REUNITE will simply forward the packets to the next hop based on the unicast destination address, without any adverse effect on the protocol other than the potential loss of efficiency. This allows REUNITE to be incrementally deployed only at a subset of network nodes, without the need of tunnelling.

Load Balancing and Graceful Degradation With REUNITE, when a router does not have resources (forwarding table entry, buffer space, processing power) to support additional multicast groups, it can simply ignore further protocol messages and the branching point will be automatically migrated to other routers.

Support for Access Control Access control can be implemented by authenticating senders at the root node.

II. Multicast Scalability and Sparse Groups

As discussed in Section I, existing multicast protocols are not scalable with respect to the number of simultaneously active groups. This is because each router needs to maintain a multicast *forwarding* table entry for every group whose distribution tree passes through the router. Techniques such as hierarchical address assignment and forwarding based on longest prefix match, which achieve great reduction in the unicast forwarding table size, cannot be easily applied to multicast [8].

While the number of multicast groups can be large, we speculate that a majority of the groups will be very sparse. An important observation is that when the members of a multicast group is distributed sparsely in the network, the data delivery tree of the group is likely to have a large number of non-branching routers or routers that have only one downstream router. To illustrate this point, we obtained results from a set of traceroute experiments[1] from Carnegie Mellon University to 15 U.S. sites and constructed the resulting tree as shown in Figure 1. Assuming routing is symmetric, DVMRP [1] would create the same tree for CMU multicasting to these 15 destinations. Clearly, most of the routers in the tree are non-branching. For example, on the path from CMU to receiver $R5$, 15 out of 16 routers are not performing any multicast operations other than forwarding the packets to the next hop. Furthermore, in the entire multicast tree, there are only 8 branching points out of 97 routers. With most existing multicast protocols, even these non-branching routers need to maintain for this group a multicast forwarding entry, which is an important scarce resource in multicast routers.

In this paper, we propose a new multicast protocol called REUNITE. One of its main advantages is that non-branching routers do not need to maintain the forwarding state for the group. This has the potential of greatly reducing the size of multicast forwarding table in a network that has a large number of sparse groups.

III. REUNITE Addressing and Forwarding Algorithm

The key idea of the REUNITE protocol is to use *recursive unicast* to implement multicast service. For each group, REUNITE builds a delivery tree rooted at a specially designated node called *root*. Every branching node of the tree maintains a list of receivers' addresses. A receiver R is said to have joined the multicast tree at node N if R's address is maintained at N. In REUNITE, a receiver's address is maintained at exactly one node in the group's delivery tree. To multicast a packet, the root sends a copy of the packet to each receiver in its list. Similarly, when a branching node forwards such a packet, it sends a copy of the packet to each receiver in its own list. This procedure continues recursively until packets reach all leaf nodes of the tree, i.e., all receivers.

Consider the example in Figure 2, which shows a multicast group with three receivers. Assume S is the source and the root, $R1$ joins at S, $R3$ at $N3$, and $R2$ at $N4$. Note that only $N3$ and $N4$ are branching nodes; $N1$ and $N2$ are not. The list of receivers maintained by each node is shown in the *last* entry of the corresponding tables. When S multicasts a packet, it simply sends the packet to all receivers in its list, which in this case consists only of $R1$. When $N3$ forwards this packet it also sends a copy to $R3$, which is the only receiver in its list. Finally, when the copy traverses $N4$, $N4$ makes another copy and sends it to $R2$.

[1] We thank Sanjay Gopinatha Rao for providing us with these data.

Using unicast addresses instead of class D addresses for data delivery is a key difference between REUNITE and all existing IP multicast protocols [1], [2], [3], [4], [5], [6]. As a result, while in these protocols each router in the multicast tree has to maintain multicast forwarding state, in REUNITE multicast packets can be simply forwarded based on a router's unicast forwarding table in any of the following cases: (a) the router is non-branching, (b) the router does not implement REUNITE, or (c) the router runs out of multicast related resources such as forwarding table entries. As will be discussed later, this results in three unique advantages of REUNITE: (a) enhanced scalability due to reduction of forwarding state, (b) native support for incremental deployment, (c) graceful degradation and load balancing.

We will present the details of the REUNITE addressing and forwarding algorithm in the rest of this section, and describe the tree maintenance protocol in the next section.

A. Addressing

One of the key distinctive features of REUNITE is that it uses only unicast IP addresses for both data forwarding and group identification purposes. In contrast, all existing IP multicast protocols use class D IP addresses.

In REUNITE, there is a special root node associated with each group. While any node can serve as the root, the source may be a more desirable choice in the case of *single-source* or *almost single-source* applications [6]. A multicast group is identified by a two tuple < *root_IP_ addr, root_port_number*>. This makes the generation of globally unique group identifiers trivial as it only requires each of the root nodes to generate a locally unique port number.

For each multicast packet, the source and destination address fields in the packet header are set to be the IP addresses of the root and one of the receivers in the group, respectively.

B. Forwarding Algorithm

For each multicast group, REUNITE builds a delivery tree that is rooted at the root node. Each REUNITE router maintains a Multicast Forwarding Table (MFT) that contains an entry for every multicast group whose data delivery tree branches at the router. An MFT entry has the following format

$$< root\_addr, root\_port >< dst, stale >$$
$$<< rcv_1, alive_1 >, \ldots, < rcv_n, alive_n >>$$

where $< root\_addr, root\_port >$ identifies the group; dst is the IP address of the first receiver that joins the group among all receivers in the downstream of the router; $recv_i, i = 1, \ldots, n$, called the *receiver list*, are IP addresses of the receivers to which the router will send replicated unicast packets when it receives a multicast packet from the group that is destined to dst; $stale$ and $alive$ are boolean variables. MFT state is soft; unless it is refreshed, an entry becomes *stale* in $TO1$ seconds. Similarly, if not refreshed, each receiver list entry becomes *not alive* in $TO1$ seconds.

Consider again the example in Figure 2, where the $stale$, $alive$, and the source port number are not shown for simplicity. Also, since the root does not use $< dst, stale >$, this is omitted. Assume $R1$ joins the group first, followed by $R3$, and then $R2$. As can be seen, only the branching nodes $N3$ and $N4$ have MFT entries for the group.

When a data packet with source address S, port number P, and destination address D arrives at a node, the forwarding algorithm searches for the entry $< S, P >< D, * >$ in the MFT (with the exception of the root node, where D is not used). If the entry exists, the packet is duplicated for each receiver in the receiver list of the group MFT entry. The destination address of each duplicated packet is replaced by the corresponding receiver's IP address. The original packet is simply forwarded based on its destination address. In the example, $N2$ will forward each multicast packet as a unicast packet because it does not have a corresponding entry in its MFT while $N3$ and $N4$ replicate the packet.

IV. REUNITE Tree Maintenance

As discussed earlier, the per group state in MFT at each branching router defines the multicast forwarding tree. The states are installed and deleted by a control protocol. In this section, we describe the control protocol that is used to create and maintain the MFT at each router. In addition to the MFT, each REUNITE router maintains another table called the Multicast Control Table (MCT). We note that a more complex version of the protocol that does not require the MCT is discussed in [9].

A router's MCT contains an entry for every group whose multicast delivery tree passes but does *not* branch at the router. A MCT entry has the following format:

$$< root\_addr, root\_port >< dst >$$

where $< root\_addr, root\_port >$ identifies the group, dst is the IP address of the first receiver that joins the group among all receivers in the downstream of the router. Again, MCT state is soft, and unless it is refreshed, an entry times out in $TO1$ seconds.

It is worth noting that if a REUNITE router is traversed by a multicast group's delivery tree, the router will maintain an entry either in its MFT (in the case that the tree branches at the router) or in its MCT (in the case that the tree does not branch). A natural question to ask is: since a REUNITE router does maintain *per group* state, why is REUNITE more scalable than other IP multicast protocols? The key observation is that only MFT needs to be maintained on the data plane, while MCT, as will be discussed later, can be maintained on the control plane. That is, when a data packet arrives, *only* MFT needs to be looked up. In contrast, MCT needs to be looked up only when control messages are processed. Therefore, by partitioning per group multicast state into forwarding and control state, REUNITE maintains a much smaller *per group forwarding table* than other IP multicast protocols in a network with a large number of sparse groups.

Unlike previous multicast protocols that only have control messages sent from receivers to the source or core, REUNITE uses two types of control messages: JOIN message, which is unicasted periodically by each receiver towards the root, and TREE message, which is *multicasted* periodically by the root along the multicast delivery tree. JOIN messages are used to

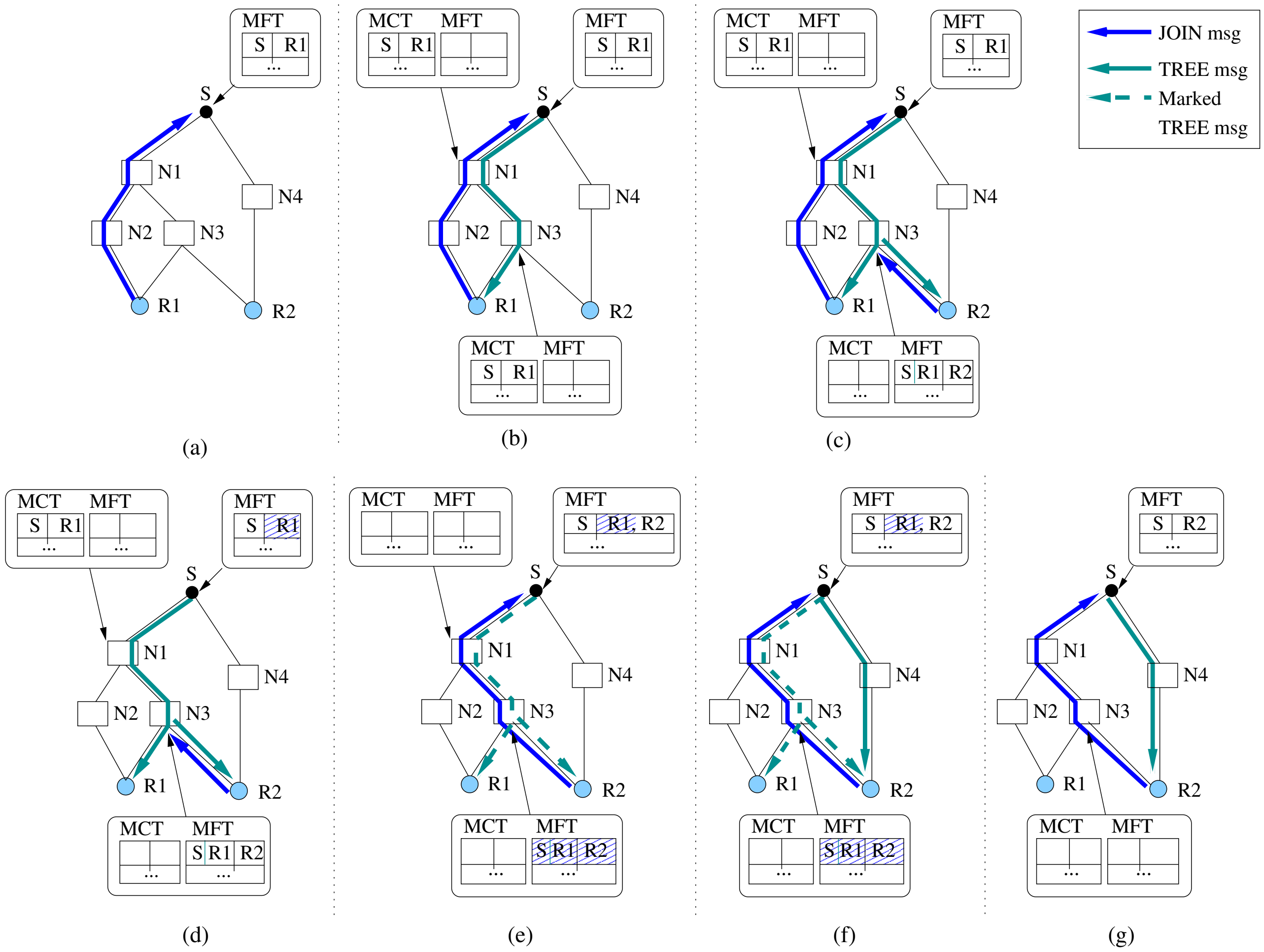

Fig. 3. Example illustrating the tree creation and maintenance protocol of REUNITE.

create and refresh the *receiver* entries in MFT, while TREE messages are used to create and refresh the entries in MCT and to refresh the *group* entries in MFT.

To describe the tree creation and maintenance operations, we use a detailed example shown in Figure 3. S is the source and the root of a group, $R1$ and $R2$ are the receivers, and $N1$ through $N4$ are router nodes. To better illustrate the properties of REUNITE, we assume the following *asymmetric* unicast routes: $S \rightarrow N1 \rightarrow N3 \rightarrow R1$, $R1 \rightarrow N2 \rightarrow N1 \rightarrow S$, $S \rightarrow N4 \rightarrow R2$, and $R2 \rightarrow N3 \rightarrow N1 \rightarrow S$. We omit the port number and the flags in the figures for simplicity. In addition, we also omit $< dst, stale >$ tuple from root's MFT, as it is not used by the root.

A. Joining a Group

Assume $R1$ is the first receiver that joins the group (Figure 3(a)). Since initially no router is aware of the group, the JOIN message sent by $R1$ is propagated all the way to S. Upon receiving this message, S creates an entry for $R1$ in its MFT. Since S maintains the MFT state for $R1$, we say $R1$ joins the multicast tree at S.

S then begins sending data packets to $R1$. In addition, S also sends periodic TREE messages down the delivery tree (Figure 3(b)). When a TREE message arrives at nodes $N1$ and $N3$, their MCT are updated to indicate that they are part of S's multicast forwarding tree. In particular, packets destined for $R1$ traverse through them.

Before we continue the example, it is worth noting that in a network where the paths between the root and a receiver are asymmetric, the JOIN and TREE messages will traverse different paths. In this example, the JOIN message from $R1$ passes $N2$, while the TREE message from S passes $N3$. This is quite different from all existing multicast protocols in which JOIN messages and data packets traverse the exact reverse paths. This is because, with REUNITE, each branch of the data delivery tree is constructed based on the *forward direction unicast* routing towards the receiver. In contrast, with other multicast protocols, the data delivery tree is constructed based on the *reverse direction unicast* routing towards the sender. Therefore, in a network with asymmetric links or paths, REUNITE may potentially generate a higher quality data delivery tree than other multicast protocols.

Now, suppose $R2$ joins the group by sending a JOIN message towards S (Figure 3(c)). Upon receiving this message, $N3$, which is part of the multicast tree, becomes a branching node. This is accomplished by removing the MCT entry for the group and creating a MFT entry for $R2$. From now on, data packets and TREE messages sent towards $R1$ by S will be replicated

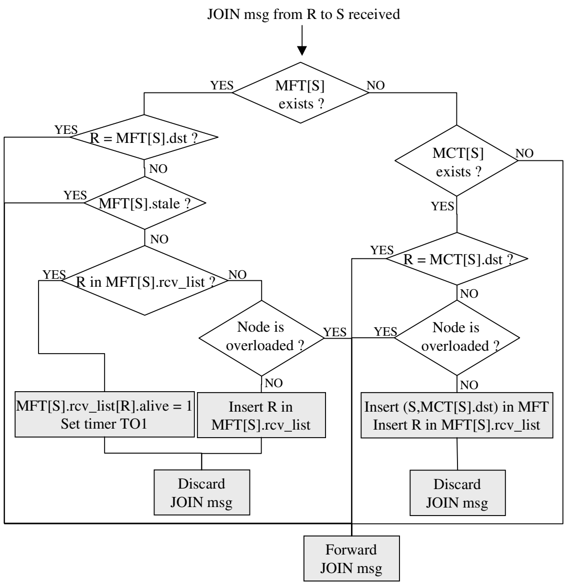

Fig. 4. Flowchart of JOIN message processing algorithm.

and sent to $R2$ by $N3$.

A receiver periodically sends JOIN messages to refresh the MFT entry at the router it joins. These JOIN messages are discarded at the router and will not be propagated further. In this example, $R1$'s and $R2$'s JOIN messages will reach S and $N3$, respectively.

B. Leaving a Group

To leave a group, a receiver simply stops sending JOIN messages. Consider the case where $R1$ decides to leave the group (Figure 3(d)). Since the MFT entry for $R1$ at S is no longer refreshed, after a time period of $TO1$ seconds, S concludes that $R1$ has left. However, note that S cannot stop sending data to $R1$ immediately, since other receivers ($R2$ in this example) might receive data that are replicated from those sent to $R1$. Thus, before S can remove the MFT entry for $R1$ and terminate the unicast flow, it *must* allow these receivers sufficient time to discover a new branch point to receive data from.

To accomplish this, S maintains the MFT $R1$ entry for an additional $TO2$ seconds, but marks it as being *not alive* (this is indicated by the shaded area in Figure 3(d)). During this time period, S keeps sending data and TREE messages to $R1$. However, these TREE messages are marked *stale* to indicate that the data flow to $R1$ is to be torn down (Figure 3(e)). When $N1$ receives such a TREE message with the *stale* bit set, it removes the corresponding entry from its MCT. When $N3$ receives such a TREE message, it marks its corresponding MFT entry as being stale as well. As a result, the next JOIN message from $R2$ is no longer intercepted by either $N3$ or $N1$. It eventually reaches S and a new MFT entry for $R2$ is created at S (Figure 3(e)).

From now on, S begins sending data and TREE messages to $R2$ and these packets pass through node $N4$ (Figure 3(f)). The TREE messages are processed by $N4$ as described before. The MFT entry for $R2$ at S is refreshed by subsequent JOIN messages from $R2$. During the time period until the stale MFT

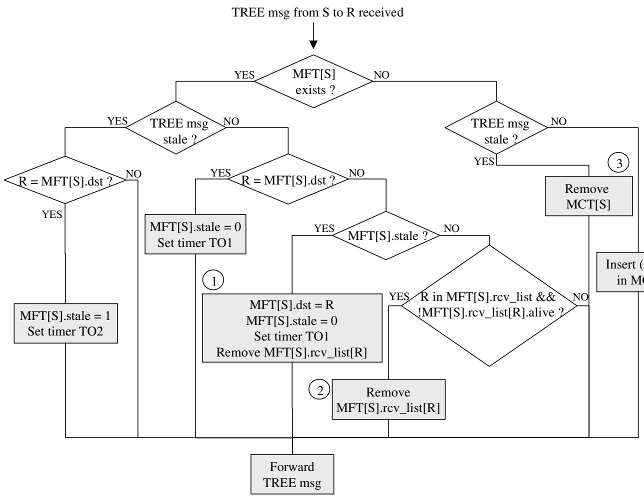

Fig. 5. Flowchart of TREE message processing algorithm.

entry for $R1$ at S is removed, $R2$ will receive some duplicate data packets. After $TO2$ seconds, the stale state at S and $N3$ for $R1$ is removed (Figure 3(g)). S therefore no longer sends any data or TREE messages to $R1$, and $R2$ will stop receiving duplicate data packets.

C. Details of the Tree Maintenance Protocol

While the previous example illustrates most of the important operations of the protocol, it is nonetheless a very simplified scenario. In this section, we specify the complete protocol by describing the details of message generation, message processing, and timeout handling.

Message Generation JOIN messages are periodically generated by each *receiver* and are unicasted to the *root* of the group. TREE messages are periodically generated by the *root* of the group and are multicast forwarded based on the root's own MFT. In both cases the message generation period should be less than $TO1$ seconds.

Message Processing Algorithms The message processing algorithms at non-root nodes are presented in Figures 4 and 5. Group address $< root\_addr, root\_port >$ is abbreviated as S, and dst_addr is abbreviated as dst.

The description of the JOIN message processing algorithm is implicitly covered in Section IV-A. The flowchart of the TREE message processing algorithm involves several cases not discussed in Section IV-B. Below, we briefly describe the other actions: (1), (2), and (3).

Action (1) is executed whenever a node that is a branching point for a group whose state is stale receives a non-stale TREE message destined to a receiver R. This can happen when the first receiver who joined at that node leaves the group, but there is another receiver who, in the meantime, has joined the group at an upstream node. When such a message is received, the group's entry in the MFT is refreshed. At the same time dst is set to the new receiver address R. This indicates that from now on the node will replicate only data and TREE messages received for R. In addition, the entry for R, if any, in the group's receiver list is removed.

Action (2) is performed when a receiver R's entry in the MFT is stale but the group entry is not stale, and an unmarked TREE message to R is received. This can happen when receiver R leaves the current node and joins at another node upstream (this may be caused by a change of the route from R to S.) In this case receiver R's entry is simply removed from the MFT as there is no longer any need to replicate and send packets to R at this node; the packets to R will be replicated by the new branching node, at which R has just joined.

Action (3) is executed whenever a *non-branching* node receives a stale TREE message. The action consists of simply removing the group entry, if any, from the MCT. This is because, the stale TREE message indicates that, after at most *TO2* seconds, data and TREE message transmissions may terminate, and as a result the node will no longer be part of the tree.

Finally, note that when a TREE message is replicated and forwarded to receiver R, if receiver R's entry in the MFT is stale, then the replicated TREE message is marked stale.

Timeout Handling When a timeout $TO1$ expires for an MFT group entry, the entire entry is marked as *stale*. A second timeout $TO2$ is set. When $TO2$ expires, the entire MFT entry is removed. When a timeout $TO1$ expires for a receiver entry in an MFT entry, the receiver entry is marked as *not alive*. A second timeout $TO2$ is set. When $TO2$ expires, the receiver entry in the MFT entry is removed. When a timeout $TO1$ expires for an MCT entry, the entry is removed.

V. REUNITE ADVANTAGES

Enhanced Scalability by Reducing Forwarding State Most of the existing multicast routing protocols require *every* router on a multicast tree to keep forwarding state for the multicast group. This is because forwarding is based on class D multicast addresses. In contrast, in REUNITE, only routers that are acting as multicast tree branching points for a group are required to keep multicast forwarding state of the group. All other non-branching nodes simply forward data packets by default unicast routing. In effect, REUNITE removes unnecessary forwarding state by converting it into control path state. As discussed in Section II this can lead to significant savings, especially in large networks with many sparse groups.

In the steady state, the amount of multicast forwarding state maintained in the entire network for a group is $O(r)$, where r is the number of receivers in the group. This is because each receiver joins the multicast tree at exactly one node, and only that node maintains the receiver's state. Note that this value is optimal for any multicast protocol that uses a tree topology. From a single router's point of view, if all routers in the network implement REUNITE, in the steady state, there is at most one receiver in a MFT entry for each of the input interfaces. This is because, in the steady state, a link in a network can be traversed by JOIN messages from at most one receiver per group.

Incrementally Deployable Most existing multicast protocols require every router in the network to implement the protocol. This introduces a deployment problem as it requires all routers in a network to be updated *simultaneously*. A possible solution is to use IP tunneling across the regions of the network that are not multicast aware.

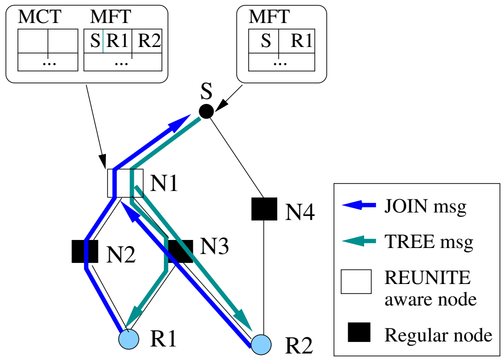

Fig. 6. Scenario illustrating the incremental deployability of REUNITE.

REUNITE, on the other hand, has native support for incremental deployment. Since all packets have unicast addresses, a router that does not implement REUNITE will forward the packets as if they are unicast packets. This does not affect the correctness of the protocol but may lose some efficiency. In the extreme case when no router implements our protocol, REUNITE degenerates into sending n unicast streams to n receivers from the root.

To illustrate, Figure 6 depicts the same scenario as in Figure 3(c), except that only node $N1$ implements REUNITE. In this case, $R2$ will join the tree at node $N1$, as node $N3$ no longer intercepts $R2$'s JOIN messages. As a result, the packets destined to $R2$ will be replicated at $N1$ instead of $N3$. Note that no tunneling is needed even though the down-stream node $N3$ is not REUNITE-aware.

Load Balancing and Graceful Degradation In multicast protocols that requires every router on a multicast tree to maintain forwarding state, if some of these routers are no longer able to store this state, either because they are overloaded or they run out of memory, the multicast tree will become partitioned. In contrast, since REUNITE does not require every router to process protocol messages, a router that is overloaded can choose to ignore further JOIN messages and let other upstream routers to process those messages and share the load.

For example, in the scenario shown in Figure 3(c), node $N3$ may choose to ignore a JOIN message from $R2$. In this case the JOIN message will simply propagate up-stream. If $N1$ choose to accept this message, then $R2$ will get multicast service from $N1$. This results in the same tree as shown in Figure 6. From the point of view of the new group, a router running out of forwarding table entries exhibits the same behavior as a non-REUNITE-aware router.

Unique Group Identification. Generating globally unique group identification is trivial in REUNITE as each root just needs to generate locally unique port numbers.

Support for Access Control REUNITE can also easily support sender access control. Since only the root is allowed to inject multicast traffic into the network, access control can be implemented simply by authenticating senders at the root node.

VI. Discussion

A. Protocol Dynamics

While the use of recursive unicast has many desirable properties, it also introduces dynamic behaviors that do not exist in other multicast protocols. In this section, we describe some situations with more complex dynamic behaviors and show that REUNITE can perform gracefully in these situations.

Tree Restructuring Due to Member Departure In REUNITE, when a receiver leaves a group, the corresponding branch in the data delivery tree will be removed and may affect other receivers on the same branch. As explained in Section IV-B and as shown in Figures 4 and 5, REUNITE has mechanisms to restructure the delivery tree so receivers do not lose any packet as a result.

Race Condition of Joins In REUNITE, an MCT entry for a group is created when a router receives a new TREE message generated as a result of a new receiver joining at an upstream node. Before this TREE message traverses the new branch and establishes MCT state on the branch, if another JOIN message from a second receiver arrives at a router on the branch, the message would be propagated upstream. However, had the MCT state been established, the JOIN message would have been intercepted by the router and this router would become the branching point for the second receiver. Due to this race condition of JOINs, the branching point of the second receiver is further upstream than necessary, resulting in a sub-optimal tree. Fortunately, the data delivery tree will only be in this sub-optimal state transiently. This is because once the MCT state has been established on the branch, subsequent JOIN messages from the second receiver will be intercepted and a new optimal branching point will be created. The previous non-optimal branching point will eventually timeout and be removed.

Duplicate Packets During Tree Restructuring As discussed in the previous paragraph, and also in Section IV-B, it is possible that during short time periods a receiver get duplicate packets. To reduce the number of duplicate packets, additional techniques can be used. For example, when a receiver joins the multicast tree at a node, the node can generate a TREE message *immediately* towards the receiver. This will update the MCT tables of nodes on the new branch immediately, without having to wait for the next TREE message generated by the root. With this technique, the time window during which new receivers cannot join at nodes on the newly created path is greatly reduced.

B. Multiple Senders

So far, we have assumed that the root is the only sender in a group in REUNITE. In this section we show how REUNITE can be extended to support multiple senders. The idea is to have the root acting like a "reflector". Suppose a host wants to multicast a packet to a group with address $< root\_addr, root\_port >$. Then, it will simply send a unicast packet with the destination address and port number set to $root\_addr$, and $root\_port$, respectively. When the root node receives the packet and determines that $< root\_addr, root\_port >$ corresponds to a multicast group rooted at itself, it just multicasts the packet to the group. Note that access control can be implemented easily by authenticating each sender before multicasting their packets.

We note that the mechanism to accommodate multiple senders in REUNITE is similar to the session relay approach proposed in EXPRESS [6]. However, unlike EXPRESS, our solution does not require an application level layer or IP encapsulation for unicasting packets from a sender to the root. Thus REUNITE can be implemented more efficiently.

However, this simple solution has several drawbacks. First, since all messages have to go through the root, any network partition or root failure will compromise the entire group. To alleviate this problem, a possible solution is to have a backup root, and use it whenever the primary one fails. Second, the transmission delay can become larger than directly unicasting a packet to the destination. However, we believe that for most applications, such delay increase is acceptable.

In comparison, solutions based on bidirectional trees, such as CBT or Simple, are more robust. In particular, in these solutions, members of a group may be able to communicate even if the core node fails. However, if access control is required, then this advantage is negated, as a special designated node, e.g., the core node [5], is assumed to perform this task.

C. Source Address Spoofing and Ingress Filtering

In REUNITE, when a router duplicates packets, it rewrites the destination address field in the packet header, but keeps the source address field to be the root address instead of overwriting the field with its own address. From the point of view of a router down-stream, this is equivalent to source address "spoofing". Routers implementing ingress filtering [10] interpret this as a security attack and automatically drop such packets. This problem is also shared by other protocols, such as Mobile IP [11].

In a network in which all routers implement REUNITE, a possible solution to protect against source address spoofing attack is to authenticate TREE messages and add in each MFT entry a *UpStreamInterface* field which is set to be the interface that the group's TREE message comes from. A multicast data packet is only forwarded if it comes from the *UpStreamInterface*. In a network in which not all routers implement REUNITE, protocol modification is needed to accommodate both REUNITE and ingress filtering. One solution is to use an IP option to store the root address and rewrite source address field whenever a packet is duplicated. The advantage of this approach is that it is compatible with non-REUNITE-aware routers that implement ingress filtering. The disadvantage is that it adds extra overhead in packet processing.

D. Unicast Packet Forwarding

When a REUNITE router receives a packet, it extracts the source and destination addresses and the source port number from the packet, and performs a lookup in the MFT. If the entry is not found, then a second lookup is performed in the IP forwarding table. Thus, when a regular *unicast* packet is received, two lookups are required. However, we note that since the MFT lookup involves an exact match as opposed to a longest-prefix-match, it can be performed faster than an IP forwarding table lookup.

E. Efficiency of Packet Replication

As discussed in Section III-B, REUNITE's MFT stores a list of *receiver addresses* to each of which a unicast packet needs to be sent. In contrast, the forwarding table of other multicast protocols stores the list of *output interfaces* to each of which a multicast packet needs to be replicated. This can be implemented efficiently by using a bitmap of output interfaces and leveraging the packet replication capabilities in the switch backplane.

Despite the MFT table's content is different, REUNITE's packet replication algorithm can also be implemented efficiently. The content of the MFT can be distributed among input and output ports of the router. At the input, an MFT entry will contain only $< root\_addr, root\_port >< dst\_addr, stale >< port\_mask >$, where $port\_mask$ is a bit mask which specifies the output ports to which a multicast packet needs to be forwarded. The receiver list associated to each group entry will be stored at the corresponding output ports. Therefore, packets can be replicated based on bitmaps and transmitted across the backplane in a fashion similar to existing IP multicast protocols. Rewriting the destination address field of duplicated packets can be done at corresponding output ports.

F. Accommodating Multicast-Capable Subnets

So far we have described REUNITE assuming a point-to-point network. However, many of the LAN and WAN technologies have native support for multicast. Sending individual unicast messages to each of the receivers in a multicast-capable subnet such as Ethernet is very inefficient.

A possible solution is to map a REUNITE group onto a *local* IP multicast group in such a network. Before joining, an end-host first sends a request containing a REUNITE group address to the local gateway. The local gateway maps this REUNITE group address onto a local IP multicast group address and replies the end-host with this local IP multicast address. Subsequently, the end-host joins the local IP multicast group by using IGMP [1], [12], [13]. The local gateway will then join the REUNITE group on behave of the local receivers. When a REUNITE packet is received by the local gateway, it translates the destination address and forwards the packet onto the local IP multicast group. There are two points worth noting. First, the IP multicast address allocation is simple because this address only has to be locally unique.[2] Second, this solution does not require changes in IGMP, or in the end-host's IP protocol stack.

VII. Simulation Experiments

We have implemented REUNITE in ns-2 [14]. In this section, we present results from three simulation experiments, illustrating three aspects of the protocol: graceful degradation and load balancing, incremental deployment, and dynamic join/leave of receivers.

A. Experiment Design

Due to the high overhead incurred by ns-2's packet-level simulation, we limit the simulation time to 60 seconds. In all experiments, senders become active during the first second and remain active afterwards. In the first two experiments, receivers join groups during the first ten seconds and remain active until the simulation ends. To remove the transient, in the first two experiments we report only the results for the last 50 seconds of the simulations, after all receivers have joined their groups. In the third experiment, receivers join and leave dynamically. Since the simulation time is short, we set the refresh period of the JOIN message to 2.5 sec. Correspondingly we set both timeouts $TO1$ and $TO2$ to 5 sec. Finally, all senders are assumed to send constant bit rate traffic with 1000 byte packets every 100 ms.

We use two performance metrics: *Average Redundancy* (AR), and *Maximum Redundancy* (MR). AR is defined over an interval $[t_1, t_2)$ as

$$AR(t_1, t_2) = \frac{P_t(t_1, t_2)}{P_u(t_1, t_2)}, \tag{1}$$

where $P_t(t_1, t_2)$ is the total number of multicast packets, and $P_u(t_1, t_2)$ is the total number of *unique* multicast packets sent during the interval $[t_1, t_2)$. For example, the AR of link $N1 : N3$ depicted in Figure 6 is two, since the link is traversed by two copies of each packet, one that is sent to $R1$ and the other that is sent to $R2$.

MR is defined as the maximum number of copies of a packet, including the original, that traverse a link. Again, in Figure 6, the MR for link $N1 : N3$ is two.

Note that AR and MR are always greater than or equal to one. Ideally, we want both to be equal to one, i.e., a link is traversed by only one copy of a packet.

B. Load Balancing and Graceful Degradation

In this experiment, we illustrate the behavior of our algorithm when routers do not have large enough MFTs to accommodate the entire multicast forwarding state.

For clarity, we use a simple topology as shown in Figure 7(a). There are 64 receivers and 16 groups and there are four receivers subscribing to each group. Ideally, we would like packets from each group to be replicated at node $N4$. However, this would require $N4$ to have at least 16 entries in its MFT, one for each group. If the MFT has less than 16 entries, some of the receivers will have to join at other routers up-stream, which will increase network load. As an example, assume that each router can store no more than six group entries in their MFTs. Then, $N4$ and $N3$ will store six group entries each, while $N2$ will store the remaining four. Consider a group that is stored at $N2$'s MFT, it is easy to see that both links 2 and 3 are traversed by four copies of each packet of this group, one for each of its receivers. Figure 7(b) plots the average redundancy (AR) along links 1, 2, and 3 versus the number of entries in the MFT. As expected, AR decreases as the MFT size increases. When MFT size is 16, AR becomes one as every receiver is able to join its group at node $N4$.

There are two points worth noting. First, even if a router does not have enough space in its MFT, the protocol continues to operate. Second, to reduce the network traffic it is more effective to have routers with large MFTs near the receivers rather than the senders, as this allows receivers to join their groups at routers in close proximity.

[2] We assume that the hosts in the subnet are only using REUNITE multicast. Otherwise, if both REUNITE and IP multicast are simultaneously used, then we assume that a block of class D IP addresses is exclusively allocated for REUNITE.

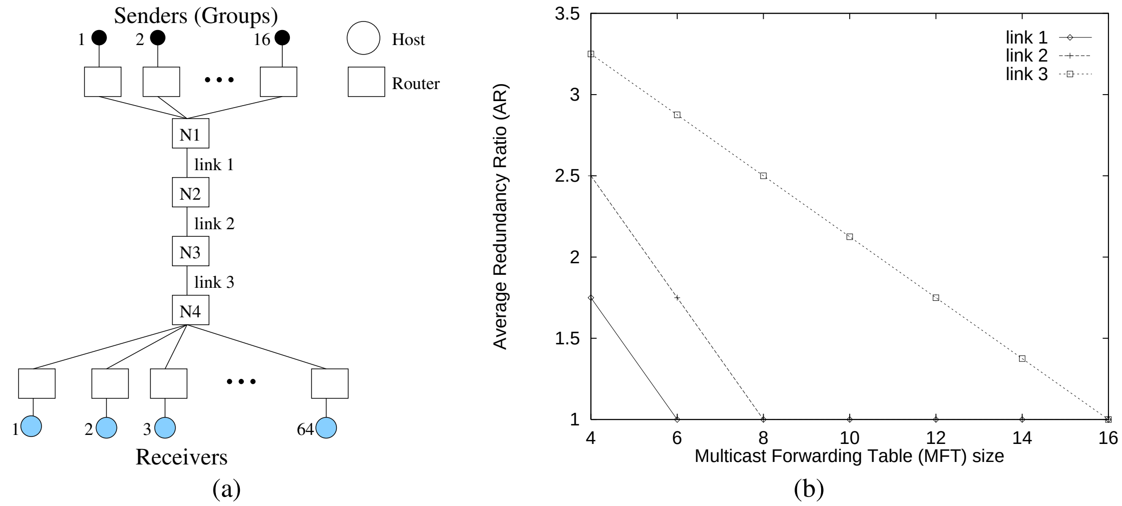

Fig. 7. (a) Experiment involving 16 groups with four receivers subscribing to each group. (b) Average redundancy (AR) of links 1, 2 and 3, versus the number of entries in each MFT.

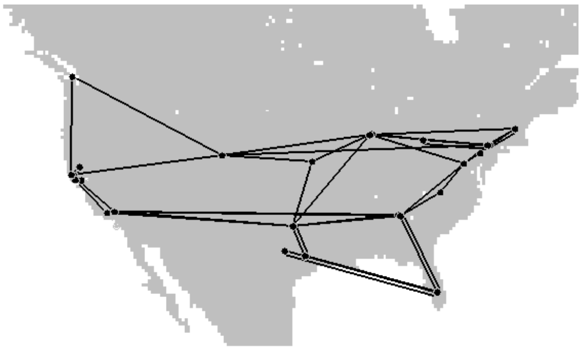

Fig. 8. MCI backbone topology. There are 8 groups and 64 receivers, randomly placed.

% REUNITE routers	0	20	40	60	80	100
AR	2.063	1.697	1.418	1.257	1.132	1
MR	12	8	5	4	3	1

TABLE I

AR AND MR ALONG ANY LINK IN THE NETWORK AS THE PERCENTAGE OF REUNITE-AWARE ROUTERS VARIES.

C. Incremental Deployment

In this experiment, we illustrates the incremental deployability of REUNITE and how the number of REUNITE-aware routers affects the performance. Here, we consider a more realistic topology, the MCI backbone network[3] shown in Figure 8. We assume there are 8 senders (or groups) and 64 receivers. Both senders and receivers are randomly placed, with the only restriction that no sender and receiver are connected to the same router. We vary the percentage of routers that are REUNITE-aware from 0 to 100 % in increments of 20%. For each percentage value *p*, we make ten independent simulations with random REUNITE-aware router assignment.

Table I shows the *AR* and *MR* for the entire network versus the percentage of routers that are REUNITE-aware. As expected, as more routers become REUNITE-aware, the lower the *AR* is. Note that when no router is REUNITE-aware, all receivers join directly at the senders, and thus the protocol degenerates into the sender generating unicast messages for each of the receivers. Note that MR is significantly larger than AR. In fact, if no router is REUNITE-aware, MR is as high as 12. Again, as the percentage of REUNITE-aware routers increases, MR decreases. When all routers are REUNITE-aware, *no* link carries duplicate packets, i.e., $MR = 1$.

[3] Topology obtained from www.caida.org in October 1998.

D. Performance with Dynamic Joins and Leaves

In REUNITE, a receiver leaving a group may cause other receivers to have to re-join the group at different nodes. As explained in Section IV-B, this may result in duplicate packets being sent to those receivers. To characterize the overhead, we conduct another experiment based on the MCI topology with all routers being REUNITE-aware. As before, there are 8 senders (or groups) and 64 receivers randomly placed. Each receiver joins and leaves the group based on an on-off process, where the active and inactive periods are exponentially distributed with means of 25 sec and 5 sec, respectively. This rather dynamic scenario is meant to stress test the algorithm. To gauge the overhead of the REUNITE protocol we compute AR over ten independent trials. The resulting average AR value is less than 1.06. Thus, REUNITE loses less than 6 % in efficiency, as compared to an ideal multicast protocol that uses the same distribution trees. In addition, the measured MR is no larger than 3. This shows that there are no significant hot-spots in the network.

VIII. RELATED WORK

In [15], a scheme was proposed to achieve similar state reduction at non-branching nodes as REUNITE. However, it requires dynamically setting up tunnels between adjacent branching routers in a multicast tree. Using an additional layer of IP header introduces 20 more bytes overhead in each header and also may result in packet fragmentation. In addition, to support dynamic membership, a sophisticated and complex control protocol is needed to dynamically set up and tear down tunnels. In contrast, REUNITE achieves the state reduction without the need for tunnelling.

The tree maintenance protocol in REUNITE exhibits similarities to other tree based protocols [2], [4], [6]. However, each new branch of the data delivery tree in REUNITE is constructed based on the *forward direction unicast* routing towards the receiver. In contrast, with other protocols, the data delivery tree is constructed based on the *reverse direction unicast* routing towards the sender.

Simple [5] and EXPRESS [6] augment the multicast class D address with a unicast address of either the *core* or the *sender* respectively. This eliminates the address allocation problem and provides support for sender access control. In contrast, REUNITE goes one step further and eliminates the class D address altogether. Using only one unicast address to identify the group makes it possible to provide additional features, such as reduced forwarding state, native incremental deployability, load balancing, and graceful degradation.

Our mechanism to provide support for multiple senders is similar to the session relay mechanism proposed in EXPRESS [6]. Unlike EXPRESS however, our solution does not require an application level layer or IP encapsulation for unicasting packets from a sender to the root.

IX. Conclusion

In this paper, we propose a novel approach, called REUNITE, that supports multicast service based on *recursive unicast* in IP networks. To the best of our knowledge, REUNITE is the only IP multicast protocol that uses only unicast addresses for both multicast forwarding and group identification. All other IP multicast protocols need class D addresses. By using recursive unicast to support multicast, REUNITE achieves many unique advantages. First, it does not require non-branching routers to maintain per group forwarding state. In addition, it is the only protocol that provides native support for incremental deployment, load balancing and graceful degradation when there are hot spots. To work in a network with unicast-only routers, all existing IP multicast solutions need to use tunnels. In addition, none of the existing solutions can recover gracefully from a scenario when a multicast request is made to a router that has run out of multicast forwarding table entries. We note that a more complex version of REUNITE that eliminates control state is discussed in [9].

A direction for future work is to study how to implement address aggregation in REUNITE to achieve further reduction of forwarding state.

References

[1] S. Deering and D. R. Cheriton, "Multicast routing in datagram internetworks and extended LANs," *ACM Transactions on Computer Systems*, May 1990.

[2] Ballardie A, "Core based trees (CBT) multicast routing architecture," Sept. 1997, RFC-2201.

[3] D. Estrin, D. Farinacci, V. Jacobson, C. Liu, L. Wei, P. Sharma, and A. Helmy, "Protocol independent multicast – dense mode (pim-dm) : Protocol specification," *Work in Progress*.

[4] D. Estrin, D. Farinacci, A. Helmy, D. Thaler, S. Deering, M. Handley, V. Jacobson, C. Liu, P. Sharma, and L. Wei, "Protocol independent multicast – sparse mode (pim-sm) : Protocol specification," Jun. 1997, RFC-2117.

[5] R. Perlman, C. Lee, T. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green, "Simple multicast: A design for simple, low-overhead multicast," Internet Draft, Internet Engineering Task Force, Mar. 1999, *Work in progress*.

[6] H.W. Holbrook and D.R. Cheriton, "IP multicast channels: EXPRESS support for large-scale single-source applications," in *Proceedings of ACM SIGCOMM'99*, Cambridge, Massachusetts, Aug. 1999.

[7] M. Handley, "Session directories and internet multicast address allocation," in *Proceedings of ACM SIGCOMM'98*, Vancouver, BC, Canada, Sept. 1998.

[8] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoglu, D. Estrin, and M. Handley, "The MASC/BGMP architecture for inter-domain multicast routing," in *Proceedings of ACM SIGCOMM'98*, Vancouver, BC, Canada, Sept. 1998.

[9] I. Stoica, T.S.E. Ng, and H. Zhang, "REUNITE: A recursive unicast approach to multicast," Tech. Rep., Carnegie Mellon University, Dec. 1999, http://www.cs.cmu.edu/˜hzhang/multicast/.

[10] P. Ferguson and D. Senie, "Network ingress filtering," Jan. 1998, RFC-2267.

[11] C. Perkins, "IP mobility support," Oct. 1996, RFC-2002.

[12] S. Deering, "Host extension for IP multicasting," Aug. 1989, RFC-1112.

[13] W. Fenner, "Internet group management protocol, version 2," Nov. 1997, RFC-2236.

[14] UCB/LBNL/VINT, "Network simulator, ns (version 2)," *http://www-mash.cs.berkeley.edu/ns*, 1999.

[15] J. Tian and G. Neufeld, "Forwarding state reduction for sparse mode multicast communication," in *Proceedings of INFOCOM'98*, San Francisco, California, Mar. 1998.