

BEAM: A Distributed Aggregated Multicast Protocol Using Bi-directional Trees

Jun-Hong Cui, Li Lao, Dario Maggiorini, and Mario Gerla

Computer Science Department, University of California, Los Angeles, CA 90095

Abstract—IP multicast confronts a severe scalability problem when there are large numbers of multicast groups in the network due to state explosion and control explosion. In backbone networks, this state scalability problem is exacerbated, since there are potentially enormous multicast groups crossing backbone domains. To improve the state scalability of multicast in backbone domains, in this paper, we propose a scalable protocol, called *BEAM* (Bi-dirEctional Aggregated Multicast), which uses the concept of aggregated multicast [11]. *BEAM* is a distributed protocol using bi-directional trees. It is simple and easy to implement. Through simulations, we show that *BEAM* can greatly improve state scalability with very low overhead: up to 98% state and tree setup and maintenance overhead reduction with less than 0.14 bandwidth waste in our experiments.

I. INTRODUCTION

Compared with multiple unicast, IP multicast is efficient for data delivering from one to many or many to many points. It uses a tree delivery structure with data only duplicated at branching nodes. However, conventional IP multicast routing protocols confront a severe scalability problem when there are large numbers of multicast groups ongoing in the networks. This is mainly due to two issues:

1. State explosion: each router needs to maintain separate states for individual groups (or group/sources). Large numbers of groups mean large amount of state to be maintained at routers, which translates into large memory requirement and slow packet forwarding.

2. Control explosion: conventional IP multicast protocols establish and maintain a multicast tree per-group or per-group/source. Large numbers of groups mean large numbers of trees to set up and maintain. Consequently, the number of corresponding tree setup and maintenance control messages will become huge and explode.

In this paper, this kind of scalability problem is referred as “state scalability problem”. In backbone networks, the state scalability problem will be exacerbated, since there are potentially enormous multicast groups crossing backbone domains. A backbone domain is typically a concentration point of the global network, and its performance greatly influences the global network’s performance. Hence, the efficiency of inter-domain multicast routing might be degraded tremendously if a conventional multicast protocol is employed as it is in backbone networks when there are a large number of concurrent active multicast groups.

The state scalability problem has prompted some recent research in forwarding state reduction. Some architectures aim to completely eliminate multicast state at routers using application level multicast [13, 5, 18], which pushes the complexity to the end-points. These solutions might be good alternatives for small-scale multicast applications; however, it is difficult for them to scale up to support multicast applications with millions of group members like Internet TV, since they need to discover and maintain all or most group members. Some schemes attempt to reduce forwarding state at non-branched routers [23, 21, 6], but they mainly target networks with a large number of sparse groups. Some other schemes try to achieve state reduction by forward-

ing state aggregation at routers [20, 22]. Thaler and Handley analyze the aggregatability of forwarding state in [22] using an input/output filter model. Radoslavov et al. propose algorithms to aggregate forwarding state and study the bandwidth-memory tradeoff with simulations in [20]. However, these state aggregation schemes attempt to aggregate routing state after the distribution trees have been established, and they tend to change the state format maintained at routers, which is generally not desired by many service providers [7]. Furthermore, the state aggregatability of this type of schemes heavily depends on multicast address allocation. It should be noted that all the above mentioned schemes or architectures are dedicated to solve the state explosion issue without examining the control explosion one.

To improve the state scalability of IP multicast, we have proposed a novel scheme, called *aggregated multicast* [11]. Aggregated multicast targets intra-domain multicast provisioning in the transit domain, especially the backbone domain. In this scheme, multiple multicast groups are aggregated at incoming edge routers to share a single distribution tree (which is called an *aggregated tree*) and de-aggregated at outgoing edge routers. In this way, core routers need to keep state only per aggregated tree instead of per group. This can significantly reduce the total number of trees in the network and thus reduce forwarding state. Thus, aggregated multicast solves both the state and control explosion issues. The trade-off is that this approach may waste extra bandwidth to deliver multicast data to non-group-member nodes. In our earlier work [11, 12], we introduced the basic concept of aggregated multicast, and gave an analysis of the trade-off between aggregation and bandwidth waste at scheme level. In this paper, we propose a distributed aggregated multicast protocol using bi-directional trees to improve the state scalability of multicast in backbone domains, which we call *Bi-dirEctional Aggregated Multicast protocol (BEAM)*. We design the detailed *BEAM* protocol. It is simple and easy to implement. Through simulations, we show that *BEAM* can obtain significant multicast state and tree management overhead reduction while only introducing very low overhead.

The remainder of this paper is organized as follows. Section II gives an overview for inter-domain multicast routing and aggregated multicast respectively. Section III describes the architecture and protocol design of *BEAM* in detail. Then in Section IV, we provide a performance evaluation of *BEAM* through simulations. And finally, Section V concludes our work.

II. BACKGROUND

A. Inter-domain Multicast Routing

The Internet consists of numerous domains (or Autonomous Systems (AS)). Domains may be connected as service provider/customers in a hierarchical manner or connected as peering neighbors, or both. Normally a domain is controlled by a single entity and can run an intra-domain multicast routing protocol of its choice, such as DVMRP [17], PIM-DM [9], MOSPF [16], PIM-SM [4], CBT [4], etc. An inter-domain mul-

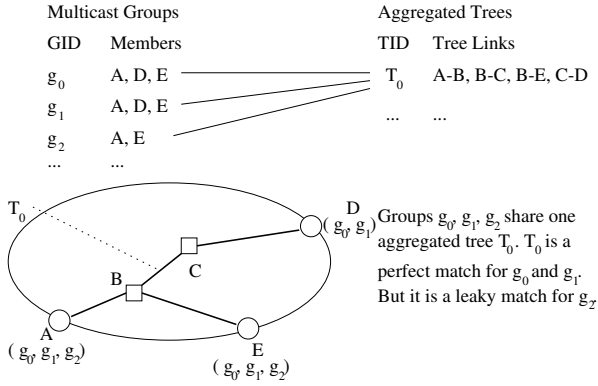


Fig. 1. Illustration of aggregated multicast

icast routing protocol is deployed at border routers of a domain to construct multicast trees connecting to other domains. A border router capable of multicast communicates with its peer(s) in other domain(s) via inter-domain multicast protocols and routers in its own via intra-domain protocols, and forwards multicast packets across the domain boundary.

Currently there are two prominent inter-domain multicast protocol suits: MBGP/PIM-SM/MSDP and MASC/BGMP [3, 15]. MBGP/PIM-SM/MSDP is a short-term solution: MSDP (Multicast Source Discovery Protocol) is responsible for detecting the existence of active sources in all peer domains; PIM-SM is used to establish a multicast tree between domains which have group members; while MBGP determines the next hop information because a join message needs to know the best reverse path toward the source. This protocol set is built on existing protocols, and thus easy to implement. However, due to its scalability problem caused by MSDP message flooding and many other issues [3], it only acts as an intermediate solution. MASC/BGMP is considered as a long-term solution, which tries to build a true inter-domain routing protocol. The key idea of BGMP is to construct bi-directional core-based trees between domains, having a single root domain for each multicast group. This protocol relies on the belief that inter-domain dependencies can be avoided by using a strict address allocation. MASC is used to allocate address between domains which guarantees that address collisions are immediately resolved. In MASC/BGMP architecture, each domain can deploy any multicast routing protocol according to its own choice.

B. Aggregated Multicast

Aggregated multicast [11] is a scheme proposed to improve multicast state scalability. The key idea is to force multicast groups to share a single distribution tree. This enforcement takes place at the edge routers of the network. Data packets from different groups are multiplexed on the same distribution tree, called *aggregated tree*. Each data packet of each group is encapsulated and travels on the aggregated tree. This way, routers in the middle of the network, namely core routers, need to keep state only per aggregated tree, which are much less in number than the groups they are servicing. Of course, edge routers of the network need to maintain sufficient information to multiplex and demultiplex groups in and from aggregated trees. Fig. 1 illustrates the basic idea of aggregated multicast.

Aggregated multicast improves the state scalability from the

following two aspects:

1. It can reduce the required multicast state. Core routers don't need to maintain state for individual groups; instead, they only maintain forwarding state for a smaller number of aggregated trees. Thus, it solves the state explosion issue.

2. It can reduce the management overhead for the distribution trees. First, there are fewer trees that exchange refresh messages. Second, tree maintenance can be a much less frequent process than in conventional multicast, since an aggregated tree has a longer life span. This is an unique advantage compared with other state reduction schemes, such as [23], [20], [22], [21], and [6], etc. In other words, aggregated multicast provides a solution to the control explosion issue.

In aggregated multicast, we need to match groups to aggregated trees. The group-tree matching problem hides several subtleties. The set of the group members and the tree leaves are not always identical. A match is a *perfect* for a group, if all the tree leaves have group members. A match may also be a *leaky match*, if there are leaves of the tree that do not have group members. In other words, we send data to parts of the tree that is not received by anyone. A disadvantage of the leaky match is that some bandwidth is wasted to deliver data to nodes that are not members for the group. Namely, we trade off bandwidth for state scalability. In an aggregated multicast protocol, a logical entity is required to conduct group-tree matching, and it can be implemented in a distributed or centralized way. We refer this logical entity as *tree manager*.

III. BI-DIRECTIONAL AGGREGATED MULTICAST (BEAM)

In this section, we describe our proposed protocol: BEAM (Bi-dirEctional Aggregated Multicast), which is designed to improve the multicast state scalability of the target backbone domain. The goal of our design is to achieve simplicity and easy implementation while significantly improve state scalability with low overhead. We start with an overview of BEAM and then give a more detailed description of the protocol.

A. An Overview

BEAM is targeted at single backbone domains. It can be built on top of any inter-domain multicast routing architecture, and its only required information from the inter-domain multicast routing protocol is group membership, that is, what edge routers are members of a multicast group (MASC/BGMP provides this information naturally).

The backbone domain which implements BEAM as the intra-domain multicast routing protocol is called a *BEAM domain*. BEAM uses the aggregated multicast concept, that is, multiple groups are forced to share a single aggregated tree. Each aggregated tree is assigned a multicast address, which is unique in the BEAM domain and transparent to other domains. Data packets are encapsulated at incoming edge routers, transmitted on aggregated trees, and decapsulated at outgoing edge routers.

To accommodate any type of group request: single source or multiple source, BEAM adopts a core-based multicast routing approach. Each aggregated tree is associated with a core. In addition, to simplify the address allocation of aggregated multicast, we borrow the idea of Simple Multicast [19], that is, each aggregated multicast is identified by a combination of the IP address of its core and a multicast address (D class). In this way, the address assignment of aggregated trees can be totally controlled by the

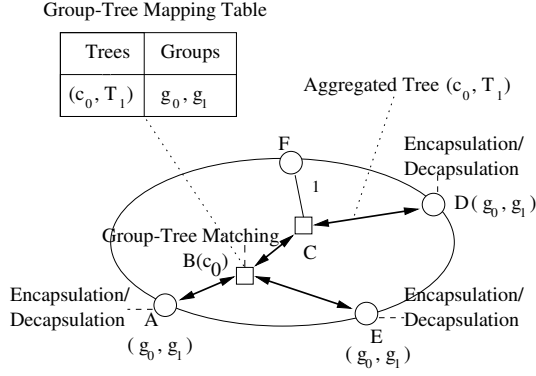


Fig. 2. A big picture of BEAM, where router B is a core (c_0) , groups g_0 and g_1 share the bi-directional aggregated tree (c_0, T_1) .

cores, and it is independent of the address allocation of multicast groups. It should be noted that, in this paper, we mainly discuss multicast groups of ASM (Any-Source Multicast) [8], where a group is identified solely by a class D address. In fact, aggregated multicast can also help to improve the state scalability of SSM [14], which is out of the scope of this paper though.

To improve the state scalability in a further step, the aggregated trees in BEAM are designed to be bi-directional. The rationale behind this is, whenever a bi-directional tree covers the members of a group, it can be used for packet delivering for the group, without being checked for transmission direction (which is necessary for unidirectional trees however). In this way, more groups can share a single tree, which means more state reduction and fewer aggregated trees.

Similar to PIM-SM/CBT, BEAM uses the bootstrap mechanism [10] to advertise the set of possible cores within the BEAM domain. When an edge router receives a join message for a group G , it can determine a core using a hash function (which is referred as *group-to-core hashing function*). This core is referred as the default core c_0 for the group G . After receiving a request from G relayed by the corresponding edge router, c_0 will find or establish a proper aggregated tree for group G .

In a nutshell, the underlying multicast routing protocol of BEAM is CBT with core-based addressing (which is actually a simplified version of Simple Multicast). And the cores conduct group-tree matching. A big picture of BEAM is illustrated in Fig. 2.

In BEAM, the multicast routing protocol for constructing and maintaining aggregated trees has existed, which we will not describe in any further detail. In the following section, we will present the new machinery needed for group-tree matching.

B. Group-Tree Matching in Cores

In a core c_0 , to match groups to aggregated trees, the tree manager module (defined in Section II) needs to maintain group and tree information and manipulate a group-tree matching algorithm. In this section, we present a simple algorithm with small additional control overhead.

First, we define a bandwidth waste function. As we said, leaky match introduces bandwidth waste while leads to more state and tree management overhead reduction. This trade-off is allowed by BEAM. Assume an aggregated tree (c_0, T) (an aggregated tree is identified by a combination of its code's IP address and a class D address) is used by groups $G_i, 1 \leq i \leq n$ (a multicast

group is identified by a class D address), each of which has a "native" tree $(c_0, T_0(G_i))$ (a native tree of a group exactly covers the group members with no data delivered to non-member nodes, and it can be computed directly by a multicast routing algorithm, for example, PIM-SM or CBT). With the assumption that all multicast groups have same bandwidth request, the **average bandwidth waste** for (c_0, T) is defined as:

$$\begin{aligned} \delta((c_0, T)) &= \frac{n \times C((c_0, T)) - \sum_{i=1}^n C((c_0, T_0(G_i)))}{\sum_{i=1}^n C((c_0, T_0(G_i)))} \\ &= \frac{n \times C((c_0, T))}{\sum_{i=1}^n C((c_0, T_0(G_i)))} - 1, \end{aligned} \quad (1)$$

where $C((c_0, T))$ is the cost of tree (c_0, T) (total cost of all (c_0, T) 's links). Intuitively, $\delta((c_0, T))$ reflects the amount of extra bandwidth wasted to carry multicast traffic using the shared aggregated tree (c_0, T) in percentage.

It should be noted that, to compute the average bandwidth waste using function (1), the tree manager module has to know the tree (including aggregated trees and native trees) topology information, which adds more complexity to the protocol. We propose an **estimated bandwidth waste** function that will take into account only the group member and tree leaf information, which is defined as

$$\begin{aligned} \delta_{est}((c_0, T)) &= \frac{n \times |leaf((c_0, T))|}{\sum_{i=1}^n |leaf((c_0, T_0(G_i)))|} - 1 \\ &= \frac{n \times |leaf((c_0, T))|}{\sum_{i=1}^n |G_i|} - 1, \end{aligned} \quad (2)$$

where $leaf((c_0, T))$ is the leaf set of tree (c_0, T) , and $|S|$ is the cardinality of set S . We use this function as an approximation of the real bandwidth waste on the aggregated tree (c_0, T) , though it will overestimate the waste, since we assume each level of in-tree nodes has the same waste as the tree leaf nodes (at the lowest level of the delivery tree). In the real implementation, we can determine good control thresholds which reflect the real bandwidth waste, as will be demonstrated in our simulation study.

B.1 A Group-Tree Matching Algorithm with Member Dynamics

To conduct group-tree matching based on function (2), each core needs to maintain a group table (with member information for each group), a tree table (with leaf information for each tree) and a group-tree mapping table. When a core c_0 receives a join or leave message of a group G (suppose c_0 is the default core of G), it conducts a group-tree matching algorithm as follows: (b_t is the given estimated bandwidth waste threshold)

(1a) If G becomes empty (the last member leaves), delete the corresponding entry in the group table. If G 's delivery aggregated tree becomes idle, the tree will be detached and the group-tree mapping table is updated.

(1b) If G is new (the first member joins), insert a new entry to the group table and go to step 2.

(1c) If G is neither new nor empty, update G 's member information, and check if G 's aggregated tree can still cover the whole group and the estimated bandwidth waste of the tree is less than b_t . If any of the conditions is not satisfied, go to step 2.

(2) Check if any existing tree can cover G . If yes, for any tree (c_0, T) which can cover group G , compute the bandwidth waste $\delta_{est}((c_0, T))$ according to (2). If $\delta_{est}((c_0, T)) \leq b_t$, then

(c_0, T) is a candidate. Among all candidates, choose the one with minimum bandwidth waste as the delivery tree for G and the corresponding tables are updated. If there is no existing tree which can cover group G , a new tree will be established using the underlying multicast routing protocol.

In the above algorithm, for each group, we only consider the trees within its default core c_0 . To achieve better aggregation, we design a new functionality: core switch, where a group's core can be changed. If there are no existing trees to cover a group G in its default core, an existing tree in another core instead of a new established tree in its default core can be used for the data delivery of group G . In this way, more groups will share a single delivery tree. However, this benefit does not come without any overhead: the additional communication between cores is needed. In the performance evaluation section, we will examine the trade-off between the gain and overhead of core switch.

C. Edge Routers in BEAM

Edge routers are possible members of multicast groups. Whenever an edge router joins a group, it needs to know the address of the corresponding delivery tree so that it can encapsulate the data packets correctly. As we know, each edge router might participate in multiple groups, which leads to the necessity to maintain a group-tree mapping table in each edge router. Of course, only the information of groups participated and corresponding aggregated trees needs to keep.

We have discussed the main component of BEAM: group-tree matching and the functionalities of cores and edge routers. In the following, we define the messages and describe each phase of the protocol in more detail.

To facilitate our description, we divide the protocol messages into two types: M-type and B-type. The underlying multicast routing messages are M-Type, such as *M-JOIN* and *M-LEAVE*. And the messages used to help matching groups to trees are B-Type, which includes *B-JOIN*, *B-JOIN-ACK*, *B-LEAVE*, *B-TREE-SWITCH*, *B-CORE-SWITCH-REQ*, *B-CORE-SWITCH-ACK*, *B-CORE-CHANGE*, *B-SEND*, and *B-SEND-ACK*. These messages are mainly transport level messages between edge routers and cores, and their functionalities will be shown in the following sections.

D. Member Join

When an edge router r receives a request to join group G from outside domains, it first uses the group-to-core hashing function to get G 's default core c_0 , it then sends a message *B-JOIN*(G) to c_0 . c_0 will trigger its tree manager module to find or establish an appropriate aggregated tree, say, (c', T) . It should be noted that this join message might activate tree switch or core switch if the existing tree could not cover group G . Then c_0 will send back a message *B-JOIN-ACK*($G, (c', T)$) to r . If r has not joined the delivery tree (c', T) , it will graft to the tree by sending *M-JOIN*(c', T).

E. Member Leave

When an edge router r wants to leave group G , it sends a message *B-LEAVE*(G) to its default core c_0 . This action will trigger an *M-LEAVE* message when r detects that there is no other group mapped to G 's aggregated tree. On receiving of the *B-LEAVE* message, c_0 manipulates the group-tree matching algo-

rithm, which might cause tree switch or core switch, as will be described in the next sections.

F. Tree Switch

When the membership of a group G is changed, its original aggregated tree (c_0, T) might not be able to cover the group G again. In this case, the tree switch procedure (in the same core) is triggered. First, c_0 finds or establishes an appropriate tree for G , say, (c_0, T') , then it utilizes a multicast message *B-TREE-SWITCH*($G, (c_0, T')$) through the multicast tree (c_0, T) to notify all the other members of group G to join (c_0, T') and leave (c_0, T) . The member routers of group G might trigger *M-JOIN* or *M-LEAVE* messages if needed.

Examples of member join, member leave, and tree switch are illustrated in Fig. 3.

G. Core Switch

In the above section, tree switch is constrained in one core. To let more groups share one single tree, BEAM allows core switch.

During the member dynamics of a group G , its original aggregated tree (c_0, T) might become too big or too small, as will cause tree switch in c_0 . If there is no appropriate tree for G in the core c_0 , c_0 will query other cores by sending out a multicast message *B-CORE-SWITCH-REQ*(G, G 's member list) through a predefined bi-directional multicast tree (c_{core}, T_{core}) , which connects all cores (where c_{core} acts as the "super core" of the this bi-directional tree). Upon receiving the *B-CORE-SWITCH-REQ* message, each core activates its tree manager module, and checks if there is a good tree to cover group G . If yes, a unicast message *B-CORE-SWITCH-ACK*($G, (c', T')$, bandwidth waste) is sent back to c_0 . From the received trees, c_0 selects the one with smallest bandwidth waste, say, (c'', T'') as the delivery tree for G . Similarly, this will trigger a tree switch procedure by sending a multicast message *B-TREE-SWITCH*($G, (c'', T'')$) through the multicast tree (c_0, T) to notify all the other members of group G to join (c'', T'') and leave (c_0, T) . At the same time, c_0 needs to record the new core c'' for further requests of group G . For example, when a new member router r joins group G , it sends a *B-JOIN* message to G 's default core c_0 . Since the core of G is switched, c_0 will send a message *B-CORE-CHANGE*(G, c'') back to r , which will activate another *B-JOIN* message to c'' . Fig. 4 illustrates the core switch procedure through an example.

H. Non-Member Sending

A non-member node r wants to deliver data to group G . It first initiates a *B-SEND*(G) message to G 's default core c_0 , which will respond with a message *B-SEND-ACK*($G, (c', T)$), where (c', T) is the delivery tree for G and it is possible that c' is not c_0 because of core switch. When r sends data, it first encapsulates the data packets with the address of (c', T) , then unicasts them toward c' . The data packets will be intercepted by any in-tree router or received by the core, then are transmitted along the bi-directional tree (c', T) .

IV. PERFORMANCE EVALUATION

In this section, we mainly examine the following performance issues of BEAM protocol through simulations (which are conducted in NS2 [2]).

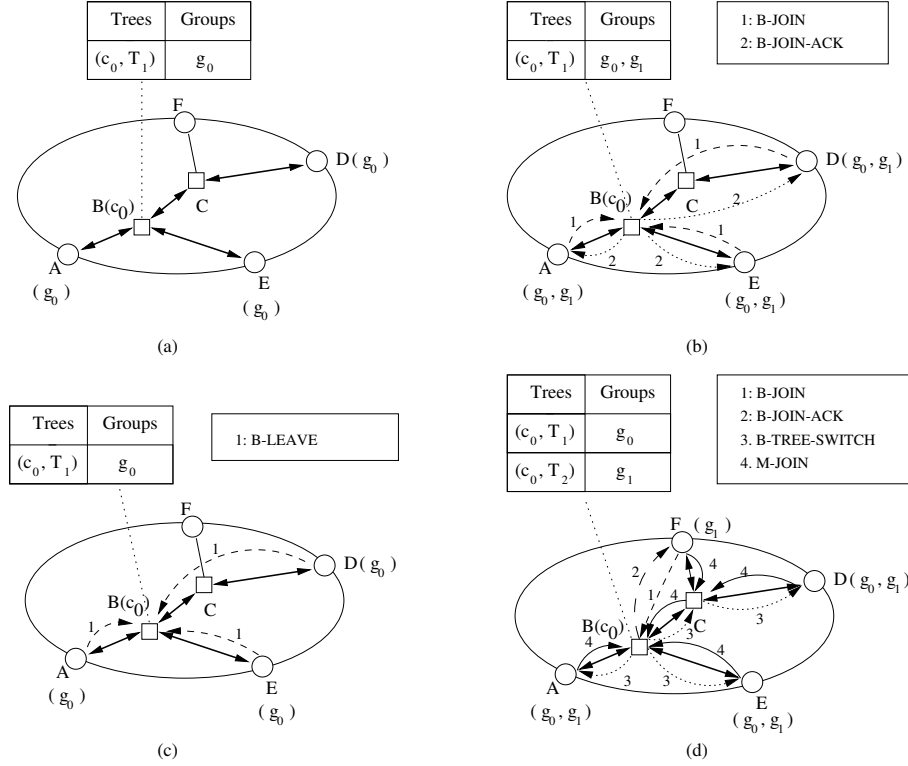


Fig. 3. (a) Initial state: group g_0 uses tree (c_0, T_1) ; (b) Member join: group g_1 starts with members A, D, E and groups g_0 and g_1 share the tree (c_0, T_1) ; (c) Member leave: group g_1 terminates with members A, D, E; (d) Tree switch: based on (b), a new member F joins group g_1 , and g_1 switches from (c_0, T_1) to (c_0, T_2) .

1. To what extent BEAM can improve state scalability.
2. The trade-off between the gain and overhead of core switch.
3. How the bandwidth waste is estimated.

We compare BEAM to CBT with core-based addressing (which is referred as *M-CBT* in this section). The difference between these two protocols is that the latter has no functionality of group-tree matching, and each group uses a delivery tree. We define the following three metrics to reflect state scalability.

State Reduction Ratio (SRR) Since multicast state in edge routers can not be reduced in any state reduction scheme, we only consider state in core routers, which is referred as *transit state*. Then SRR is defined as

$$SRR = 1 - \frac{\# \text{ of transit state entries of BEAM}}{\# \text{ of transit state entries of M-CBT}} \quad (3)$$

Tree Setup Overhead Reduction Ratio (TSORR) The number of M-JOIN messages is a measurement of tree setup overhead. Thus we define TSORR as

$$TSORR = 1 - \frac{\# \text{ of M-JOIN messages of BEAM}}{\# \text{ of M-JOIN messages of M-CBT}} \quad (4)$$

Tree Maintenance Overhead Reduction Ratio (TMORR) The measurement of tree maintenance overhead depends on how to keep multicast state: hard state or soft state. Though the underlying multicast routing protocol of BEAM is closer to CBT, BEAM does not exclude PIM-SM like semi-state maintenance. At this stage, we approximately use the number of multicast trees as a metric for tree maintenance overhead. Hence, we define TMORR as

$$TMORR = 1 - \frac{\# \text{ of multicast trees of BEAM}}{\# \text{ of multicast trees of M-CBT}} \quad (5)$$

A. Simulation Environment

In our simulations, we use a network abstracted from a real network topology, Abilene backbone [1]. This abstracted network has 12 core routers, and each is attached with an edge router. Thus, there are totally 24 routers in the target network.

To generate multicast groups more realistically, we use one of our previous group models developed in [12]: the random node-weighted model. In this model, each node is assigned a weight, which is the probability of the node to participate in multicast groups. In the target network, core routers will not be members for any multicast group and thus are assigned weight 0. Any other edge router is assigned a weight 0.2 or 0.8 according to the real-time traffic on its links connected to the corresponding core routers. The rationale behind this is, for a router, more traffic means more participation in the network communication, thus there is higher probability for it to join a multicast group.

In our simulation network, we assume that any core router can be a core. The group-to-core hashing function we use is simply a Modulo- n function from a group ID to a core ID. We also assume multicast group requests arrive as a Poisson process with arrival rate λ , and groups' life time has an exponential distribution with average μ^{-1} . Then, at steady state, the average number of groups is $\bar{N} = \lambda/\mu$. In our simulation experiments, we fix the group average life time as 100s, and change the group arrival rate in order to get different number of groups. We run the simulations for 1000s, and collect performance data after steady state is reached (after 400s in our scenario).

The group-tree matching algorithm for BEAM used in our experiments is described in Section III-B.1. Note that this algorithm considers the estimated bandwidth waste instead of real

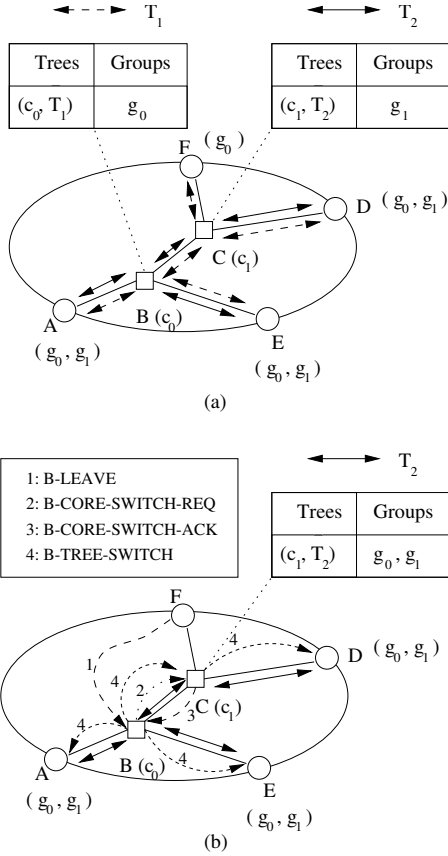


Fig. 4. (a) Before core switch: group g_0 uses tree (c_0, T_1) while group g_1 uses tree (c_1, T_2) ; (b) Core switch procedure: when member F leaves group g_0 , g_0 switches from core c_0 to c_1 and shares the tree (c_1, T_2) with group g_1 .

bandwidth waste.

B. Results and Analysis

In our experiments, we vary the estimated bandwidth waste threshold (denoted as bth) from 0 to 0.2 for BEAM. To investigate the trade-off involved in core switch, we also run BEAM when core switch is not allowed.

Fig. 5 shows the results for SRR (State Reduction Ratio) vs the average number of concurrent active groups. For all the curves in Fig. 5, we can see that the state reduction ratio increases significantly when the number of groups grows: the more groups become active, the more state reduction we expect. We also observe that, when the estimated bandwidth waste threshold (bth) is lifted, more state reduction is achieved, as verifies the trade-off between state reduction and bandwidth waste. Moreover, given any bth , the state reduction ratio for BEAM with core switch is much higher than that for BEAM without core switch. For example, when the average number of groups is 3500 and $bth = 0$ (that is, no bandwidth waste), the state reduction ratio is 77% when core switch is allowed, while it is only 41% when there is no core switch. As we mentioned before, this gain does not come without any cost: core switch needs additional control messages for the communication between cores, such as *B-CORE-SWITCH-REQ* and *B-CORE-SWITCH-ACK* messages. The control overhead depends on many factors, including possible core placement, group-to-core hashing function, and bandwidth waste threshold. From our experiments, we can observe that, when the

bth increases, the number of core switch messages decreases. As is consistent with our intuition: a group has higher probability to stay and use an existing tree at its current core if more bandwidth waste is sacrificed.

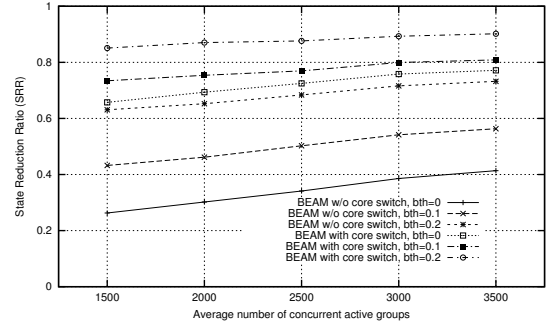


Fig. 5. State Reduction Ratio (SRR) vs average number of groups.

Fig. 6 and Fig. 7 plot the results for TSORR (Tree Setup Overhead Reduction Ratio) and TMORR (Tree Maintenance Overhead Reduction Ratio) vs the average number of concurrent active groups respectively. These two figures show very similar trends to Fig. 5: more overhead reduction ratio when there are more concurrent groups and more bandwidth are wasted. When we compare BEAM without core switch to BEAM with core switch, we draw the same conclusions as above also.

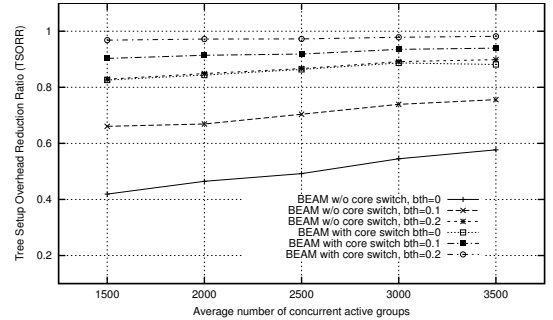


Fig. 6. Tree Setup Overhead Reduction Ratio (TSORR) vs average number of groups.

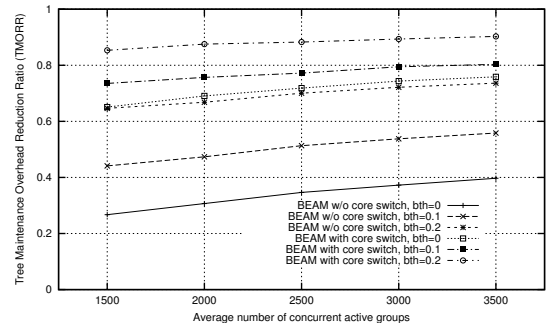


Fig. 7. Tree Maintenance Overhead Reduction Ratio (TMORR) vs average number of groups.

One concern about core switch is load concentration: since groups tend to be switched to their most suitable cores, will the load be concentrated on some cores? We use the relative standard deviation (which is equal to the standard deviation divided

by the average) of link loads to measure the load concentration. In our experiments, we assume every group has the same bandwidth requirement. Fig. 8 illustrates the corresponding results. We can see that, when there is core switch, the load concentration is no worse than the case without core switch. This is because the groups tend to concentrate on their default cores even if there is no core switch. Another observation is that, when bth is increased, the load concentration is leveraged. As is not a surprise: link loads are more evenly distributed since more bandwidth are wasted on some links which have smaller loads.

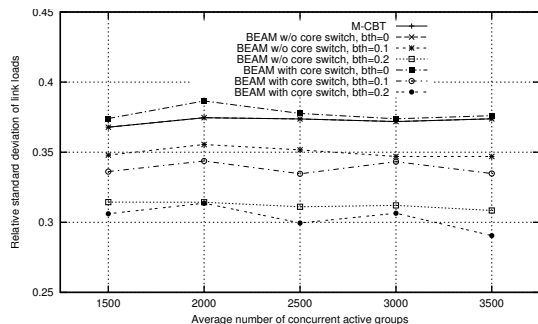


Fig. 8. Relative standard deviation of link loads.

In the group-tree matching algorithm, we compute estimated bandwidth waste instead of real bandwidth waste. Given an estimated bandwidth waste threshold, what is the real bandwidth waste? Fig. 9 provides the answer for our simulation scenario. When there is no core switch, $bth = 0.1$ corresponds to real bandwidth waste 0.037 and $bth = 0.2$ corresponds to real bandwidth waste 0.106, in average. For the core switch case, the gap between estimated and real bandwidth waste is smaller, when $bth = 0.1$ and $bth = 0.2$, the corresponding real bandwidth wastes are 0.059 and 0.136 respectively. The difference between these two cases is mainly due to the fact that groups can find appropriate aggregated trees more easily when core switch is activated.

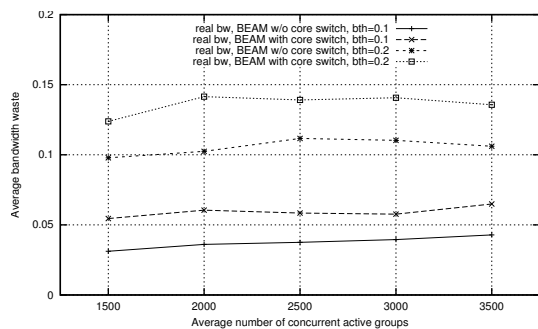


Fig. 9. The real bandwidth waste for given estimated bandwidth waste threshold.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we propose BEAM to improve the multicast state scalability of backbone domains. We design the detailed BEAM protocol and show that it is simple and easy to implement. From our simulation results and analysis, we can conclude that BEAM has greatly improved state scalability with very low overhead: up to 98% state and tree setup and maintenance overhead reduction with less than 0.14 real bandwidth waste. Moreover, BEAM

scales well with the number of groups: the more groups come, the more state and tree management overhead reduction. Finally, BEAM does not cause worse load concentration compared with M-CBT.

Future Work We would continue our work in the following two directions:

1. Evaluate the performance of BEAM in various scenario parameters, such as network topologies, group models and group-to-core hashing functions, etc.
2. Improve load balancing of BEAM. In our current protocol design, there is no explicit load balancing control. However, BEAM potentially has the ability to balance load since it allows communication between cores. And this can also help to decrease the effect of the group-to-core hashing function on load concentration.

REFERENCES

- [1] Abilene network topology. <http://www.ucaid.edu/abilene/>.
- [2] The Network Simulator - ns-2. <http://www.isi.edu/nsnam/ns/>.
- [3] K. Almeroth. The evolution of multicast: From the MBone to inter-domain multicast to Internet2 deployment. *IEEE Network*, Jan./Feb. 2000.
- [4] A. Ballardie, P. Francis, and J. Crowcroft. Core Based Trees (CBT). *Proceedings of ACM SIGCOMM*, pages 85–95, Sept. 1993.
- [5] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. *Proceedings of ACM SIGMETRICS*, June 2000.
- [6] L. H. M. Costa, S. Fdida, and O. C. M. Duarte. Hop-by-hop multicast routing protocol. *Proceedings of SIGCOMM'01*, Aug. 2001.
- [7] J. Crowcroft. Multicast Address Translation. *Internet draft: draft-crowcroft-mat-00.txt*, Nov. 2001.
- [8] S. Deering. Multicast routing in a datagram internetwork. *Ph.D thesis*, Dec. 1991.
- [9] S. Deering, D. Estrin, D. Farinacci, and V. Jacobson. Protocol Independent Multicast (PIM), Dense Mode Protocol : Specification. *Internet draft*, Mar. 1994.
- [10] D. Estrin, M. Handley, A. Helmy, P. Huang, and D. Thaler. A dynamic bootstrap mechanism for rendezvous-based multicast routing. *Proceedings of IEEE INFOCOM'99*, 1999.
- [11] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast: an approach to reduce multicast state. *Proceedings of Sixth Global Internet Symposium (GI2001)*, Nov. 2001.
- [12] A. Fei, J.-H. Cui, M. Gerla, and M. Faloutsos. Aggregated Multicast with inter-group tree sharing. *Proceedings of NGC2001*, Nov. 2001.
- [13] P. Francis. Yoid: extending the internet multicast architecture. <http://www.aciri.org/yoid/docs/index.html>.
- [14] H. Holbrook and B. Cain. Source-Specific Multicast for IP. *Internet draft: draft-holbrook-ssm-arch-03.txt*, Nov. 2001.
- [15] S. Kumar, P. Radoslavov, D. Thaler, C. Alaettinoğlu, D. Estrin, and M. Handley. The MASC/BGMP architecture for inter-domain multicast routing. In *Proceedings of ACM SIGCOMM'98*, pages 93–104, Sept. 1998.
- [16] J. Moy. Multicast routing extensions to OSPF. *RFC 1584*, Mar. 1994.
- [17] C. Partridge, D. Waitzman, and S. Deering. Distance Vector Multicast Routing Protocol. *RFC 1075*, 1988.
- [18] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. *Proceedings of 3rd Usenix Symposium on Internet Technologies and Systems (USITS 2001)*, Mar. 2001.
- [19] R. Perlman, C.-Y. Lee, A. Ballardie, J. Crowcroft, Z. Wang, T. Maufer, C. Diot, J. Thoo, and M. Green. Simple Multicast: A design for simple, low-overhead multicast. *Internet draft: draft-perlman-simple-multicast-03.txt*, Oct. 1999.
- [20] P. I. Radoslavov, D. Estrin, and R. Govindan. Exploiting the bandwidth-memory tradeoff in multicast state aggregation. Technical report, USC Dept. of CS Technical Report 99-697 (Second Revision), July 1999.
- [21] I. Stoica, T. Ng, and H. Zhang. REUNITE: A recursive unicast approach to multicast. In *Proceedings of IEEE INFOCOM'00*, Tel Aviv, Israel, Mar. 2000.
- [22] D. Thaler and M. Handley. On the aggregatability of multicast forwarding state. *Proceedings of IEEE INFOCOM*, Mar. 2000.
- [23] J. Tian and G. Neufeld. Forwarding state reduction for sparse mode multicast communications. *Proceedings of IEEE INFOCOM*, Mar. 1998.