SoMR: A Scalable Distributed QoS Multicast Routing Protocol *

Shigang Chen (correspondence author)

Department of Computer & Information Science & Engineering

University of Florida, Gainesville, Florida 32611, USA

Email: sgchen@cise.ufl.edu

Phone: 352 392 2713

Fax: 352 392 1220

Yuval Shavitt

Department of Electrical Engineering - Systems

Tel-Aviv University, Tel-Aviv 69978, Israel

March 17, 2007

*The preliminary version of the paper is published in [1].

Abstract

Many Internet multicast applications such as teleconferencing and remote diagnosis have Quality-of-Service (QoS) requirements. The requirements can be additive (end-to-end delay), multiplicative (loss rate), or of a bottleneck nature (bandwidth). Given such diverse requirements, it is a challenging task to build QoS-constrained multicast trees in a large network where no global network state is available. This paper proposes a scalable QoS multicast routing protocol (SoMR) that supports all three QoS requirement types. SoMR is scalable due to small communication overhead. It achieves favorable tradeoff between routing performance and routing overhead by carefully selecting the network sub-graph in which it searches for a path that can support the QoS requirements. The scope of search is automatically tuned based on the current network conditions. An early-warning mechanism helps detect and route around the long-delay paths in the network. The operations of SoMR are completely decentralized. They rely only on the local state stored at each router.

keywords: multicast routing; quality of service; delay constraint

1 Introduction

Multicast is an efficient way of delivering content to a large group of receivers by using a tree structure embedded in the network. Building a multicast tree, called *multicast routing*, has been studied extensively. Finding a minimum-cost multicast tree, e.g., one that uses a minimal number of links, is known to be NP-hard [2]. Heuristics are deployed to find trees that are practically "good enough". The simplest heuristic is to form a tree based on unicast routing paths between the sender and the receivers [3, 4]. This approach was adopted by IETF standards: DVMRP [5], PIM [6], MOSPF [7]. The unicast-path trees work well for delivering information that is not QoS sensitive. However, when applications have non-trivial QoS constraints, the unicast-path trees may not have adequate characteristics to meet the constraints [8].

QoS-constrained multicast routing has many important applications such as teleconferencing, IPTV, and on-demand vedio distribution. Today, these applications are mostly implemented through unicast connections, which have lower quality and higher cost. Protocols for QoS-constrained multicast routing, if implemented, will improve the quality and reduce the cost of these applications. However, the legacy design of the Internet is not compatible with QoS provision. Designed thirty years ago, the current Internet has fundamental problems in QoS support, security and manageability. The networking community has been building consensus that we should take a "clean-slate" approach to design the next-generation Internet, which is embodied in the FIND initiative from NSF. The research on QoS multicast routing has a potential to enable various multimedia applications on the future Internet.

A *feasible tree* is a multicast tree that satisfies the QoS requirements. A *feasible tree branch (path)* is a path that connects a new group member to a multicast tree without breaking any QoS constraint. The task of QoS multicast routing is to find feasible tree branches for new group members. A survey of this research area can be found in [9]. Finding feasible tree branches is difficult in a large network such as the Internet. It is impossible to maintain global QoS state at any single node. A brute-force flooding algorithm that searches all possible paths in the network guarantees to find a feasible branch if one exists. However, the excessive overhead of flooding deems to be impractical for all but small networks. Consequently, for applications that require QoS guarantees, recent research focuses on distributed multicast

routing algorithms that search a selected subset of the network to find feasible tree branches for new group members [8, 10, 11, 12, 13, 14, 15].

A good QoS routing protocol should achieve favorable tradeoff between routing overhead and routing performance (the ability of finding a feasible branch when one exists). In addition, it should minimize the extra state information kept at the routers, decentralize the routing operations, adapt the routing activity based on the current network conditions, and avoid the congested network areas. QMRP [8] has many of the good merits mentioned above. However, it suffers from two problems. First, it deposits per-join state at routers. When there are many concurrent joins for a multicast group, a router has to keep separate state information for each join that it assists. It is highly desirable for the routers to not keep any per-join information but only maintain per-group information. Second, QMRP is designed for applications with bottleneck QoS requirements such as bandwidth and buffer space. It lacks effective mechanisms to handle additive/multiplicative QoS requirements such as delay or packet loss. QoS routing for additive/multiplicative requirements is a more difficult problem. How to extend QMRP for all kinds of requirements is a challenge by itself.

Spanning join [10] and QoSMIC [12] do not have the above problems. However, they have much higher overhead and lower success probability [8]. The spanning-join protocol relies on expanding-ring flooding to find an on-tree node for a new member to join. QoSMIC combines a local search and a tree search to avoid large-scale flooding, but its overhead can still be high for large multicast trees.

One may argue that any distance-vector protocol will work well for constructing multicast trees with an additive delay constraint. However, if there is an additional bandwidth constraint and different multicast groups have different bandwidth requirements, then the protocol has no way to determine which links should be excluded from the shortest-path computation, because a link that has sufficient bandwidth for one group may not be qualified for another group. Note that we cannot run the distance-vector protocol separately for each multicast group, which does not scale in a large network with many groups. OSPF-like protocols are not a viable option either because maintaining global QoS state at each node is too costly for a large network. Hence, further study for a scalable, efficient QoS multicast routing protocol that can handle all types of QoS requirements is under call.

In this paper, we propose a scalable QoS multicast routing protocol, called SoMR, that eliminates the

use of per-join routing state. In QMRP, each new member initiates a search tree, which grows towards the multicast tree. The search tree is per-join state information. A router involved in multiple search trees has to keep state information for each tree. SoMR does not use search trees. Instead it grows the multicast tree towards new members. The maximum memory space a router uses for a multicast group is fixed, which is one multicast routing entry. SoMR not only gets rid of per-join state but also allows dynamic aggregation of multiple join requests, where a single tree branch may grow toward multiple new members.

To improve the chance of finding a feasible path, SoMR may branch out to grow multiple branches in the multicast tree towards the new member. The key problem is to select the optimal locations at which branching should occur. Such locations are called *branching points*. For delay or other additive/multiplicative requirements, the optimal branching points are not at the locations where the requirements are violated. It is often because an early link has too large delay. The branching should occur there. SoMR uses a novel early-warning (EW) mechanism which finds the real problematic locations in the network and makes branches to detour around those locations. By doing this, SoMR increases the success ratio in finding feasible branches for new members.

We also propose several optimization techniques to control the worst-case overhead. We prove the correctness of SoMR and analyze its performance. We perform extensive simulations to demonstrate that SoMR is able to achieve high success ratio at low overhead.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 defines our network model. Section 4 describes the routing protocol. Section 5 and Section 6 present analytical and simulation results, respectively. Section 7 draws the conclusion.

2 Related Work

A multicast tree is incrementally constructed as members leave and join a multicast group. When an existing member leaves the group, it sends a control message up the tree to prune the branch which has no members attached any more. When a new member joins the group, the tree must be extended to cover the new member. Based on how the new member is connected onto the tree, the multicast routing

protocols can be classified into two broad categories: *single-path routing protocols* (SPR) and *multiple-path routing protocols* (MPR). An SPR protocol provides a single path connecting the new member to the tree, whereas an MPR algorithm provides multiple candidate paths to choose from.

2.1 Single-path routing

Most SPR protocols were originally designed for the best-effort data traffic. We briefly discuss two representative protocols and point out why they are not suitable for QoS traffic. CBT (Core-Based Tree) [3, 16] and PIM (Protocol Independent Multicast) [4] connect a new member i to the multicast tree along the unicast routing path from i to the root (core) of the tree. The unicast path is typically the shortest path in term of hops. The resulting shortest-path trees are good for best-effort traffic. However, when QoS is considered, such shortest-path trees may not have the resources to support the quality requirement.

2.2 Multiple-path routing

In order to increase the chance of finding a feasible tree, the MPR protocols provide multiple candidate paths for a new member to be connected to the tree. Among the candidates the new member selects the best one.

Spanning-joins [10]: In the spanning-joins protocol proposed by Carlberg and Crowcroft, a new member broadcasts join-request messages in its neighborhood to find on-tree nodes. Whenever an on-tree node receives the message, it sends a reply message back to the new member. The path of the reply message, determined by the unicast routing algorithm, is a candidate path. The new member may receive multiple reply messages corresponding to multiple candidate paths. Each reply message collects the QoS properties of the path it traverses. The new member selects the best candidate path based on the information received in the reply messages. Consecutive broadcasts are necessary to search increasingly larger neighborhood until on-tree nodes are found, and this process can increase the overhead significantly.

QoSMIC [12]: In the QoSMIC protocol proposed by Faloutsos et al, the search for candidate paths consists of two parallel procedures: *local search* and *tree search*. The local search is equivalent to the spanning-joins protocol, except that only a small neighborhood is searched. The tree search handles the

case when there is no on-tree node in the neighborhood checked by the local search. In the tree search, a new member sends an M-JOIN message to a designated Manager node for the group. Upon receipt of the message, the Manager multicasts a BID-ORDER message in the tree to select a subset of on-tree nodes. The selected nodes send BID messages to the new member. The paths of the BID messages, determined by the underlying unicast routing protocols, are candidate paths. The tree search allows QoSMIC to restrict its *flooding* local search in a small neighborhood.

Both spanning-joins protocol and QoSMIC are not QoS-aware in selecting candidate paths. The selection of on-tree nodes, to which the new member may join, is based on the connectivity.[1] The candidate paths are simply the unicast routing paths from the selected on-tree nodes to the new member. These paths are typically the shortest paths in terms of number of hops, and may not be the best choice for the QoS requirements specified in other terms such as delay or bandwidth. Hence, the information about the specific QoS requirement and the availability of relevant resources should be used to make more effective selection of candidate paths.

QMRP [8]: QMRP grows a search tree from each new member to the root of the multicast tree. The branches growing in the search tree must satisfy a bandwidth requirement. When a branch in the search tree touches a branch in the multicast tree, a feasible path that connects the new member to the multicast tree is discovered. QMRP only considers bandwidth constraints. Its search tree is per-group-per-join state.

2.3 Other Related Work

There are other related works that study the QoS multicasting problem from different aspects. Rong et al. integrate active admission control into traditional QoS multicast routing algorithms to prevent bandwidth fragmentation in multicast networks [17]. Striegel and Manimaran present a multicast "life-cycle" model, identifying various issues that are involved in a typical multicast session [18]. Pradhan et al. propose a hierarchical multicast routing protocol that achieves scalability by organizing a network as a hierarchy of domains using the full-mesh aggregation technique [14]. Li investigates the constrained multicast routing

[1] After candidate paths are selected, the protocol becomes QoS-aware because the QoS properties of the candidate paths are collected and checked to see if any of them meet the requirement.

problem in networks with imprecise state information [19]. Lui et al. propose a receiver-initiated QoS multicast protocol that aims at reducing the bandwidth used in building a multicast tree for heterogeneous receivers [15]. Charikar et al. study several NP-hard resource optimization problems in multicast routing and propose heuristic solutions with provable performance bounds [20]. Sai Sudhir et al. address the problems of static and dynamic heterogeneous QoS multicasting in DiffServ networks [21].Rong et al. integrate active admission control into traditional QoS multicast routing algorithms to prevent bandwidth fragmentation in multicast networks [17]. Striegel and Manimaran present a multicast "life-cycle" model, identifying various issues that are involved in a typical multicast session [18]. Pradhan et al. propose a hierarchical multicast routing protocol that achieves scalability by organizing a network as a hierarchy of domains using the full-mesh aggregation technique [14]. Li investigates the constrained multicast routing problem in networks with imprecise state information [19]. Lui et al. propose a receiver-initiated QoS multicast protocol that aims at reducing the bandwidth used in building a multicast tree for heterogeneous receivers [15]. Charikar et al. study several NP-hard resource optimization problems in multicast routing and propose heuristic solutions with provable performance bounds [20]. Sai Sudhir et al. address the problems of static and dynamic heterogeneous QoS multicasting in DiffServ networks [21].

3 Network model

We make the following assumptions about the network.

1. There exists an underlying unicast routing protocol which can deliver a message between any two connected nodes in the network. A node knows the length (number of hops) of the unicast routing path to any destination. Many widely used unicast routing protocols such as RIP and OSPF provide this information.

2. Every node maintains its up-to-date local state, such as the delay of each outgoing link, which includes the processing time, buffering delay, and link propagation delay. Assume that once resources are committed, such delay can be assured during the lifetime of data communication. How to make resource reservation [22, 23] and what packet scheduling algorithms should be used [24, 25] are beyond the scope of this paper.

A node is not required to maintain any global state information or the local state of any other node. This distinguishes our work from many other works that need to acquire the global state [26, 27] or a partial map [28] of the network.

With the above assumptions, we study how to construct QoS-constrained multicast trees. We use the *delay* constraint as example to illustrate the protocol, while other additive metrics such as *cost* can be supported similarly. Multiplicative metrics such as *loss rate* can also be supported by using logarithm functions to convert them into additive metrics. Bottleneck metrics such as *bandwidth* are easier to handle: the links having the resources have zero weights; the links not having the resources have infinite weights; feasible paths are those whose weights are zero.

There are two types of multicast trees: *sender-based trees* and *core-based trees*. For a delay-constrained sender-based tree, the delay from the sender to any other node in the tree has to be bounded by a delay requirement D. A delay-constrained core-based tree can be modeled as two subtrees. One is from the core to all receivers, and the other is from all senders to the core. Given a delay requirement D, if both subtrees are bounded by $D/2$, then the delay requirement is met. Constructing the first subtree is the same problem as constructing a sender-based tree, while constructing the second subtree is a reverse problem. To simplify the presentation, we will use sender-based trees to illustrate the protocol in the rest of the paper.

We assume that any new member is able to map a multicast group address to the root node of the tree on demand possibly by a query/response Session Directory [29].

Each on-tree node has a multicast routing entry, specifying which node is its parent and which nodes are its children. We define notations in the following. Let k and i be two on-tree nodes. The path in the multicast tree connecting them is called the *in-tree path*, denoted by $P_{k,i}$. The guaranteed delay bound of this path is called the *in-tree delay*, denoted by $delay(P_{k,i})$. Let T be the set of on-tree nodes and r be the root of the tree. A delay-constrained multicast tree requires that

$$\forall i \in T,\ delay(P_{r,i}) \leq D$$

We require each on-tree node i to know $delay(P_{r,i})$. In fact, as we will see later in the protocol description, our protocol makes sure that any node joining the tree will have this value.

We assume that each link (i, j) can ensure a certain delay bound for the Class of Service (CoS) which the multicast group is associated with. We denote this delay bound as $delay(i, j)$.

4 A New QoS Multicast Routing Protocol

In this section, we discuss our design objectives, describe the new routing protocol, and present the pseudo code of the protocol.

4.1 Design Objectives

We design our QoS multicast routing protocol based on the following objectives.

Favorable Tradeoff: Like many other network functions, QoS routing has multiple performance metrics, including *success ratio* and *overhead*, which represent the average probability of finding a feasible path and the average consumption of network resources (e.g., bandwidth, memory, and CPU), respectively. These performance metrics often conflict with each other — it normally takes more overhead to achieve better success ratio. There are two extremes in designing a QoS routing protocol: (1) searching only one path to minimize the overhead while sacrificing the success ratio, and (2) searching all paths to optimize the success ratio at the cost of heavy overhead. Many existing protocols [10, 12, 8] chose a tradeoff between the two extremes: searching multiple but not all paths. The rational behind these protocols is that sub-optimal success ratio can be achieved by searching a set of carefully selected paths. The key is how to make the path selection. A good path-selection strategy achieves *favorable tradeoff* — improving success probability significantly at an insignificant overhead increase.

Adaptivity: The design of many QoS routing protocols is geared towards *congestion* conditions where the total demand of all QoS traffic exceeds the resource supply. These protocols employ extra operations (and overhead) in order to find a good way to lay out the routing paths so that more connections can be accommodated. The problem is that, for spanning-join and QoSMIC, the same operations are performed for any network conditions. The extra overhead for congestion condition becomes a waste under normal traffic condition, where networks are designed to operate for most of the time.[2]

[2]Even when QoS traffic is light, best-effort traffic may be heavy [30]. In order to improve the performance of best-effort

Therefore, an adaptive routing algorithm is desired. It should be able to detect the network congestion condition and introduce extra overhead only when necessary. In addition, the amount of extra overhead should also be adaptive according to the traffic condition in the area where the routing takes place.

4.2 Protocol Overview

SoMR consists of two phases. Let r be the root node of the tree. The first phase is similar to shortest path routing (SPR), in which a JOIN message is sent from a new member t to the root r along the unicast routing path. The JOIN message accumulates the path it traverses. It also accumulates the delay of the path in the reverse direction. When the JOIN message reaches an on-tree node k, if the accumulated delay plus the in-tree delay from r to k does not violate the delay requirement, a feasible tree branch is detected, which is the traversed unicast path.[3] A CONSTRUCTION message is then sent back along the path (using *IP source routing*) to construct a tree branch connecting the new member. Since the new member joins the tree successfully, the second phase will not be activated.

On the other hand, if the delay requirement is violated at k, the JOIN message continues traveling to the root r. When the root receives the message, it starts the second phase, which employs multi-path routing. The root sends GROW messages to its neighbors. These GROW messages will then travel along the unicast routing paths towards the new member. As they travel, they try to construct new tree branches hop by hop along the way. It should be emphasized that, even though multiple temporary tree branches may grow to a new member, all but one will withdraw. Moreover, any router in the network will keep *at most one multicast routing entry* no matter how many concurrent joins there are. *No per-join information is kept.* In Section 4.3 and Section 4.4, we will show that, when two growing branches (either for the same join or for different joins) meet at a common node, one branch will withdraw and the other branch will continue growing towards one or multiple new downstream members.

Each GROW message carries the delay requirement D. It also accumulates the delay of the con-

traffic, the excess overhead by the QoS routing protocol is always undesirable.

[3] In addition to delay, if there are additional additive constraints (such as cost), they will be handled in the same way. A feasible branch is detected when none of the additive constraints is violated. If there are additional bottleneck constraints (such as bandwidth), then we can simply set the delay of any link that violates a bottleneck constraint to be infinite, and the rest of the protocol remains the same.

structed tree branch. Hence, when an intermediate node i receives GROW, it knows the in-tree delay from r to i, $delay(P_{r,i})$. Now, let us describe the actions that i will take after receiving GROW. First, i performs an EW (Early Warning) test to see how likely the unicast path to the new member t will satisfy the delay requirement D. If the EW test is passed, GROW will continue traveling along the unicast path towards t; otherwise, i attempts multi-path routing which may result in multiple downstream tree branches to be constructed. Let j be the next hop on the unicast path. The EW test takes four input parameters, D, $delay(P_{r,i})$, $delay(i,j)$, and l, which is the length of the unicast path from i to t. The test is defined as follows.

if $delay(i,j) > (D - delay(P_{r,i}))/l$ **then** warning **else** pass

$D - delay(P_{r,i})$ is the remaining slack of the delay requirement that further tree construction is allowed to have. $(D - delay(P_{r,i}))/l$ is the "fair share" of this slack for each link on the path from i to t. The above EW test states that if the delay of the link is larger than the fair share, a warning should be triggered; otherwise, the test is passed. More sophisticated EW tests are possible, but are not considered in this paper since the above simple test already works well in our simulations.

If the EW test is passed, which means that the current unicast path is likely to satisfy the QoS requirement, i adds link (i,j) into the multicast tree and forwards the GROW message to the next hop j. If every intermediate node passes the EW test, a feasible branch is established for the new member.

However, if the EW test warns that the unicast path may violate the QoS requirement, extra effort needs to be taken. Searching multiple downstream paths will increase the chance of success. Namely, the tree construction needs to *branch out*. We call i a *branching point*. GROW messages are sent to a subset of adjacent nodes x that satisfy the following *QoS test*:

if $delay(i,x) > D - delay(P_{r,i})$ **then** fail **else** pass

Apparently, x can be the node j that just failed the EW test, but x should not be the adjacent node from which the GROW was previously sent to i. For the purpose of overhead reduction, we may select only some of the nodes that pass the QoS test (see section 4.6).

If the QoS test is failed for every adjacent node, a BREAK message is sent back to trim the partially constructed tree branch. When a node k receives a BREAK message from a node i, it first deletes link

(k, i) from the multicast tree, and then if k becomes a leaf node and is not a member of the multicast group, it will delete itself from the multicast tree and propagate the BREAK message to its parent node. As BREAK travels back to r, the new tree branch is deleted.

There is a difference between the EW test and the QoS test. The EW test tries to make early guess on whether the path ahead is likely to satisfy the delay constraint. If it sees signs of trouble, it triggers branching to improve the chance of success. The QoS test is to check if the delay constraint has already been violated. If it is, no further construction will be done towards this direction.

At the beginning of the second routing phase, our protocol requires the root to be a mandatory branching point (an EW test is not necessary). Our simulations consistently show that SoMR performs better this way. The reason is that an early branching widens the search range and gives the subsequent tree construction more flexibility. It should be noted that the second phase commences only if the first phase fails, i.e., the SPR path fails.

Whenever a GROW message reaches t, a feasible tree branch is found. However, t may receive multiple GROW messages from different branches. Fig. 1 gives an example. The multicast tree is shown by bold lines. i is a branching point, from which two branches reach t. In this case, t needs to send back BREAK messages to tear down all but the best branch. t can use the information (delay, bottleneck bandwidth, etc.) collected by the received GROW messages to select the best branch, and it will send a resource reservation message back up to reserve resources along this branch. It is rare but possible that, due to concurrent joins in other multicast groups, the selected tree branch becomes infeasible during the short period after GROW passes through the branch and before the resource reservation message comes back. When this happens, the new member will have to perform the join again.

4.3 Breaking Loops

As tree branches are constructed towards new member(s), loops may form in the multicast tree. Fig. 1 gives one example. A loop forms when two growing tree branches reach the same node. This is not a classical loop in which a message will travel forever. It is a loop formed by two tree branches joining at a common node. The node would receive two copies of the same message if the loop was not broken.

Before we provide a general solution to the looping problem, we need to study GROW messages

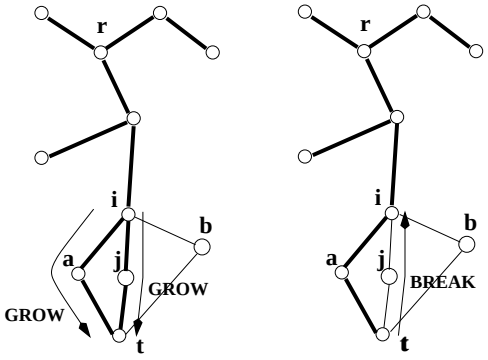

Figure 1: GROW and BREAK

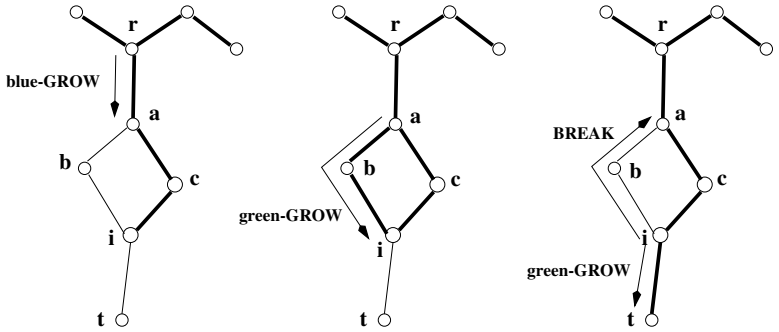

Figure 2: Break loops

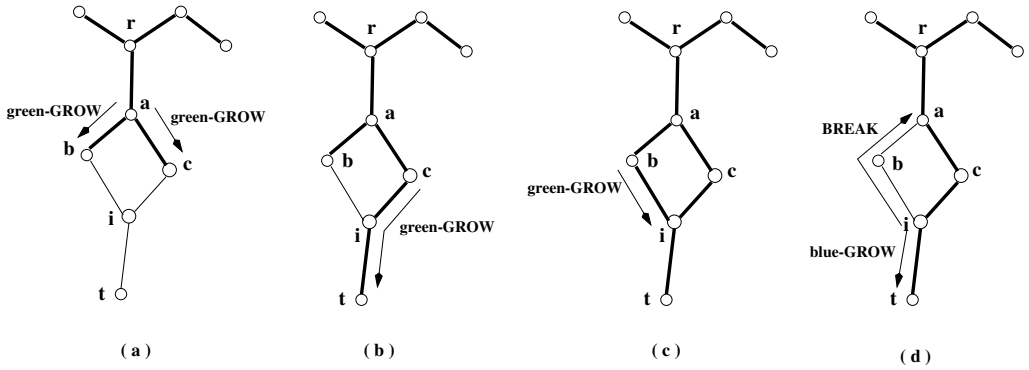

Figure 3: Two growing tree branches meet

more closely. Consider a GROW message that constructs a tree branch along a unicast path P to the new member. Some of the links on P may be already in the multicast tree while the others are not. When a GROW message travels along a link that is already in the multicast tree, we assign a color of *blue* to the GROW message. When a GROW message travels along a link that is not in the multicast tree, we assign *green* to the message.[4] Only green-GROW messages may form loops, because green-GROW messages join new links to the tree while blue-GROW messages follow existing tree links.

The sender of a GROW message can determine the color of the message as follows. When a node i sends a GROW to an adjacent node j, if j is the parent or a child of i in the multicast tree, i marks the GROW to be blue; otherwise, it marks the GROW to be green.

Using the coloring scheme, loop detection is easy. When an on-tree node receives a green-GROW message, a loop is formed. Fig. 2 gives an example, where the existing tree is shown in the left plot. The GROW message is blue when it traverses an on-tree link $r \rightarrow a$, and then it turns green when traversing links outside of the tree, $a \rightarrow b \rightarrow i$, which are consequently included in the tree. When the on-tree node i receives a green-GROW message, a loop is detected. A BREAK message is sent back to break the loop, while the GROW message continue constructing a tree branch towards the new member.

Arriving at i, the GROW message has the in-tree delay of the new tree branch ($r \rightarrow a \rightarrow b \rightarrow i$). i knows the in-tree delay of the old tree branch ($r \rightarrow a \rightarrow c \rightarrow i$). The BREAK message can be sent to tear down the new branch, in which case the in-tree delay in the GROW message needs to be updated to equal that of the old tree branch. Or the BREAK message can be sent to break the old branch based on certain optimization criteria (e.g., the in-tree delay of the new branch is smaller), and in this case the in-tree delay stored at i needs to be updated.

Two growing branches may meet at a common node to form a loop, which will be broken similarly. In Fig. 3 (a), two green-GROW messages grow two tree branches simultaneously. Suppose the green-GROW message from c arrives at i first in Fig. 3 (b). The message adds i to the tree and continues to travel towards the destination along unicast path. Then in Fig. 3 (c) the second green-GROW message arrives at i. This message triggers a BREAK message to break the loop and a blue-GROW message follows the same unicast path to the destination in Fig. 3 (d). The blue-GROW message is necessary

[4]The green-GROW message will join this link to the multicast tree.

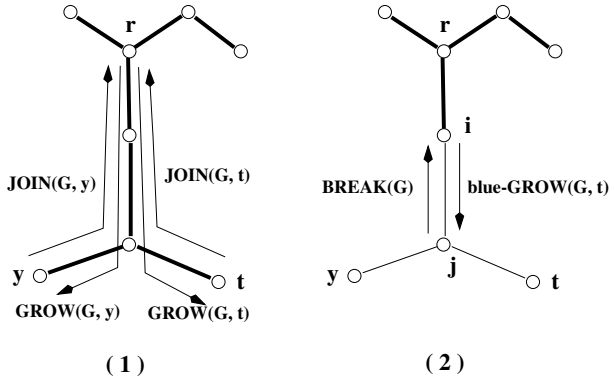

Figure 4: Concurrent joins

because i does not keep per-join routing state to indicate that a previous green-GROW for the same destination has been sent downstream.

No blue-GROW messages but only green-GROW messages will cause BREAK messages.

4.4 Concurrent Joins

SoMR can efficiently support concurrent join requests. Fig. 4 (1) gives an example, in which each GROW message is identified by the group address G and the new member address. Two concurrent new members, t and y, join the tree at the same time. Two GROW messages independently construct two new branches, which are partially overlapped. The messages contain different new member addresses, and thus the branches grow towards different directions.

Fig. 4 (2) gives an abnormal case. Suppose the green-GROW message for y arrives at j first. It finds that link (j, y) will violate the delay requirement. It sends a BREAK message back to i to tear down the partially constructed tree branch. While the BREAK message is on the way to i, i sends a blue-GROW message for t down the same link. When j receives the blue-GROW, link (i, j) no longer belongs to the multicast tree and thus j should not construct a tree branch further towards t. Therefore, when a node (j) receives a blue-GROW message and finds that it is no longer an on-tree node, it should discard the blue-GROW message without further constructing tree branch.

4.5 Pseudo Code

The first phase of SoMR is quite straightforward thus we present only the pseudo code of the second phase. Two types of routing messages are of concern.

1. A GROW message grows a new tree branch to connect the new member.

2. A BREAK message tears down a tree branch in order to break a loop in the multicast tree.

Both messages carry the multicast group address. GROW messages also carry the new member address, the delay bound D, the in-tree delay from the root to the current node, and the message color.

Each node in the multicast tree keeps a *multicast routing entry*, which is denoted as $M\{G, in, out\}$, where $M.G$ is the address of the multicast group, $M.in$ is the parent node in the multicast tree, and $M.out$ is the set of child nodes. The next hop on the unicast routing path from node i to node t is denoted as $N_{i\to t}$. Given any node i, our protocol is implemented by the following pseudo code. Suppose i received a control message from k.

Node i:

```
switch (the received message)
case GROW(G, t, ...):
    if (it is a green-GROW)
        if (i is an on-tree node)
            /* break loop in the tree */
            send BREAK(G) back to k
        else
            /* add i into the tree */
            create a multicast entry M(G, out, in)
            M.in = k
            M.out = ∅
    if (i = t)
        routing is successful
    else
        /* further grow tree branch to next hop */
        if (N_{i→t} ∈ M.out or N_{i→t} = M.in)
            send blue-GROW(G, t, ...) to N_{i→t}
        else if (the EW test is passed)
```

$M.out := M.out + \{N_{i \to t}\}$
send green-GROW(G, t, ...) to $N_{i \to t}$
else
for every adjacent node j, $j \neq k$
if ($j \in M.out$ **or** $j = M.in$)
send blue-GROW(G, t, ...) to j
else if (j passes the QoS test)
$M.out := M.out + \{j\}$
send green-GROW(G, t, ...) to j
if (no GROW was sent by the **for** loop, $M.out = \emptyset$, and i is not a group member)
/* tear down partially constructed branch */
remove the multicast entry $M(G, out, in)$
send BREAK(G) to k
case BREAK(G):
$M.out := M.out - \{k\}$
if ($M.out = \emptyset$ and i is not a group member)
/* remove i from the multicast tree */
remove the multicast entry $M(G, out, in)$
send BREAK(G) to $M.in$

4.6 Optimization

Whenever the EW test generates a warning at an intermediate node, the node becomes a branching point and multiple tree branches may grow out from this node towards the new member.[5] The number of branching points, if not restricted, can potentially be large, which will result in large routing overhead. We define two protocol parameters that are used to restrict the number of constructed tree branches.

Maximum Branching Level (MBL): An easy way to control the number of branching points is to maintain an assertion: when a GROW message travels from the root and a new member, it may pass at

[5]The BREAK messages will cut all but one branch. Hence, there will be only one tree branch connecting the new member eventually. The key difference from QMRP is not about less number of temporary tree branches per join but about removing temporary per-join state information from routers. SoMR does not require any information other than one multicast entry to be stored at a router for a group.

most m branching points, where m is a system parameter called *maximum branching level*. Since the root is a mandatory branching point, each GROW message from the root carries a counter whose initial value is $m-1$, specifying the maximum number of downstream branching points that it can pass. When a GROW message reaches a node that fails the early warning test, if the counter in the message is above zero, the node becomes a branching point, which forwards GROW to multiple neighbor nodes with the counter value decreased by one; however, if the counter is already zero, the node will not become a branching point and the GROW is forwarded only on the unicast path to the new member.

It is easy to see that the maximum number of branching points is bounded by $\sum_{i=0}^{m-1}(d-1)^i$, where d is the maximum degree of a node. When $d=2$, $\sum_{i=0}^{m-1}(d-1)^i = m$; when $d>2$, $\sum_{i=0}^{m-1}(d-1)^i = \frac{(d-1)^m-1}{d-2}$. Such a restricted version of SoMR is denoted as SoMR-m. An illustration of SoMR-3 is given in Fig. 5. SoMR-m can be easily implemented by augmenting the GROW messages with a counter.

Directivity can be implemented to discourage tree branches growing away from the new member t. When a GROW is sent from i to j, if the distance from j to t is not shorter than the distance from i to t, the counter for MBL is set to zero, indicating that there is no branching point allowed for this GROW message.

Maximum Branching Degree (MBD): A branching point may have a large number of adjacent links, which can also cause excessive overhead. SoMR-m can be further augmented with an additional parameter, *maximum branching degree*, which specifies the maximum number of GROW messages that are allowed to be sent by a branching point. If the maximum branching degree x is smaller than the node degree minus one,[6] the node selects x outgoing links (based on distance to the new member or randomly) from which GROW has not been received, and sends GROW messages out along these links.

We suggest both MBL and MBD to be implemented. With MBL = m and MBD = x, the maximum number of branching points is $\Sigma_{i=0}^{m-1}x^i = \frac{x^m-1}{x-1}$. Therefore, the overhead can be controlled by these two parameters.

[6]The node should not send GROW to a link from which a GROW message has been received previously.

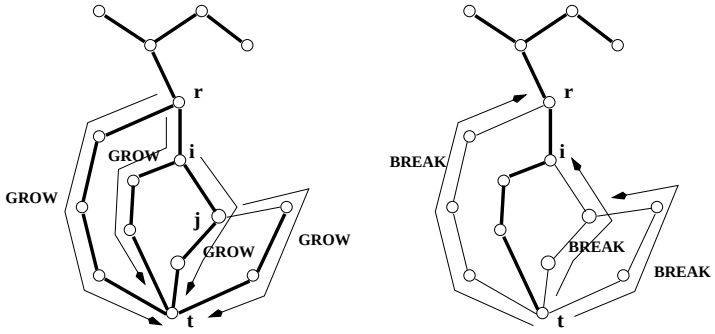

Figure 5: an example of SoMR-3

5 Analysis

5.1 Correctness Analysis

We show that SoMR does not form any persistent loop, does not partition the multicast tree, and terminates in finite time.

Theorem 1 *SoMR never forms a persistent loop in the multicast tree.*

Proof: Only green-GROW messages add new links to the tree and can form loops. Suppose j sends a green-GROW message to i, adds link (j, i), and forms a loop. Let k be the parent node of i in the tree. By the structure of trees, both link (j, i) and link (k, i) belong to the loop. According to Section 4.3, a BREAK message will be sent to either j or k to tear down (j, i) or (k, i), which breaks the loop immediately. Hence, any loop can not persist. □

SoMR grows new tree branches from the tree towards new members. In many other protocols, new branches are constructed from new members towards the tree. PIM is an example. Before new branches actually reach the tree, the tree can be viewed as being partitioned. This is fine without QoS constraints because those growing branches will eventually reach the tree by following the unicast paths. However, with QoS constraints, it can cause a problem. When a growing branch reaches an on-tree node, the node may belong to another growing branch, which may be torn down later due to the violation of QoS constraints. This case never arises in SoMR.

Theorem 2 *SoMR never causes the multicast tree to be partitioned into more than one disconnected pieces.*

Proof: New tree branches always grow from the multicast tree.[7] Hence, any new on-tree nodes must be connected with the tree. BREAK messages only remove leaf nodes from the tree and thus have no chance to partition the tree. □

The above two theorems can also be applied to SoMR-m for any m.

The routing process terminates when no control message of a join request is still in transit.

Theorem 3 *SoMR-m terminates in finite time.*

Proof: Consider an arbitrary join request. SoMR-m has at most $\sum_{i=0}^{m-1}(d-1)^i$ branching points, where d is the maximum degree of a node (Section 4.6). Each branching point generates at most d GROW messages. Hence, the total number of GROW messages is finite. Every GROW message follows the unicast path to the destination and thus terminates in finite time. Each GROW message may or may not generate one BREAK message. Hence, there are finite BREAK messages. Every BREAK message follows a multicast tree branch backward to prune the branch and terminates when reaching a non-leaf node. There are one JOIN message and one CONSTRUCTION message. Both of them follow the unicast path and terminate in finite time. Because there are finite control messages that all terminate in finite time, SoMR-m terminates in finite time. □

5.2 Performance Analysis

We analyze the length of the tree branch established by SoMR-m and the overhead of SoMR-m.

SoMR-m prefers the shortest path provided by the underlying unicast routing protocol. However, when the shortest path does not satisfy the QoS requirement, the GROW messages are sent along other paths. Though it increases the chance of success, too long paths are often undesired due to larger consumption of resources. The following theorem shows that SoMR-m can make sure that the increase in path length is bounded.

Theorem 4 *Suppose the unicast routing paths are the shortest paths in terms of hops. For SoMR-m, a tree branch from the root to a member is at most $2m$ hops longer than the shortest path. If the directivity is implemented, a tree branch is at most two hops longer than the shortest path.*

[7] This is different from PIM [6], which grows branches from new members to the tree.

Proof: When a tree branch is constructed, SoMR-m allows at most m branching points on the branch. A GROW message always travels along the shortest path unless at the branching points. At each branching point, GROW may travel one hop backward away from the new member, which increases two hops over the shortest path. There are at most m branching points, which can increase at most $2m$ hops over the shortest path. Hence, any constructed tree branch to a new member is no more than $2m$ hops longer than the shortest path.

When the directivity is implemented, according to Section 4.6, if a GROW message does not follow a shortest path after a branching point, its MBL is set to be zero and no more branching is allowed. Therefore, a GROW message can travel backward only once. Hence, any constructed tree branch is at most two hops longer than the shortest path. □

In the following, we compare the worst case overhead of three protocols: spanning join, QoSMIC, and SoMR. To simplify the problem, we consider a network of n uniformly connected nodes. Let the diameter of the network be 2ω hops. Assume the number of nodes in the k-neighborhood of a node, $\mathcal{N}_k$, grows quadratically with k, i.e., $\mathcal{N}_k = \alpha k^2$. Thus, the diameter is given by $\alpha\omega^2 = n$. When the spanning join protocol broadcasts in a neighborhood with a radius of k hops, the number of messages sent is αk^2. Hence, in the worst case the total number of messages sent in consecutive broadcasts are

$$\sum_{k=1}^{\omega}(\alpha k^2) = \alpha\frac{\omega(\omega+1)(2\omega+1)}{6} \approx \frac{n(2\omega+1)}{6} \in O(n\omega)$$

The local search of QoSMIC broadcasts in a small neighborhood with a constant radius. The message overhead of this part can be considered as a constant. Let T be the size of the multicast tree. The worst case overhead of the tree search is $O(T)$. For a dense tree that populates the entire network, $O(T) = O(n)$.

Consider SoMR-m with MBD $= x$. The maximum number of branching points is $\frac{x^m-1}{x-1}$ (Section 4.6). The maximum number of branches is $x\frac{x^m-1}{x-1}$. Note that x and m are both small constants. The length of any branch is bounded by $O(2\omega)$. Therefore, the total number of messages sent is bounded by $O(x\frac{x^m-1}{x-1}2\omega) = O(\omega)$ in the worst case. We shall do a more detailed study on overhead in Section 6 by simulation. What the above analysis tells us is that, as the network size increases, the worst case overhead of spanning join, QoSMIC, and SoMR increases in the order of $n\omega$, n, and ω, respectively.

For a perfect uniformly-connected network with every node degree being d, $\omega = O(\sqrt[d/2]{n})$. Among the three, SoMR is the least sensitive to the size of the network, which means better scalability.

6 Simulation

Extensive simulations were conducted to study the performance of SoMR. Two performance metrics, *success ratio* and *average message overhead*, are defined as follows.

$$\text{success ratio} = \frac{\text{number of successful joins}}{\text{total number of join requests}}$$

$$\text{avg. msg. overhead} = \frac{\text{total number of messages sent}}{\text{total number of join requests}}$$

When the message overhead is calculated, sending a message over a path of l hops is counted as l messages.

Four multicast routing protocols were simulated: *SPR*, *SoMR-3*, *QoSMIC* [12], and *spanning-joins* [10]. SPR stands for single-path routing, which is the traditional way of connecting a new member to the multicast tree by using the unicast path from the member to the sender (or core). Under a delay constraint, the join fails if the path has too large delay. The maximum branching degree of SoMR-3 is 5, i.e., a branching point can send at most 5 GROW messages to its neighbors. For QoSMIC, the local search and the tree search are implemented as sequential procedures; the tree search is executed only when the local search fails. Comparing with the parallel implementation of these two search procedures, the sequential implementation minimizes the overhead, but may introduce additional delay. *Directivity*, *local minima*, and *fractional choice* [12] were also implemented. For spanning joins, we implemented its directed flooding version, called *directed spanning joins* [10]. We assume a unicast routing protocol providing the shortest path in term of hops between each pair of nodes.

Our simulations were conducted on Power-Law network topologies [31] and Waxman network topologies [32]. The Power-Law topologies are based on the results reported in [31], which showed that the node degrees in the Internet obey a power-law distribution. We used a topology generator described in [33]. For the Waxman based topologies we used the Waxman method [32] that spreads nodes randomly on a grid and adds links randomly, such that the probability of a link to be included decreases

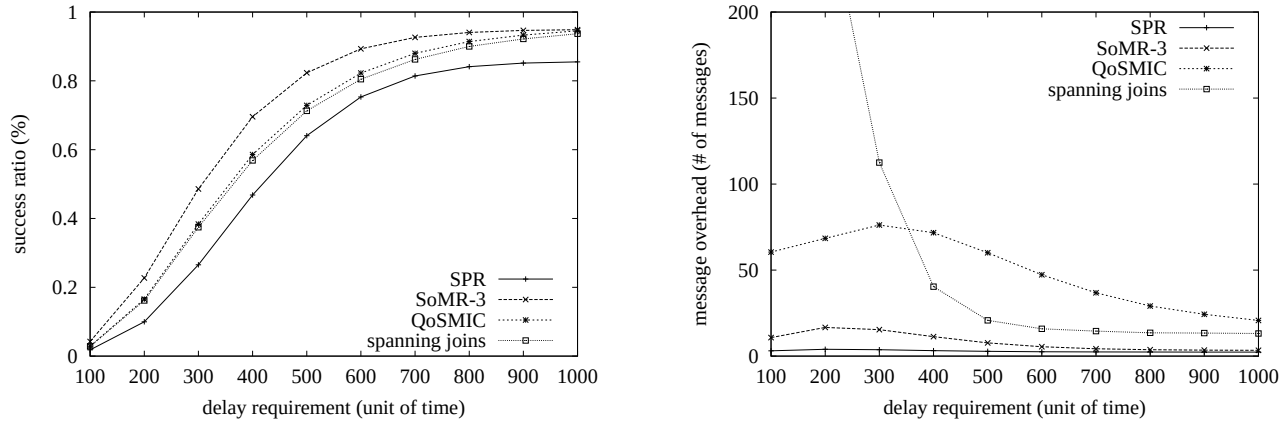

Figure 6: Power-Law topology, 600 nodes, 5% links saturated

exponentially with its length.

In the simulation, a small percentage of links in the topology are randomly selected as *saturated links*, which refuse to accept more QoS traffic due to the lack of resources. The delays of these links are thus considered to be infinite.[8] When the unicast path has a link that is saturated for QoS traffic, SPR will fail but the other protocols may still succeed because they explore more paths than the unicast one. The delays of the remaining links are uniformly distributed in the range of [0, 200] units of time.

In each simulation run, the link delays are first randomly generated, the root of the tree is randomly selected, and a delay requirement for the multicast tree is set. Then, the nodes in the network start to join the tree in a random order; each node attempts once. Upon completion, the next simulation run starts. Two hundred simulation runs are conducted on each of six randomly generated topologies. The average result (success ratio/message overhead) of all simulation runs yields one data point in the figure. The standard deviation is less than 4% of the value of the data point.

Fig. 6 compare the success ratio and the message overhead of the four routing protocols. The horizontal axis represents different delay requirements of the multicast trees. Power-Law topologies with 600 nodes are used. Five percent of links are saturated links. The figure shows that the success ratio of SoMR-3 is better than those of QoSMIC and spanning joins. Remarkably, SoMR-3 achieves better

[8]Each link typically has a "quota" on the maximum amount of resources allowed to be reserved for QoS traffic in order not to starve the best-effort traffic. Once this quota is reached, the link refuses to accept more QoS traffic. It is then a *saturated link*. Note that the delay of a saturated link is infinite for new QoS traffic but is not infinite for the best-effort traffic. While the underlying unicast routing algorithm works on the best-effort traffic, it may select saturated links on its routing paths.

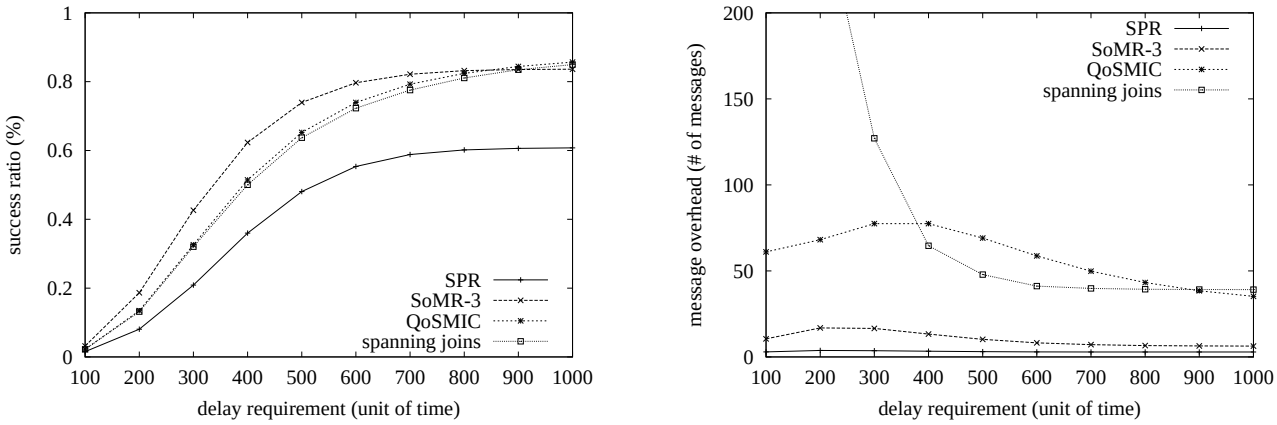

Figure 7: Power-Law topology, 600 nodes, 15% links saturated

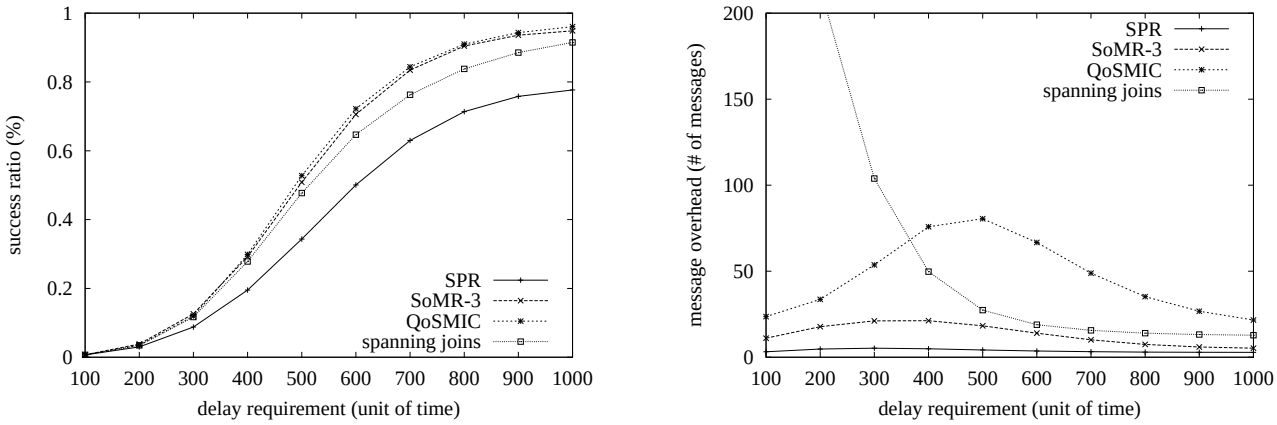

Figure 8: Waxman topology, 600 nodes, 5% links saturated

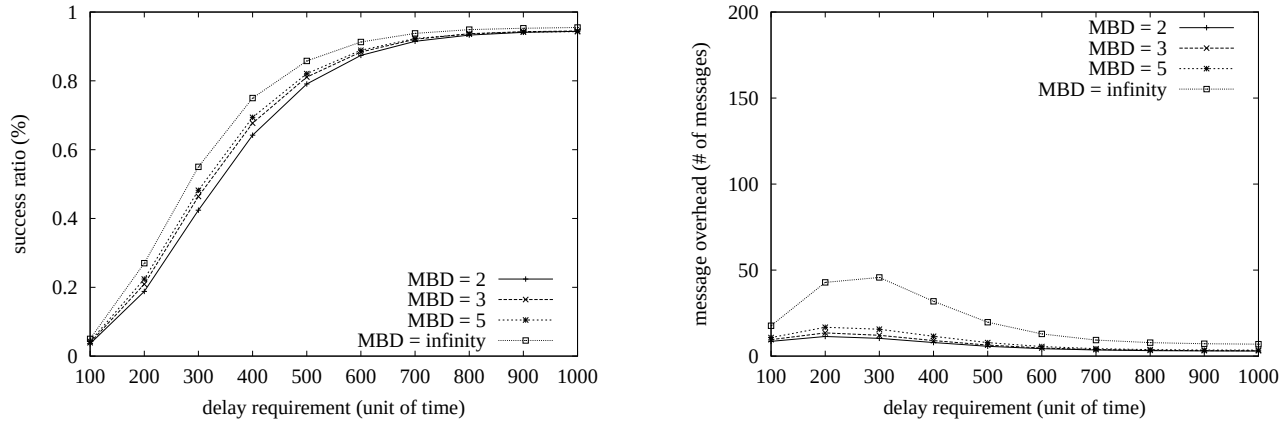

Figure 9: Power-Law topology, 600 nodes, 5% links saturated, SoMR-3, different maximum branching degrees

success ratio at much lower message overhead, as shown in the right plot. When the delay requirement is small (i.e., 100), the spanning joins protocol has very large overhead (more than 600 messages per join request). That is because the multicast tree is always small and most join requests result in large scale flooding. Although the overhead of SoMR-3 is higher than that of SPR, it is worth mentioning that for join requests SPR is able to find feasible paths, SoMR-3 behaves just like SPR and thus has the same overhead. Only for join requests SPR is unable to find feasible paths, SoMR-3 sends more control messages.

The reasons for SoMR-3 achieving low overhead and high success ratio are as follows: (1) it becomes SPR if the shortest path succeeds, (2) it branches out to search more paths only when needed, (3) it branches at the ”right” places where the EW test fails, and (4) it allows only limited number of branching points.

We repeated the above simulation with different percentage of saturated links (Fig. 7) and different type of topology (Fig. 8). Similar results were always observed. In Fig. 7, 600-nodes Power-Law topologies are used with 15% of saturated links. In Fig. 8, 600-nodes Waxman topologies are used with an average node degree of 3.5. In all our simulations, the success ratio of SoMR-3 is better than or comparable to those of QoSMIC and spanning joins, and the overhead of SoMR-3 is lower. For small or medium delay requirements, the overhead of SoMR-3 is often significantly lower than those of QoSMIC and spanning joins.

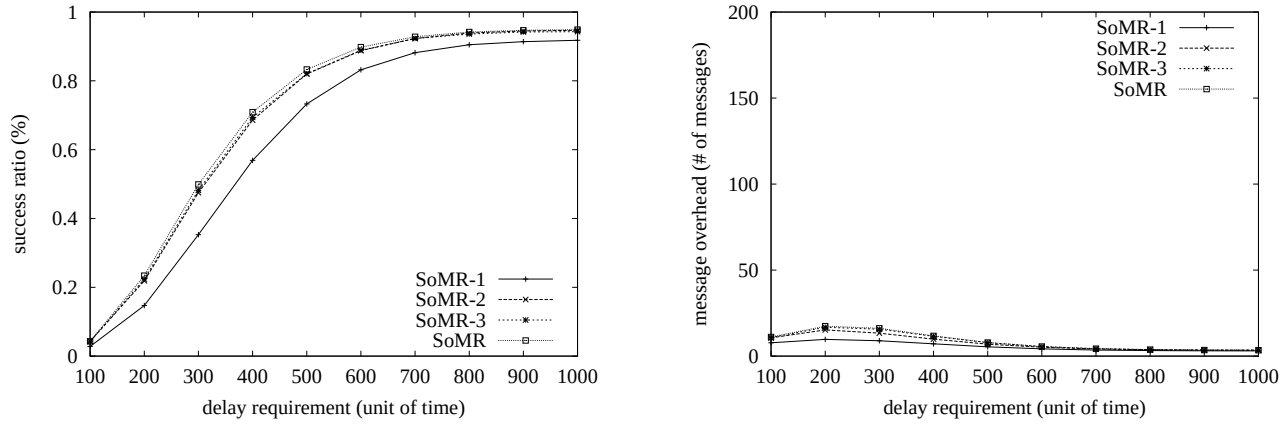

Figure 10: Power-Law topology, 600 nodes, 5% links saturated, MBD = 5, different maximum branching levels

In Section 4.6, we propose optimization techniques, particularly, maximum branching level (MBL) and maximum branching degree (MBD), to control the worse-case overhead. Here we perform simulations to evaluate their impact on the performance/overhead tradeoff for the average case. First, we let MBL $= 3$ (i.e., SoMR-3) and vary MBD from 2 to infinity. When MBD is infinity, the technique of maximum branching degree is effectively turned off. Figure 9 shows that a larger MBD will lead to increase in both success ratio and message overhead. However, the overhead is not very large even when MBD is infinity, but a modest value for MBD (such as 5) achieves a better balance between performance and overhead. Second, we let MBD $= 5$ and vary MBL from 1 to infinity. When MBL is infinity, the technique of maximum branching level is effectively turned off. Figure 10 shows that a larger MBL will also lead to increase in both success ratio and message overhead. There is not much performance/overhead difference between SoMR-3 and SoMR (with MBL = infinity). However, to guard against large worse-case overhead, SoMR-3 is preferred.

7 Conclusion

We presented SoMR, a new QoS multicast routing protocol that has a favorable tradeoff between the communication overhead and the success probability. It was shown that the protocol overhead is lower then previously suggested protocols, spanning join and QoSMIC, while its success probability is higher

in most cases than other protocols (In some cases QoSMIC has comparable success probability but with higher overhead). The protocol maintains no state in the network and works with both additive and non-additive QoS requirements.

References

[1] S. Chen and Y. Shavitt, "A Scalable Distributed QoS Multicast Routing Protocol," *IEEE International Conference on Communications*, June 2004.

[2] F. K. Hwang and D. S. Richards, "Steiner Tree Problems," *Networks*, vol. 22, pp. 55–89, 1992.

[3] T. Ballardie, P. Francis, and J. Crowcroft, "An Architecture for Scalable Inter-domain Multicast Routing," *ACM SIGCOMM*, pp. 85–95, September 1993.

[4] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.-G. Liu, and L. Wei, "An Architecture for Wide-Area Multicast Routing," *ACM SIGCOMM*, pp. 126–135, August 1994.

[5] S. Deering, "Host extensions for IP multicasting," Aug. 1989, Internet RFC 1112.

[6] 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)," June 1998, Internet RFC 2362.

[7] J. Moy, "Multicast extensions to OSPF," Mar. 1994, Internet RFC 1584.

[8] S. Chen, K. Nahrstedt, and Y. Shavitt, "A QoS-Aware Multicast Routing Protocol," *IEEE JSAC*, vol. 18, no. 12, pp. 2580–2592, Dec. 2000.

[9] B. Wang and C-J Hou, "A survey on multicast routing and its QoS extension: problems, algorithms, and protocols," *IEEE Network*, January/February 2000.

[10] K. Carlberg and J. Crowcroft, "Building Shared Trees Using a One-to-Many Joining Mechanism," *Computer Communication Review*, pp. 5–11, January 1997.

[11] I. Cidon, R. Rom, and Y. Shavitt, "Multi-Path Routing Combined with Resource Reservation," *IEEE INFOCOM'97*, pp. 92 – 100, April 1997.

[12] M. Faloutsos, A. Banerjea, and R. Pankaj, "QoSMIC: Quality of Service sensitive Multicast Internet protoCol," *SIGCOMM'98*, September 1998.

[13] S. Chen and K. Nahrstedt, "Distributed QoS Routing in Ad-Hoc Networks," *IEEE JSAC, special issue on Ad-Hoc Networks*, Aug. 1999.

[14] S. Pradhan, Y. Li, and M. Maheswaran, "QoS-Aware Hierarchical Multicast Routing on Next Generation Internetworks," *20th IEEE International Performance, Computing, and Communications Conference (IPCCC 2001)*, April 2001.

[15] K.-S. Lui, J. Wang, L. Xiao, and K. Nahrstedt, "QoS Multicast Routing with Heterogeneous Receivers," *IEEE Global Communications Conference (Globecom 2003)*, December 2003.

[16] H.-Y. Tyan, J. Hou, B. Wang, and Y.-M. Chen, "QoS Extension to the Core Based Tree Protocol," *NOSSDAV'99*, 1999.

[17] B. Rong, M. Bennani, M. Kadoch, and A. K. Elhakeem, "Bandwidth Fragmentation Avoided QoS Multicast Routing by Employing Admission Control," *19th International Conference on Advanced Information Networking and Applications (AINA'05)*, 2005.

[18] A. Striegel and G. Manimaran, "A survey of QoS multicasting issues," *IEEE Communications*, vol. 40, no. 6, June 2002.

[19] L. Li and C. Li, "QoS Multicast Routing in Networks with Uncertain Parameters," *International Parallel and Distributed Processing Symposium (IPDPS'03)*, 2003.

[20] M. Charikar, J. Naor, and B. Schieber, "Resource optimization in QoS multicast routing of real-time multimedia," *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, 2004.

[21] A. Sai Sudhir, G. Manimaran, S. Tirthapura, , and P. Mohapatra, "Heterogeneous QoS multicast in DiffServ-like networks," *Poster paper, IEEE INFOCOM*, March 2005.

[22] D. Ferrari and D. C. Verma, "A Scheme for Real-Time Channel Establishment in Wide-Area Networks," *IEEE JSAC*, vol. 8, no. 3, pp. 368–379, April 1990.

[23] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A New Resource ReSerVation Protocol," *IEEE Network*, September 1993.

[24] H. Zhang and S. Keshav, "Comparison of Rate-Based Service Disciplines," *ACM SIGCOMM'91*, 1991.

[25] H. Zhang, "Service Disciplines For Guaranteed Performance Service in Packet-Switching Networks," *Proceedings of the IEEE*, vol. 83, no. 10, October 1995.

[26] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos, "Multicast Routing for Multimedia Communication," *IEEE/ACM Transactions on Networking*, June 1993.

[27] G. N. Rouskas and I. Baldine, "Multicast Routing with End-to-End Delay and Delay Variation Constraints," *IEEE Journal on Selected Areas in Communications*, April 1997.

[28] D. Zappala, "Alternate Path Routing for Multicast," *IEEE INFOCOM*, March 2000.

[29] Handley and V. Jacobson, "SDP: Session Directory Protocol (draft 2.1)," *Internet Draft — Work in Progress*, February 1996.

[30] S. Chen and K. Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions," *IEEE Network, Special Issue on Transmission and Distribution of Digital Video*, Nov./Dec. 1998.

[31] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," *ACM SIGCOMM 1999*, Aug. 1999.

[32] B. M. Waxman, "Routing of Multipoint Connections," *IEEE Journal of Selected Area in Communications*, pp. 1617–1622, Dec. 1988.

[33] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On placement of internet instrumentation," *IEEE INFOCOM 2000*, pp. 295 – 304, Mar. 2000.