November 21, 2005

RT0633
Network 13 pages

Research Report

Recursive Peer-to-Group Routing on a Network Hierarchy for Stable Overlay Multicast

Shu Shimizu, Taiga Nakamura, Ryo Sugihara, Ken Masumitsu

IBM Research, Tokyo Research Laboratory
IBM Japan, Ltd.
1623-14 Shimotsuruma, Yamato
Kanagawa 242-8502, Japan

Research Division
Almaden - Austin - Beijing - Haifa - India - T. J. Watson - Tokyo - Zurich

Recursive Peer-to-Group Routing on a Network Hierarchy for Stable Overlay Multicast

Shuichi Shimizu, Taiga Nakamura, Ryo Sugihara, Ken Masumitsu
IBM Japan, Tokyo Research Laboratory
1623-14 Shimotsuruma, Yamato-shi, Kanagawa 242-8502 Japan
Email: {shue,taiga,sugiryo,masu}@jp.ibm.com

***Abstract*— This paper presents a new application-layer multicast technique, *recursive peer-to-group routing*, in which data delivery paths are not fixed but change dynamically and frequently even when no failures occur at receiver hosts. The advantages of our technique include (1) well-distributed and well-balanced network-relaying traffic for all hosts over the delivery system, (2) automatic and rapid adaptation of delivery routes to the non-uniformity and temporary fluctuations in host performance, and (3) stability of the whole system even with frequent joins and departures of receiver hosts. We exploit a network hierarchy for system scalability just as in the conventional fixed path approaches, but we also introduce layered destinations without assigning any fixed hierarchical roles to the host nodes. In addition, the determination of the multicast paths is decentralized and distributed to the end nodes. Link stresses for uplinks and downlinks are almost the same as for the original data rate at any hierarchical level, which localizes and reduces network traffic. We give an analysis of latency and loss rates with concatenated queuing systems and show how to control the final loss rate by determining such system parameters as data rate, network performance, and recovery rate of erasure code words. We also present and discuss some simulation results with a blocking system, in which the data rate is maximized at the point of the highest system utilization.**

I. Introduction

Multicast is an efficient infrastructure for push-style data delivery such as streaming of digital media and bulk data transfer to a large number of receivers. It shares network resources to avoid duplicating the same data on delivery paths to multiple receivers. Network-layer multicast or IP Multicast [11], which mainly reduces router usage, was proposed more than fifteen years ago, but it has not been widely deployed yet due to such issues as scalability, network management, and higher layer support of error and congestion controls [10].

To address this situation, several application-layer multicast protocols have been proposed as new architectural alternatives to network-layer multicast. They don't depend on network devices such as routers but rely on end hosts that may also be receivers of the transmitted data. For example, Scribe [18] is a publish-subscribe application built on top of Pastry [17], which builds a multicast tree using some of the underlying hosts, each of which has a unique ID and remains active during the broadcast service. Bayuex [21] is built on top of Tapestry [20], in which the hosts that forward data to the others are not necessarily receivers, similar to Pastry. They mainly focus on location mapping to find destinations based on a distributed hash table (DHT) that maps names to addresses. According to Castro et al. [8], tree building approach (Pastry-style) outperforms flooding approach (CAN-style [16]) in terms of several criteria including delivery delay and link stress. Yoid [12] defines a protocol for building a multicast tree by distributed end hosts. ALMI [15] uses a centralized algorithm to build a minimum spanning tree. Narada [10] builds a mesh connecting end hosts and then constructs a shortest path spanning tree on it. These protocols mainly focus on building "good" trees in terms of decreasing the latency from a root to the end hosts, but they may frequently stop for a highly dynamic population of receiver hosts, since they rely on a single path (a *stability* issue). In Overcast [13], data distribution trees adapt to changing network conditions by tracking the global status of the trees. Yang [19] proposes a proactive approach to pre-compute alternative paths as a recovery plan. PRM [4] discusses a multicast data recovery scheme that has both a proactive and reactive component.

Some protocols use multiple paths to address the stability issue. Splitstream [7] stripes data across a "forest," in which the forwarding load is distributed among all hosts. Bullet [14] proposes overlay "mesh" rather than a tree for higher bandwidth and reliability. NICE [3] exploits the network hierarchy so that it scales better than the flat mesh-based approaches (a *scalability* issue), but it still takes a long time to repair paths, on the order of 30 seconds, mainly because it depends on probing hosts over the network, which is time-consuming. CoopNet [9] combines multiple paths and source coding technique to make "live" streaming applications robust against node failures.

We propose a new routing approach for multicast on an overlay network, in which (1) multicast paths are not fixed to a single or few trees but change frequently to avoid reconstruction overhead as membership dynamically changes; (2) delivery paths are automatically determined so that poorly performing hosts are unlikely to appear in the middle of paths resulting in more reliable delivery; (3) path determination is not centralized at a root but distributed to the end hosts for scalable operations; and (4) network traffic is localized recursively in a network hierarchy to avoid duplicating the same data for scalable delivery. We also analyze the latency and packet loss rate of our approach.

The body of this paper is organized as follows. In Section II, we present an overview of *recursive peer-to-group routing* and its important characteristics. Then, in Section III, we present

formal protocols and procedures with pseudocode that implements our approach. In Section IV, we present a performance analysis of latency and error rates. Then, we summarize and compare important characteristics with a simple tree-based approach, followed by consideration of some open issues in Section V, and finally conclude in Section VI.

II. Overview of Recursive Peer-to-Group Routing

In this paper, we assume that the network consists of a single distribution server (or root) and multiple receiver nodes. The overlay multicast system conveys data packets from the root to the receivers with no dedicated help from the routers for relaying. Receivers can join or leave the system at any time, and they contribute to the system only when they also want to receive the packets.

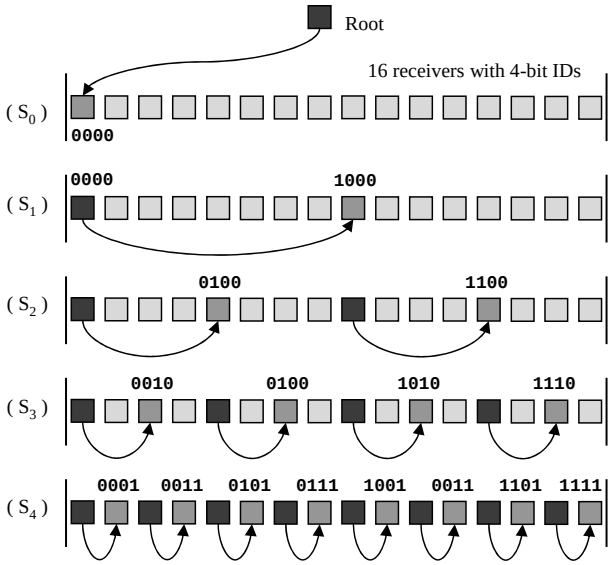

Fig. 1. Scalable overlay multicast by introducing hierarchy of node IDs. Data propagation is divided and depicted for each stage, S_0, S_1, ..., or S_4.

Figure 1 shows a simple and scalable overlay multicast system with 16 receiver nodes, each of which is assigned a unique 4-bit ID (0000, 0001, ..., and 1111). In this example, a root sends a data packet to the node with ID 0000 at the stage S_0, and the node 0000 forwards the packet to the node 1000 at the stage S_1. The packet is further forwarded from 0000 and 1000 to 0100 and 1100, respectively, at the stage S_2. The forwarding operations continue in a recursive way, and eventually all nodes receive the packet by stage S_4 with no duplicates. Note that the forwarding operations are not necessarily synchronized at each stage. Seen from nodes, the node 0000 relays to four nodes, while half of the nodes such as 0001 and 0011 do not relay but only receive, which is a typical tree-based approach.

Our idea comes from this naive propagation method based on the hierarchy of node IDs, but it aims at stable multicasting under dynamic membership changes, with decentralized and scalable operations and well-balanced network traffic. In the following sections, we present our approach and its characteristics.

A. Path-varying Approach for Stable Multicast

When propagation is fixed on a single path and one or more upstream nodes leave the multicast system, some downstream nodes will fail to receive the following packets, and thus path recovery is required to continue the multicast. Reactive path recovery calculates the next available path after detecting the failures, and proactive path recovery prepares the alternative paths before failure [4], [19]. Unlike the conventional methods, our approach is *path-varying propagation*, which is neither reactive nor proactive. Due to the frequent path changes, our *path-varying propagation* quickly works around fault situations, which turns out an implicit error recovery.

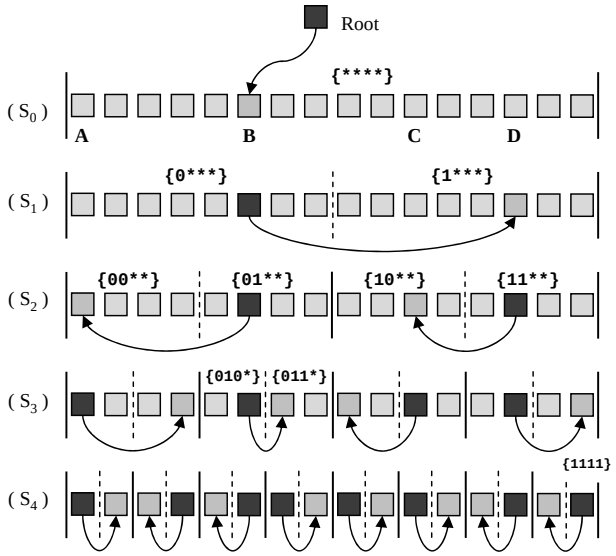

Fig. 2. Path-varying propagation, in which any relaying node (in dark squares) chooses a next destination out of its adjacent group in each layer.

Before discussing the path-varying propagation, we first define *layered groups*, which are essential for complete propagation with no loops nor duplicates. In the example (Figure 2), the largest node group contains all of the IDs, and is denoted as $\{****\}$. The second largest groups are denoted as $\{0***\}$ and $\{1***\}$, which comprise the IDs that begin with prefix 0 and 1, respectively. These two groups are strict subsets of the first group $\{****\}$. Smaller groups, $\{00**\}$, $\{01**\}$, and so on, are defined similarly, up to $\{1111\}$, which contains only the ID 1111. We call these segmentations the layered groups. Each receiver node belongs to multiple groups: exactly one group in each layer. For example, the node with an ID 0100 belongs to the layered groups $\{0100\}$, $\{010*\}$, $\{01**\}$, $\{0***\}$ and $\{****\}$.

The path-varying propagation works for each data packet as follows. First, at the stage S_0, the root chooses a node from the group $\{****\}$ (i.e., the group including the entire node sets) and sends it a data packet. In the example of Figure 2, the node 'B' (ID:0100) received the packet. Then, 'B' chose

the next node 'D' out of its adjacent group $\{1***\}$ to forward the packet. Next, 'B' and 'D' forwarded the packet to their adjacent groups, $\{00**\}$ and $\{10**\}$, respectively at the stage S_2. Those forwarding operations continue in a recursive way using smaller groups, until the packet finally propagates to all receivers at the stage S_4. There are no loops of propagation nor duplicate arrivals of packets. Each relaying node routes data packets not to a fixed node but to one of the nodes in the group. Since it can be seen as a unicast from a peer node to the entire group, we call this *peer-to-group routing*. The path-varying propagation is thus composed of recursive peer-to-group routings.

Each time a sender chooses the next node from the *available* and *active* candidate nodes. *available* nodes are the ones currently in the service, and *active* nodes are the ones which are not busy and ready to receive the next packet. To do this, senders only send the next packet to those who have returned *acknowledgments* for the previously sent packets. This makes the entire multicast system stable, because upstream paths favor the active (or high performance) hosts while inactive (or lower performance) hosts automatically wind up to the final end of paths.

For the next packet in the example, the root may choose 'D' as the first node and 'D' forwards the packet to its adjacent segment $\{0***\}$. From the viewpoint of propagation paths, the node 'B' is upstream relative to the node 'D' in propagating the first packet, but it is downstream relative to 'D' for the next packet. This indicates that the path-varying propagation is constructed by the *role-varying* behavior in each node. We discuss more about the advantages of the role changes in the following sections. It should be noted that path/role-varying propagation obviously inherits the important characteristics of the fixed path approaches — scalability and complete delivery with no loops nor duplicates — as mentioned above.

B. Decentralized Path Determination

In the path-varying propagation, the root does not determine the entire path but only chooses the first node. The rest of the paths are determined at each relaying node and eventually they are fully constructed for each packet. The behavior of the relaying nodes varies according to both their node IDs and the stages in which they receive data packets.

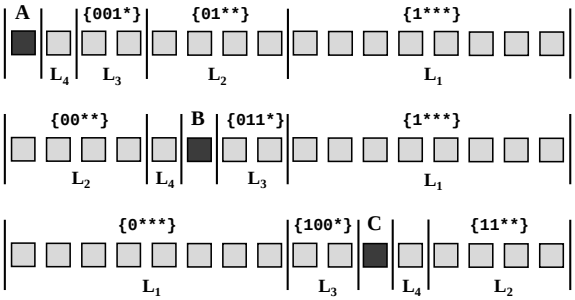

Fig. 3. The forward-to groups: layered groups for the forwarding operations from a node.

Figure 3 shows the layered groups for the forwarding operations from the node 'A', 'B' and 'C'. When 'A' receives a data packet from the root, it then forwards the packet to the layered groups, $\{1***\}$, $\{01**\}$, $\{001*\}$ and $\{0001\}$, while 'C' forwards to $\{0***\}$, $\{11**\}$, $\{100*\}$ and $\{1011\}$ in the same situation. We call these layered groups for each node *the forward-to groups* and also refer these as L_1, L_2, and so on from the largest group, respectively. When a node receives a data packet at the stage S_k ($k \geq 0$), then it is responsible for relaying the packet to its forward-to groups of L_{k+1}, L_{k+2}, ..., and L_n, where n is the maximum level of stages (or the number of the forward-to groups), and n is linear to the logarithm of the total number of receiver nodes. These context-based forwarding operations assure our multicast system has no propagation loops and no duplicate packet arrivals.

The stage level in which a node has received a data packet from a *source node* is determined as follows. The node compares its own ID with the ID of the source node beginning from the highest bit and finds the first bit that doesn't match. The mismatched bit position indicates that they belong to the different groups from that point. For example, when the node 'A' (ID:0000) receives a packet from the node 'B' (ID:0100), their IDs mismatch at the second bit, and thus we find the stage is S_2 and 'A' is responsible for forwarding the packet to its L_3, L_4, ..., and L_n in turn. Thus, each packet does not have to contain any information about the stages.

C. Distributed Network Traffic

As any receiver node always receives data packets with no duplicates, the incoming data rate at the receiver's downlink, *the downlink stress*, is the same as the original data rate. This is also true for any level of the layered groups, as shown in Figure 2, and thus the downlink stress at any layered group is the same as the original data rate. For example, when a layered group topologically corresponds to such a network segment that a router manages, it receives no more or less than a single set of data packets even when the segment actually includes a number of active end hosts.

On the other hand, the outgoing data rate at the node's uplink, *the uplink stress*, ranges from zero to n times the original data rate in the fixed path approaches. For example, the node 0000 in Figure 1 always forwards data packets to four peer nodes, which requires four times more bandwidth than in its downlink, while half of the nodes just receive but never forward any packets, that is, their uplink stress is zero. This unbalanced situation in uplinks would not meet the requirements of asynchronous channels such as DSL.

In the role-varying propagation, one of its advantages is to average the uplink stress. The dark squares in Figure 2 indicates the nodes that forward a data packet at their uplinks. Their roles will change for the consecutive packets. When all nodes (i.e., all IDs) are available and they have the same activities, then their uplink stress approaches $(1 - 1/2^n)\lambda$ as the time range increases, where λ is the original data rate and n is the number of the forward-to groups. Thus, each node has λ-in and almost the same value of λ-out.

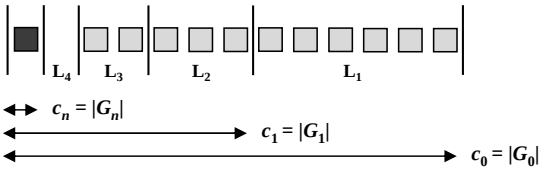

Fig. 4. The situation when receiver nodes are not fully available in each group. The dark square node belongs to the layered groups, G_n, ..., and G_0, and the numbers of nodes in each group are c_n, ..., and c_0, respectively.

In the more general case when some of the layered groups contain no *available* nodes, then the averaged uplink stress U at a certain node is calculated by

$$U = \lambda \sum_{k=0}^{n-1} \frac{1}{c_k}, \tag{1}$$

where c_k is the number of available nodes in the layered group G_k that is complementary to $L_1 \cup \cdots \cup L_k$, as shown in Figure 4, and thus $1/c_k$ indicates the probability that data packets are relayed to the forward-to group L_k from a node in G_k. The uplink stress at the complementary group G_i ($0 < i \leq n$) is also calculated by

$$U_i = \lambda \sum_{k=0}^{i-1} \frac{c_i}{c_k}, \tag{2}$$

which is the summation of the data rates emitted from all of the nodes in G_i to L_i, ..., and L_1. Note that $U_n = U$. The uplink stress U and U_i may exceed λ in such situations when the nodes are not well distributed against the hierarchy of node IDs, for example, if $c_{k-1} = c_k + 1$, which indicates that the population is mostly concentrated in small groups, not uniformly distributed. However, when we have enough nodes available, then we can expect well-distributed situations, and thus we also expect λ-in and less than λ-out traffic at all nodes and layered groups. In such situations, when the hierarchy of node IDs matches the actual network topology, network traffic is well localized and decreases at the routers, as the downlink and uplink stresses directly indicate the number of packets at the routers.

The peak uplink stress is still $n\lambda$, which may be very high and would lengthen the latency of packet arrivals at the terminal hosts in the paths. However its probability is very low and so it does not always impact the performance of the delivery systems. We will discuss and analyze the latency issues in Section IV.

D. Node Join

While the hierarchy of node IDs never changes during a multicast service, the availability (joined or not) of nodes may frequently change. This is an issue of system stability under dynamic membership changes. Before discussing the stability issue, we start by describing procedures required for the root to accept a new receiver node. When a new receiver node wants to join the multicast service, it sends a "JOIN" request to the root and waits for initial information about peer nodes for each of its forward-to groups. The procedures for accepting each request at the root is as follows:

i. Get the ID of the new receiver node,
ii. Register the node,
iii. Calculate its forward-to groups according to the ID,
iv. Select destination candidate nodes for each group,
v. Inform the new node of the candidate nodes.

These steps are performed in a one-to-one way between the root and the new node. Since the receiver nodes are maintained in a hierarchical way, and a constant or limited number of *destination candidate nodes* are randomly selected for each of the forward-to groups, even when many more nodes are available, the number of steps at the root for each request is only $O(\log N)$, where N is the total number of available nodes. In order not to disrupt the forwarding operations, the new receiver should not become a forwarding node until it is ready to send, that is, until it gets a complete list of its destination candidates. Otherwise, incomplete paths would be built towards old nodes.

The steps at the new receiver side are as follows.

i. Send a "JOIN" request to the root,
ii. Wait for the initial information sent by the root,
iii. Set up the forward-to groups,
iv. Inform the root of the completion of setup, and
v. Announce itself to the other nodes.

New receiver nodes are not informed of all peer nodes but only of the limited number of nodes for their destination candidates, and so they maintain paths to those candidates in a scalable and stable way. After informing the root of the completion of setup, new nodes announce themselves to the other available nodes by using the recursive peer-to-group routing as well as for data packets, which is also scalable. The announcement may cause updates of the destination candidates in the already running nodes, as will be described in Section III-D.

E. Node Departure

Since node departure will break propagation paths in the overlay network, it sometimes results in packet loss in the conventional fixed path approaches. In contrast, in our path-varying approach, simple cleanup procedures performed by each departing node avoid the packet loss, without rebuilding paths, as follows. When a node is leaving the multicast service, (1) it should notify its source nodes of the disconnection so that they can remove it from their destination candidates and so they will send no more packets to the leaving node; (2) It should also complete forwarding any unprocessed packets. We call this situation *graceful departure*, and there is no chance of losing packets in the multicast system as long as nodes leave gracefully.

When a running node stops unexpectedly due to unpredictable problems such as a system crush, one or more packets stored in the stopped node may be lost. To avoid this situation, its source nodes can resend those packets to the other candidate nodes when the disconnection of the stopped node is detected.

If no explicit sign is given of the disconnection, the sender can watch for a timeout of the receipt acknowledgments. We call this *local error recovery*. However, this may not reduce the loss rate to zero, and so we need loss recovery at the receiver side, such as forward error correction (FEC) [5] or multiple description coding (MDC) [9]. Note that we should not rely on recovery requests sent to the root, because this may cause scalability problems.

III. Protocols and Procedures

Now, we define the recursive peer-to-group routing more formally. The pseudocode of the peer-to-group routing so that a data packet d is forwarded from a source node p to a group G is as follows:

P2G(p, G, d)
1. $g \leftarrow$ CHOOSE(G),
2. Trigger FORWARD(p, g, d),

where the first step chooses an active node that is ready to accept the next packet from the layered group G and then marks the chosen node g as inactive for the source node p. The node g will become active again when the source node p receives an acknowledgment from g. The second step sends the data d to g, and then it asynchronously starts FORWARD(p, g, d) at the receiver g's side and immediately returns. The first step may block until one or more nodes become active in the group G. We call this a *blocking* version of peer-to-group routing. When the data rate is not fixed beforehand, this version can maximize it according to the minimum performance of all receiver nodes.

An alternative implementation of the peer-to-group routing is a *queuing* version, as follows.

P2G(p, G, d)
1. $Q \leftarrow$ QUEUEOF(G),
2. ENQUEUE$([p, d], Q)$,

where the first step gets the queue that is bound to the forward-to group G, and the second step puts a pair of p and d into the queue Q and immediately returns with no blocking. The actual forwarding operations are performed in a separate asynchronous process that is bound to the group G in each node, as follows.

SERVICE(G)
1. $Q \leftarrow$ QUEUEOF(G),
2. $[p, d] \leftarrow$ DEQUEUE(Q),
3. $g \leftarrow$ CHOOSE(G),
4. Trigger FORWARD(p, g, d),
5. Go to the first step,

where the second step retrieves the pair of p and d from the queue Q bound to the group G. The second and third steps may block. Each node has n SERVICE processes for its forward-to groups, and they run concurrently and asynchronously.

A. Packet Propagation

The root r maintains a set S of available (i.e., currently running) receiver nodes. The pseudocode for the root to deliver a data packet d is as follows.

ROOTDELIVER(d)
1. P2G(r, S, d).

The pseudocode for the node g to forward the data packet d is recursively defined as follows.

FORWARD(p, g, d)
1. If p is the root r, then $k \leftarrow 0$,
   Else $k \leftarrow$ (mismatched bit position between the IDs of p and g),
2. For $i = k$ to $n - 1$, do P2G(g, L_{i+1}, d),
3. UPDATE(g, d) if d is a node info,
4. PROCESS(g, d) if d is a data packet,
5. ACKNOWLEDGE(p, g),

where n is the number of forward-to groups for the node g, the mismatched bit position ranges from one to n, and L_i is the i-th forward-to group from g. When d is a node notification, then the third step updates the destination candidates of g, if necessary, which will be described later. When d is a data packet, then the fourth step processes it in the receiver node g. For example, error recovery operations may be applied in a buffer of codewords prior to sending it to outer application systems such as a media player for streaming data. As the data packets do not necessarily arrive in the original sequence, the buffer is also used for sorting them. Finally, the fifth step sends an acknowledgment from g to the source node p so that p can choose g again as an active node.

B. Join

When a new receiver node joins the multicast service, it first sends a "JOIN" request to the root, and gets the initial information about its destination candidates and builds the forward-to groups, as follows.

NODEJOIN
1. For $i = 1$ to n, do:
   $L_i \leftarrow$ (candidates for the i-th group),
2. Reply to the root,
3. ANNOUNCE itself,

where ANNOUNCE uses the recursive peer-to-group routing as well as data packets, as described in Section III-D.

When the root receives a "JOIN" request from the new receiver node p, the root inserts it into the set S; calculates the destination candidates for p; and finally sends it the list of candidates, as follows.

ROOTACCEPTJOIN(p)

1. $S \leftarrow \{p\} \cup S$,
2. For $i = 1$ to n, do:
   $L_i \leftarrow$ (up to m candidates for p's i-th forward-to group),
3. Notify p of $[L_1, ..., L_n]$,

where m is the predefined number to limit the size of the forward-to groups in each receiver node, so that receiver nodes can maintain a scalable number of peer nodes. The candidate nodes are chosen at random in the second step in order to increase the number of potential propagation paths.

C. Leave

Receiver nodes should leave the multicast service in a graceful way, as follows.

NODELEAVE

1. Notify the source nodes of disconnection, and
2. Flush the unprocessed data packets or time out.

If receiver nodes unexpectedly stop without any notification of disconnection, their source nodes may send some consecutive packets that will be lost. Even in such a case, since the source nodes receive no more acknowledgments, the stopped nodes will not be chosen as active ones after that. The timeout in the second step is for the case when the destination nodes don't respond.

When the root detects the disconnection of the node p, it removes p from the set S and sends a notification for replacement using a node which is located near p so that the running nodes can replenish their destination candidates, as follows.

ROOTDETECTLEAVE(p)

1. $S \leftarrow S - \{p\}$, and
2. $q \leftarrow$ (choose a node hierarchically near to p),
3. Tell q to ANNOUNCE itself.

Receiver nodes have two or more destination candidates for each of their forward-to groups, but the number of candidates are limited up to m for scalable operations. If all candidate nodes bound to a certain forward-to group that potentially include other available nodes leave the system and no new nodes join to take their places, then potential paths will be cut off. The above announcement step is to avoid this situation by providing possible candidates.

Each receiver node removes the node p from its forward-to group, if included, when detecting p's disconnection, as follows.

NODEDETECTLEAVE(p)

1. For $i = 1$ to n, do $L_i \leftarrow L_i - \{p\}$.

Note that the removal of p from L_i succeeds no more than once out of $i = 1, ..., n$, because the forward-to groups are exclusive of each other for all nodes.

D. Announce and Update

The announcement also exploits the recursive peer-to-group routing, as follows.

ANNOUNCE(p)

1. For $i = 1$ to n, do P2G(p, L_i, p),

where the operations are performed in the p's process.

Each receiver node updates its forward-to groups when it is notified of a new peer node c by an announcement, as follows.

UPDATE(p, c)

1. For $i = 1$ to n, do:
   If c is potentially in L_i of the node p and $c \notin L_i$ and $|L_i| < m$, do $L_i \leftarrow \{c\} \cup L_i$.

When the node p already maintains the announced node c or has enough destination candidates for the corresponding forward-to group, it ignores the announcement, so that each forward-to group holds up to m destination candidate nodes.

E. Implementation Notes

1) Explicit Hierarchy: Since IPv6 addresses except for the lower 64 bits (used for storing the interface ID) indicate the hierarchy of the networks [1], which is one of the main objectives of the design of IPv6, the hierarchical and recursive peer-to-group routing should work especially well for IPv6 networks, by making use of the IPv6 addresses as the IDs. The higher bits in IPv4 addresses may also indicate the hierarchy of well designed networks, but the lower bits will not in most cases. For example, the subnet 10.1.2.0/24 may not be close to the subnet 10.1.1.0/24, although they are very close to each other in their network addresses, as shown in Figure 5.

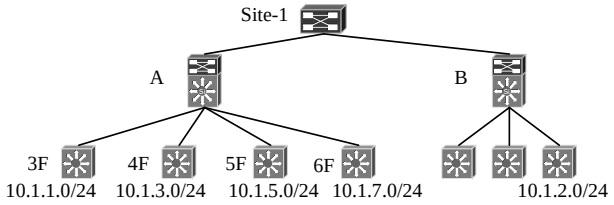

Fig. 5. Hierarchy example of IPv4 networks with two buildings and seven subnets. The IPv4 addresses do not necessarily represent the hierarchy of networks.

In order to reduce the network traffic over routers by localizing the traffic of relayed packets within each subarea, it is better for the forward-to groups in each node to correspond to the actual hierarchical structures. Figure 5 shows an example of such explicit hierarchical groupings, in which hierarchical groups do not correspond to network addresses with prefixes but do represent physical subareas such as the 4th floor and Building "B". This may no longer be a binary tree, but the relaying operations are almost the same as in binary trees. When the node of 10.1.1.23 in the 3rd floor receives a data packet as a representative of its site, then it relays the packet

to one of the destination candidates for Building "B", one for the 4th floor, one for the 5th floor, one for 6th floor, and so on.

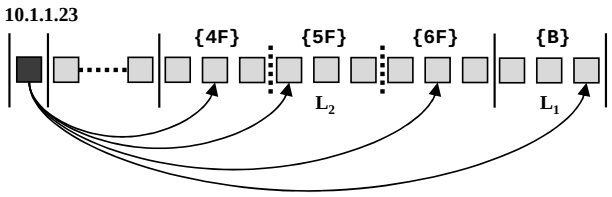

Fig. 6. Non-binary forward-to groups. The number of destinations is one or more in each group.

Because the higher bits in the IP addresses of the nodes include no actual hierarchical structure in this case, the bootstrap (join) and announcement operations should be slightly modified. At bootstrap time, each new node must be explicitly informed of the hierarchical line to which it belongs (e.g., the lowest 8 bits of the IP address, "3F", "A" and "Site-1"), the forward-to groups for each level of the line (e.g., $\{$B$\}$ for "Site-1", $\{$4F$\}$, $\{$5F$\}$ and $\{$6F$\}$ for "A"), and the initial destination candidate nodes for each group. Next, the new node sends its hierarchical line information to its destination candidate nodes so that they can determine the levels of the new source node, as seen from their location. For example, suppose that a destination node belongs to the line of "Site-1", "A" and "4F", which is different from the line for the node 10.1.1.23, then the both lines do match up to "A", but do not match at the next level "3F" and "4F". Therefore the destination node determines that the new source comes from the level of "4F" and all packets received from the node should be relayed only within "4F".

Also, when a new node is announced, its hierarchical line information must be carried with its IP address so that already running nodes can determine which group it should belong to as a destination candidate. For example, when a new node with the line of "Site-1", "B" and "1F" is announced, the node of 10.1.1.23 in Figure 6 may update the destination group "B" to insert it with or without removing an old one. Thus, the amount of information increases for the bootstraps and announcements in handling an explicit hierarchy, but the data packets still do not need to contain any hierarchical information in their control channel such as in the packet headers.

2) Shared LAN: Micro-segmentation at the end switches makes possible good performance of the peer-to-group routing, since it builds exclusive paths between any pair of end nodes and there is no interference occurring with any pairs, as long as the total number of packets does not exceed the capacity of the switch. On the other hand, for a shared LAN, such as a legacy 10BASE-T network with repeater hubs, the network traffic may explode because of the exchange of data packets and may reach the maximum capacity of the network, because it is not localized but linear to the square of the number of end nodes. In order to avoid this situation, the recursive peer-to-group routing should stop when data packets arrive at any nodes in the shared LAN, and there the packets should be shared by using local multicast systems such as the link-layer multicast of Ethernet.

IV. Performance Analysis

In this section, we analyze and simulate packet loss rate and latency in the multicast system based on the recursive peer-to-group routing, and we show how to design a reliable and stable multicast system by determining several system parameters, such as the number of candidates and the recovery rate of erasure codes.

A. Latency in the Queuing Version of P2G

The propagation paths, especially their length, may vary for each data packet in the recursive peer-to-group routing, and so the fluctuation of data arrival time is larger, but the mean of the path length is shorter than for the conventional fixed-path multicast systems.

1) $G/M/m$ Queuing Model for P2G: We assume that we have the full number of 2^n available nodes in an n-stage recursive peer-to-group routing system, as shown in Figure 2, to simplify the following discussion, which leads to the worst case analysis. When the number of total nodes is less than 2^n, the propagation paths become shorter, which reduces the propagation time and the total system time from the root to receiver nodes. When a node directly receives data packets from a root, it is at the 0-th stage and this happens once out of 2^n times on average. When a node receives packets at the k-th stage ($1 \leq k \leq n$), then the arrival rate is $\lambda/2^{n-k+1}$, where λ is the data rate that the root produces. For the forwarding operations, each node is responsible for relaying a data packet to $(n-k)$ destination nodes with a probability of $1/2^{n-k+1}$.

Each forwarding operation corresponds to the SERVICE procedure mentioned in Section III. The destination nodes in the group G become active after they receive the data and complete their own PROCESS in the FORWARD procedures. Since the sending operations in the fourth step of SERVICE are performed under $(n-k)$ overlaps with a probability of $1/2^{n-k+1}$, the time distribution for when the destination nodes receive the data shows exponential characteristics. We assume that the time required for the PROCESSing of the data is negligibly small in comparison with the time for transferring the data, and thus we model each peer-to-group routing by using a $G/M/m$ queuing system, in which we also assume that the inter-arrival times are i.i.d. (independently and identically distributed) with general distribution; that there are m parallel servers that relay data to next nodes; and that their service times are i.i.d. with an exponential distribution and any data transmission takes exponential time with rate μ. Each stage has different arrival rates, but we suppose a unique arrival rate λ with the same process, $\mathrm{P}[X \leq t] = F_X(t)$ for the inter-arrival time X, at each stage in order to simplify the discussion.

2) K-Step Propagation Paths: The number of relaying nodes or steps in the propagation paths ranges from one to n, except for the step from the root to the first node. Note that

the steps indicate the relaying nodes, while the stages indicate the levels of the forwarding operations. The k-step propagation paths are followed with a probability of $\binom{n}{k}/2^n$ for each data packet. The number of steps k ranges from zero to n, and the zero step indicates no waiting, since the waiting time from the root to the first node is negligibly small when the root has a number of candidates for choosing from as the first node. Thus, the total waiting time distribution when a data packet waits for more than t time units in the n-stage recursive peer-to-group routing system is calculated by weighing the k-step $G/M/m$ queuing system with $\binom{n}{k}/2^n$, as follows:

$$F_W^c(t) = \mathrm{P}[W > t] = \frac{1}{2^n}\sum_{k=1}^{n}\binom{n}{k}F_{W_k}^c(t), \qquad (3)$$

where $F_{W_k}^c(t)$ is the total waiting time distribution for the concatenation of k queuing systems waiting for more than t time units. This is calculated as

$$F_{W_k}^c(t) = \sum_{i=1}^{k}\binom{k}{i}\left(1-F_{W_1}^c(0)\right)^{k-i}F_{W_1}^c(0)^i \times Q(i-1; m\mu(1-\beta)t), \qquad (4)$$

where β is the characteristic root with regard to the inter-arrival process[1], $Q(k;t)$ is the cumulative Poisson distribution[2], and $F_{W_1}^c(0)$ is the probability that an arriving packet waits for more than zero time units in a single queuing system[3]. It should be noted again that the above calculation defines an upper bound to the actual waiting time due to the assumption that the system is full of available nodes, and that the inter-arrival rate λ is higher than the actual rates.

The total service time follows the *hyper* n-step Erlangian distribution, in which $(k+1)$-step paths are weighed by $\binom{n}{k}/2^n$ for $0 \le k \le n$. The probability distribution that a data packet stays more than t time units in the system is calculated by

$$F_S^c(t) = \mathrm{P}[S > t] = \frac{1}{2^n}\sum_{k=0}^{n}\binom{n}{k}Q(k;\mu t). \qquad (5)$$

[1]The characteristic root β satisfies $\beta = f_X^*(m\mu(1-\beta))$, where $f_X^*(s)$ is the Laplace-Stieltjes transform of $F_X(t)$.

[2]The cumulative Poisson distribution is

$$Q(k;t) = \sum_{i=0}^{k}\frac{t^i}{i!}e^{-t}$$

[3]It is calculated by

$$F_{W_1}^c(0) = \frac{a_m}{1-\beta}.$$

where a_m is the probability that each arrival sees m packets in a stage. It is calculated by

$$a_m = \left[\frac{1}{1-\beta} + \sum_{j=1}^{m}\binom{m}{j}\frac{1}{(1-\gamma_j)C_j}\frac{m(1-\gamma_j)-j}{m(1-\beta)-j}\right]^{-1},$$

where

$$\gamma_j = f_X^*(j\mu) \quad \text{and} \quad C_j = \prod_{i=1}^{j}\frac{\gamma_j}{1-\gamma_j}.$$

The mean service time is $\overline{S} = (n+2)/(2\mu)$, which is half of the simple $(n+2)$-step Erlangian distributions.

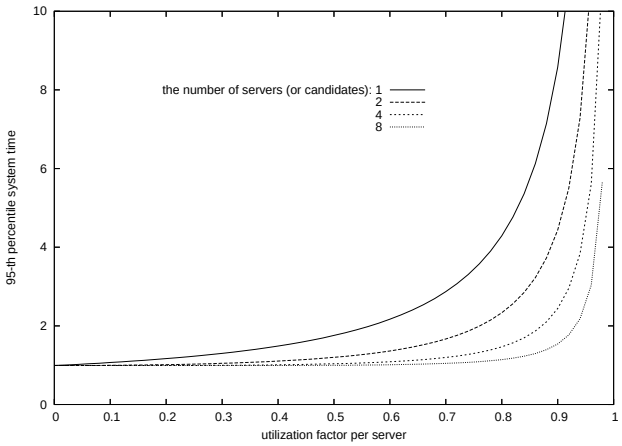

Fig. 7. The 95th-percentile system time of the 8-stage recursive peer-to-group routing system. The $M/M/m$ queuing system is assumed for each P2G routing. The vertical scale is normalized by $(n+2)/\mu$.

3) System Time Distribution of the Queuing P2G: The latency of data packet arrivals is characterized by the system time distribution, which is calculated by the convolution of the waiting time and service time, as follows.

$$F_T^c(t) = \int_0^\infty F_W^c(t-x)f_S(x)dx, \qquad (6)$$

where $f_S(t)$ is the PDF of the service time. Figure 7 plots the 95th-percentile system time against the utilization factor per server, where we assume Poisson arrivals for an 8-stage recursive peer-to-group routing system and use numerical integration. It shows cases for 1, 2, 4, and 8 servers, respectively, where the vertical scale is normalized by $(n+2)/\mu$. For example, $(n+2)/\mu$ time units allow us to receive 95% of the packets under utilization 0.5, when each peer-to-group routing has eight servers (i.e., candidates). Even for a single server, $2(n+2)/\mu$ time units cover 95% of the packets, which is scalable with the total number of nodes as $n = \log N$.

B. Loss and Recovery

Packets that do not arrive in time should be handled as lost packets. In this section, we refer to loss recovery at the client sides when using FEC and present how to control the *unrecoverable loss rate*. The mean of data packet loss for the interval of $[T_0, T_0+L]$ is calculated by

$$l = \int_{T_0}^{T_0+L}F_T^c(t)dt. \qquad (7)$$

Denoting the number of packets in a code block as B and redundant packets as R, the code rate is $\lambda' = \lambda B/(B-R)$. Each receiver client can recover up to R lost packets in a FEC block. Suppose there is a buffer to handle an FEC block ($L = B/\lambda'$) and to hold some extra packets (T_0), then the mean l is calculated for lost packets in the FEC block. Assuming

that packet loss process is completely random and it follows a Poisson distribution, $P(i;T) = T^i/(i!)e^{-T}$, then the final loss rate or unrecoverable loss rate is calculated by

$$\epsilon = \sum_{R<i}^{\infty} P(i;l). \tag{8}$$

as the sum of the possibilities of unrecoverable packets.

TABLE I
THE LOSS RATES AFTER FEC IS APPLIED.

(B,R)	$T_0 = 0$	$T_0 = 8/\lambda'$	$T_0 = 16/\lambda'$	$T_0 = 32/\lambda'$
(8, 1)	7.7×10^{-1}	6.1×10^{-1}	2.6×10^{-1}	3.0×10^{-3}
(16, 2)	8.7×10^{-1}	5.9×10^{-1}	1.4×10^{-1}	1.3×10^{-4}
(32, 4)	7.4×10^{-1}	2.7×10^{-1}	1.4×10^{-2}	7.3×10^{-8}
(64, 8)	1.9×10^{-1}	1.0×10^{-2}	1.8×10^{-5}	2.6×10^{-15}

Table I shows the loss rates after FEC is used for an 8-stage and 4-server recursive peer-to-group routing system, supposing that the utilization factor per server is 0.6, which means a relatively high load. For example, the unrecoverable loss rate can be less than 10^{-7} (i.e., the column of 7.3×10^{-8}) by introducing a lookahead buffer in which the system holds a 32-packet FEC block with four redundant packets and another 32 recent packets before processing them. Suppose that the original data rate is $\lambda = 16$ [packets/sec], then the latency is $64/\lambda' = 3.5$ [sec], which may be acceptable for one-way broadcast applications.

It should be noted that the loss rate becomes higher when the packet loss occurs not at random but in a burst, and/or when the forwarding processes do not follow the exponential distribution.

C. Latency in the Blocking Version of P2G

One of the alternative implementations is a blocking system which does not queue, but blocks data packets until the downstream nodes are ready to accept them. This is especially desirable when the data rate should be automatically determined by the service rate in the system. In other words, when the data rate should be at the maximum rate that the system offers.

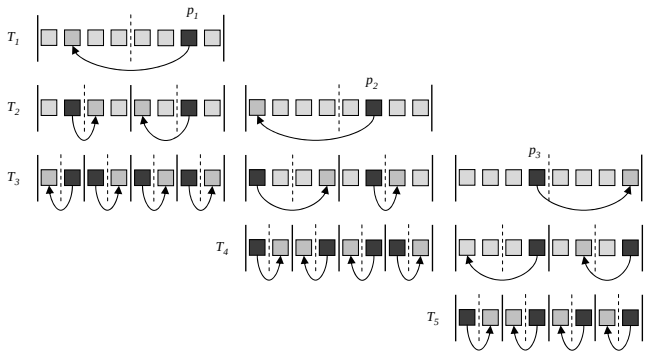

Fig. 8. The forwarding operations synchronized in layers not only for each data packet but also between data packets.

Figure 8 shows a possible situation when the i-th forwarding operations are synchronized in the i-th stages for each data packet and also the stages are synchronized between the data packets with one stage shifted. In each time slot (e.g., T_3), no i-th forwarding operations conflict with each other, and so there should be no large network delays due to congestion in the recursive peer-to-group routing. The nodes send or receive n data packets at maximum, and one packet on the average. The number of stages that each data packet takes ranges from one to $n+1$, and its mean is $n + 1/2^n$, since it takes $k+1$ stages with the probability of $1/2^{n-k+1}$ for $0 < k \leq n$, and one stage with $1/2^n$. In the blocking version, the shift against the previous data packet dominates the data rate, and it is automatically determined by the activities of the receiver nodes.

The simulation results of the blocking version of recursive peer-to-group routing systems are shown in Figure 9. This shows the three cases of 4, 6, and 8 stages with four servers (or destination candidates). The vertical scale indicates the mean system time until the nodes receive the data packets after the root transmits them, and it is normalized by $(n+2)/\mu$, where $1/\mu$ is the mean time for data transfer between nodes. We assume that it follows the exponential distribution. The horizontal scale indicates the data rate at which the root transmits data packets, and it is normalized by $1/\mu$. For the lower data rate, the mean system time is around 0.5, as well as in the queuing version. This gradually increases around the data rate 1.0, and finally it reaches the exponential value for n, that is, it becomes linear to the total number of nodes N, at the maximum data rate that the system offers. This is because the entire system buffers a number of data packets that is linear to N when the data rate is high, and thus the time difference from when the root transmits data packets until the nodes receive them becomes linear to N. The fluctuation of data arrivals is also linear to N. However, this is not a problem for the applications that do not require real-time packet by packet connections, such as file transfers, because they eventually receive all of the data and finish. In addition, when the total number of data packets is less than N, the system time does not exponentially increases but it is scalable against the large N.

The existence of single points of failure remains as a serious drawback. The whole system may stop due to one or more nodes having unexpected problems. The use of multiple candidates does not work well because one or more nodes will be stuck in the final stages and the undesirable situation propagates towards the earlier stages. One of the possible solutions is to remove the troubled groups from the routing when all of the candidate nodes belonging to them have been busy for more than a predefined timeout period. When one or more of the nodes become available again, the groups can be reactivated. The system time or latency may increase when timeout occurs, but this may be acceptable for applications that have no predefined data rates but which need the highest throughput possible, such as file transfers.

TABLE II

COMPARISON TO THE FIXED-PATH APPROACH

	simple fixed tree	P2G
forward traffic	$n\lambda, (n-1)\lambda, ..., \lambda, 0$	$< \lambda$
latency	$1, 2, ..., n+1$	$1 + n/2$
stability	- single point of failures - static tree optimization	- alternative destinations - dynamic tree optimization
scalability	control centralized at a root node - global error recovery (receiver-based ARQ)	control decentralized to receiver nodes - local error recovery (sender-based re-route)
throughput	up to bandwidth	$\propto m/\mathrm{RTT}$

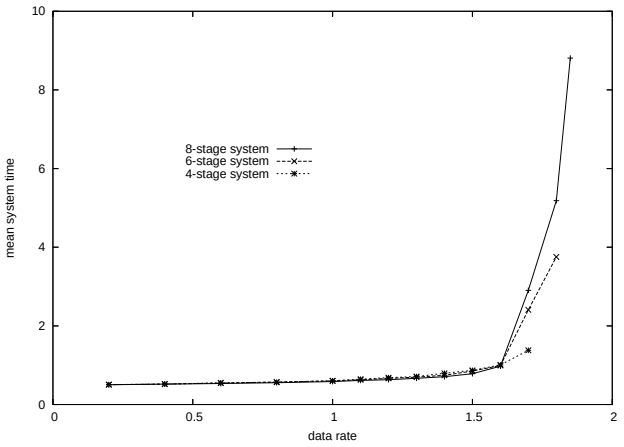

Fig. 9. Simulation results of the blocking version of recursive peer-to-group routing systems. The vertical and horizontal scales are normalized by $(n+2)/\mu$ and $1/\mu$, respectively.

V. NOTABLE CHARACTERISTICS AND DISCUSSIONS

In this section, we summarize and discuss more about the scalability and stability issues and other important characteristics followed by open technical issues. Table II shows the summary of the discussions below. An implementation of the P2G for streaming can be found at the IBM alphaWorks site [2].

A. Stability

As long as each node correctly maintains one or more active destination nodes in all of the forward-to groups, the propagation paths always exist and the delivery succeeds in a stable way. When there are no available nodes under some forward-to groups, they obviously need no destination nodes. The announcement of new receiver nodes and the re-announcement for replenishment of the departed nodes will provide new destination candidates at each node, as described in Section III. Thus, the multiple candidates and announcement have an important role for keeping the propagation stable under dynamic membership changes.

The activities of destination nodes are measured by using acknowledgments that they have returned soon or not. The nodes which take a longer time to return acknowledgments are implicitly treated as lower performers, and they are chosen as the destinations less times. This optimizes the propagation trees in that lower performers do not appear upstream often but stay downstream in the trees most of time. This *dynamic tree optimization* also works when a receiver node's performance temporarily changes, which is very likely to occur because the nodes are not dedicated to relaying data but they have other processes such as reading e-mail.

B. Scalability

Each receiver node maintains $O(\log N)$ peer nodes, where N is the total number of the running receiver nodes, since each of the forward-to groups maintains a constant number of peer nodes and the number of groups for each node to maintain is $O(\log N)$. Each node receives data packets at the original data rate and forwards some of them at almost the same rate. Thus, receiver nodes work in a scalable manner relative to the total number of available nodes.

The announcement operations are also scalable as new receiver nodes initialize them in a decentralized way and the announcement is propagated by using the recursive peer-to-group routing as well as data packets.

The root's acceptance of a new receiver nodes takes $O(\log N)$ time, as mentioned in Section II-D. Choosing a first node is also done in $O(\log N)$. The root node is responsible for maintaining all of the receiver nodes, but it may scale well because node-related calculations are minimal. Even so, if the join rate is very high, the root node may exhaust its computing resources. To avoid this situation, the root node should limit the number of acceptances in a unit time (*a moderate join rate*). On the other hand, a node departure triggers the re-announcement initialized by and centralized at the root, and so if the node departure rate is very high, the re-announcement may not be performed in time, and as a result, some paths are temporarily broken and some packets will be lost. Assigning multiple candidates in each forward-to group improves this situation but still may not overcome *catastrophic node losses*, as when most of the receiver hosts leave at the same time.

Error recovery is initiated by the sender side to re-route the sent packets for which no acknowledgments have been returned. Thus, it is decentralized and locally handled in each node, and so it scales very well in comparison with

receiver-based Auto Repeat Request (ARQ), which is a global operation and does not scale well.

C. Throughput

The dynamic path determination is based on the acknowledgments returned from the destination nodes, and so the throughput is dominated not only by bandwidth and network congestion but also by the round trip time (RTT). That is, it depends on network distance in inverse proportion to the RTT. However, since each node including the root transmits a data packet to multiple destination nodes, the actual throughput becomes much higher than for a single fixed destination. For example, the root transmits data packets at the data rate λ to N receiver nodes with no duplicates, and thus the throughput required for each receiver is λ/N, which is much less than the original data rate. This is the same for the relaying nodes. For example, when many nodes belong to each of two segments that are located far apart, data transmission is not performed from a certain fixed node in one segment to another node in the other, but it is performed between two or more nodes in each segment with no data duplicates, and thus it may show higher performance than the conventional fixed path approaches.

D. Other Technical Issues and Possible Solutions

1) IDs and Hierarchy: The recursive peer-to-group routing relies on the node IDs which represent the hierarchical structure of receiver nodes in the network topology. IPv4/IPv6 addresses can be used as the IDs, but they do not necessarily correspond to the whole network topology, especially in the lower bits of addresses, as mentioned in Section III-E. The P2G routing still works in such a case, but the traffic for relaying will increase and may impact the network. The explicit ID structuring by hand which represents the actual network topology works well up to company scales, but not for the Internet scale.

When a subnet is large and consists of several high-performance bridges and potentially a large number of nodes (e.g., 4,096), IP addresses will not work well as the IDs, because they do not always represent the underlying topology. In this case, we need another ID system, or must resort to link-layer multicasts, after data packets arrive at any of the receiver nodes in the huge subnet (the nodes which have received the data packets become senders of the multicast).

When receiver nodes have multiple network interfaces (i.e., multi-homed), we may see ambiguity on IDs for a single node, but this situation can be resolved by specifying the interface explicitly.

2) Scalability and Availability of Root Node: From the view point of data traffic, the root node scales very well for a number of receiver nodes because of the peer-to-group transmission with no duplicate data. Redundant data packets for error correction and acknowledgments returned from the receiver nodes are smaller than the original data, and so the root's workload scales also for handling them. On the other hand, the root is a single special node (1) to accept new receiver nodes, (2) to maintain a list of all receiver nodes, and (3) to trigger re-announcement to compensate for the departed nodes, as discussed in the previous sections. For the first issue, limiting the number of acceptance in a unit time helps the root avoid exhausting its computing resources. The second issue depends on the size of memory resources in a linear way, and so it may still scale up to thousands of receiver nodes. For the third issue of re-announcement, since the remaining receiver nodes are able to announce themselves, it should be not necessary for the root to handle the departure events of receivers except for keeping the list up-to-date, but the self-announcement by receiver nodes might cause another scalability issue in terms of traffic.

As the root node is also a single point to provide data packets, if it fails, then the whole system stops. In order to avoid this single point of failures, we can introduce two or more root nodes, each of which separately and independently maintains the whole list of receiver nodes, where a *source server* provides data packets to the root nodes with no duplicates. The source server is a single meta node but is stateless and simple, and so this configuration would help the whole system be more stable and robust to the failures in the root nodes.

3) Inconsistent Overlay Network: In the recursive peer-to-group routing, data flow is not one-way but uses bi-directional propagation. For example, since firewalls usually inhibit outside nodes from accessing inside ones, no traffic comes from the outside to the inside, which is an inconsistent situation for overlay networks and causes packet losses for the inside nodes. In order to avoid such inconsistent situations, we may need to limit service areas by using explicit restrictions on the hierarchy.

Once a node accidentally takes one or more peer nodes away from its destination candidates, the destination peer nodes may no longer receive data directly from the source node. However, they still have chances to receive data from "near" peer nodes, that is, via other paths. Even in such situations, this may cause inconsistencies in the overlay networks. Re-announcement of the destination nodes recovers to a consistent network state, as long as the source node adds them again after the second announcement. Note that all of the nodes are able to announce themselves to the peer-to-group network, and this is an decentralized operation. This would be an easy but final resort to recover potential paths towards the nodes.

4) Complete Error Recovery: Any lost packets must be recovered for any applications that need the complete set of data packets to reproduce the original data. Any feedback channels such as Auto Repeat Request (ARQ), in which a number of end nodes ask a single root to resend various lost packets, would break the scalability of the system. A combination with erasure codes with high redundancy ($R \gg B - R$), such as Tornado codes [6], will work well with any multicast protocols and systems, while priority coding [9] works well for applications that allow distortions up to some extent.

VI. Conclusions

This paper presented a new application-layer multicast technique, recursive peer-to-group routing, in which data delivery paths are not fixed but change dynamically and frequently even when no failures occur at receiver nodes. Since it assigns no fixed roles to any nodes and makes them work on the basis of equal responsibility, the whole system is robust to membership changes while minimizing errors and avoiding single points of failure. Determining multicast paths is decentralized and distributed to end nodes. It also offers automatic and rapid adaptation of delivery routes to the non-uniformity and temporary fluctuations in host performance.

In order to address scalability issues for multicast, we exploit the network hierarchy and introduce hierarchical groups as layered destinations in each node. The network hierarchy and layered groups never change during a multicast service, while membership may change in each layered group. Each node independently handles the membership changes, and this is scalable. However, a root is responsible for accepting and maintaining new nodes when they join in the service, which may not scale for a large number of receiver nodes.

Analysis of network traffic is discussed in terms of uplink and downlink stresses at each node and at each hierarchical segment in the network. The downlink stress is always the same as the original data rate at any hierarchical levels and the uplink stress is close to but less than the data rate when the hierarchy is symmetric and balanced. This indicates that network traffic is well localized when the hierarchy matches the actual network topology. Another analysis on latency and loss rate was discussed. This was based on concatenated queuing systems with multiple servers (i.e., destination candidates), in which exponential service time is assumed to model network transmission. According to this analysis, we can control the latency and final loss rate based on the offered load, the number of destination candidates, the number of stages in the concatenations, and the recovery rate of the erasure codes. We also presented a blocking system that does not queue but waits until destinations are ready, and our simulation shows that it allows the maximum throughput that the system offers while it also works well for near-realtime applications with predefined data rates, such as video and audio streaming systems.

References

[1] RFC2460: Internet Protocol, version 6 (IPv6) specification, 1998.

[2] http://www.alphaworks.ibm.com/tech/p2g, 2005.

[3] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. ACM SIGCOMM*, Aug. 2002.

[4] S. Banerjee, S. Lee, B. Bhattacharjee, and A. Srinivasan. Resilient multicast using overlays. In *Proc. ACM SIGMETRICS*, June 2003.

[5] J. Blomer, M. Kalfane, R. Karp, M. Karpinski, M. Luby, and D. Zuckerman. An xor-based erasure-resilient coding scheme. Technical report, International Computer Science Institute, Berkeley, California, 1995.

[6] J. W. Byers, M. Luby, M. Mitzenmacher, and A. Rege. A digital fountain approach to reliable distribution of bulk data. In *Proc. ACM SIGCOMM*, Sept. 1998.

[7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth content distribution in a cooperative environment. In *Proc. IPTPS*, Feb. 2003.

[8] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlay networks. In *Proc. IEEE INFOCOM*, Apr. 2003.

[9] P. Chou, V. Padmanabhan, and H. Wang. Resilient peer-to-peer streaming. Technical Report UCB/CSD-01-1141, Microsoft Research, Redmond, WA, Mar. 2003.

[10] Y.-H. Chu, S. G. Rao, and H. Zhang. A case for end system multicast. In *ACM SIGMETRICS*, June 2000.

[11] S. Deering. Multicast routing in internetworks and extended lans. In *Proc. ACM SIGCOMM*, Aug. 1988.

[12] P. Francis. Yoid: extending the multicast internet architecture. preprint available from http://www.yallcast.com, Sept. 1999.

[13] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. OSDI*, Oct. 2000.

[14] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. ACM SOSP*, Oct. 2003.

[15] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proc. USITS*, Mar. 2001.

[16] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proc. ACM SIGCOMM*, Aug. 2001.

[17] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2218, 2001.

[18] A. I. T. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The design of a large-scale event notification infrastructure. In *Networked Group Communication*, 2001.

[19] M. Yang and Z. Fei. A proactive approach to reconstructing overlay multicast trees. In *IEEE INFOCOM*, 2004.

[20] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, UC Berkeley, Apr. 2001.

[21] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide area data dissemination. In *Proc. NOSSDAV*, June 2001.

Appendix

A. Derivation of Waiting Time Distribution

The PDF of the waiting time distribution of a single $G/M/m$ system is

$$f_W(t) = (1-a)\delta(t) + abe^{-bt}, \tag{9}$$

and its Laplace transformation is

$$f^*_W(s) = (1-a) + \frac{ab}{s+b}, \tag{10}$$

where

$$a = \frac{a_m}{1-\beta}, \tag{11}$$

$$b = m\mu(1-\beta) \tag{12}$$

If the number of steps i ranges from zero to k and it is weighted by $\binom{k}{i}/2^k$, then the Laplace transform of the k-step $G/M/m$ system is:

$$f^*_{W_k}(s) = \left\{(1-a) + \frac{ab}{s+b}\right\}^k, \tag{13}$$

$$= \sum_{i=0}^{k} \binom{k}{i}(1-a)^{k-i}a^i\left(\frac{b}{a+b}\right)^i, \tag{14}$$

and thus the waiting time of the k-step $G/M/m$ system is

$$F_{W_k}(t) = 1 - \sum_{i=1}^{k} \binom{k}{i} (1-a)^{k-i} a^i Q(i-1; bt) \qquad (15)$$

Finally, the total waiting time distribution of the hyper k-step $G/M/m$ system is

$$f_W^*(s) = \sum_{k=0}^{n} \frac{1}{2^n} \binom{n}{k} \left\{ (1-a) + \frac{ab}{s+b} \right\}^k, \qquad (16)$$

$$F_W(t) = 1 - \frac{1}{2^n} \sum_{k=1}^{n} \binom{n}{k} F_{W_k}^c(t) \qquad (17)$$

B. Derivation of Service Time Distribution

If the number of steps i ranges from one to $(n+1)$ and is weighed by $\binom{n}{i}/2^n$, then the Laplace transform of the hyper k-step Erlangian process is

$$f_S^*(s) = \frac{1}{2^n} \sum_{i=0}^{n} \binom{n}{i} \left(\frac{\mu}{s+\mu} \right)^{i+1}, \qquad (18)$$

and the total system time is

$$F_S(t) = 1 - \frac{1}{2^n} \sum_{i=0}^{n} \binom{n}{i} Q(i; \mu t). \qquad (19)$$

The mean system time is then calculated as follows.

$$\overline{S} = \int_0^\infty F_S^c(t) dt, \qquad (20)$$

$$= \frac{1}{2^n} \sum_{k=0}^{n} \binom{n}{k} \frac{k+1}{\mu}, \qquad (21)$$

$$= \frac{1}{\mu} + \frac{n}{2\mu} \sum_{k=1}^{n} \binom{n-1}{k-1} \Big/ 2^{n-1}, \qquad (22)$$

$$= \frac{n+2}{2\mu}. \qquad (23)$$

C. Mean Length of Stages in Recursive P2G Routing

The mean length of stages from a root in the n-stage recursive peer-to-group routing is

$$a_n = 1/2^n + \sum_{k=1}^{n} (k+1)/2^{n-k+1}, \qquad (24)$$

which is converted into

$$a_0 = 1, \qquad (25)$$

$$a_n - a_{n-1} = 1 - \frac{1}{2^n}, \qquad (26)$$

and thus the mean is simplified to

$$a_n = n + \frac{1}{2^n}, \qquad (27)$$

which approaches n as n increases.