Enabling Content Dissemination Using Efficient and Scalable Multicast

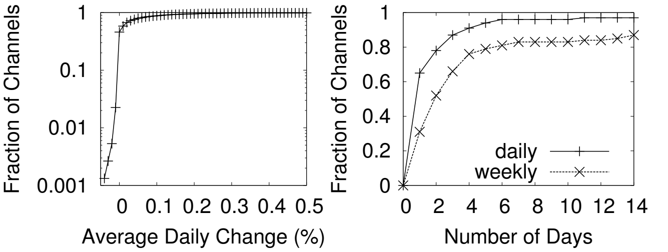
Tae Won Cho*, Machael Rabinovich†, K.K. Ramakrishnan‡, Divesh Srivastava‡, Yin Zhang*
*Univ. of Texas at Austin, †Case Western Reserve University, ‡AT&T Labs–Research
khatz@cs.utexas.edu, micha@eecs.cwru.edu, kkrama@research.att.com, divesh@research.att.com, yzhang@cs.utexas.edu

Abstract—**Multicast is an approach that uses network and server resources efficiently to distribute information to groups. As networks evolve to become information-centric, users will increasingly demand publish-subscribe based access to fine-grained information, and multicast will need to evolve to (i) manage an increasing number of groups, with a distinct group for each piece of distributable content; (ii) support persistent group membership, as group activity can vary over time, with intense activity at some times, and infrequent (but still critical) activity at others. These requirements raise scalability challenges that are not met by today's multicast techniques. In this paper, we propose the MAD (Multicast with Adaptive Dual-state) architecture to provide efficient multicast service at massive scale. MAD can scalably support a vast number of multicast groups, with varying activity over time, based on two key novel ideas: (i) decouple group membership from forwarding information, and (ii) apply an adaptive dual-state approach to optimize for the different objectives of active and inactive groups. We focus on the scalability characteristics of MAD and demonstrate through analysis, simulation and implementation that the architecture achieves high performance and efficiency.**

I. Introduction

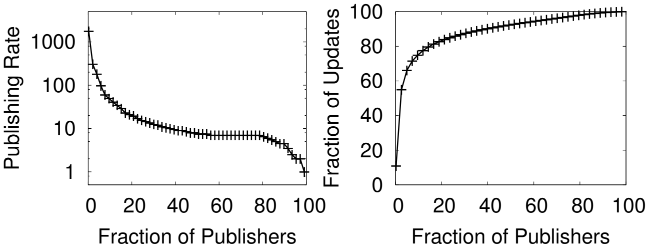
Multicast is an approach that uses network and server resources efficiently to support multipoint communication. Despite its clear performance benefit, multicast has not seen wide deployment over the past two decades. Some of the past barriers to widespread use have been the lack of support by Internet service providers and (possibly as a consequence) a lack of application demand for multicast.

Recently, however, multicast is seeing a resurgence. As the Internet evolves to become information-centric, network services increasingly demand scalable and efficient dissemination of information from a multitude of distributed information producers to large groups of interested information consumers. These information-centric services are growing rapidly in use and deployment. For example, IPTV services that use IP multicast as the underlying distribution technology are being deployed by multiple carriers. These services are gaining rapid adoption, with the number of subscribers and revenues growing rapidly [1]. Multiplayer online games (MMORPGs) are reportedly seeing 30-100% annual subscription growth [7]. Other common examples of deployed services that are information-centric include: file sharing, software updates, RSS dissemination, video conferencing, online markets, video-on-demand, and grid computing. All these services require the capability of large-scale information dissemination and can therefore benefit significantly from the communication efficiency of multicast delivery.



(a) Changes in channel subscription count (b) Channel lifetime in top 100

Fig. 1. YouTube channel characteristics.



(a) Publishing rate (# updates/month) (b) CDF of feed updates

Fig. 2. Publishing characteristics of RSS feeds

A. Requirements of Information-centric Network Services

Information-centric network services create not only new opportunities but also significant new challenges for multicast. These services exhibit several key characteristics:

Vast number of groups. Given the increasing amount of electronic content and the need to ensure that only relevant information is disseminated, multicast will need to manage an increasing number of fine granularity groups, with a distinct group for each piece of distributable content. For example, eBay lists over ten million new items every day [18], each of which can be a potential group. As a result, the number of groups that the underlying multicast architecture can support will need to significantly increase from what we typically see with IP multicast in the underlay, or with overlay multicast.

Long-lived group membership. As the network evolves to support models of information dissemination (such as publish/subscribe), membership is likely to be long-lived. Users tend to subscribe but do not unsubscribe and continue to be interested in receiving information sent infrequently by publishers. As an example, we analyzed the average daily changes in the subscription counts of 754 YouTube [26] channels after they become inactive (*i.e.*, stop appearing in any popular channel list). Figure 1(a) shows that only 2.3% of the channels experience a decrease in their subscription counts. Long-lived membership can significantly increase the state that has to be maintained in routers (that may have limited table space). Long-lived membership can also gradually increase the group size over time, resulting in higher control overhead.

Wide range of activity level across groups. Group activity tends to exhibit a skewed distribution. That is, most groups generate relatively infrequent and/or small amount of data traffic, yet a small fraction (*e.g.*, 20%) of active groups account for the vast majority (*e.g.*, 80%) of data traffic. To illustrate this 80-20 rule, we analyze the publishing activity of RSS feeds using data from [15]. Figure 2(a) shows that only 5% of the RSS feeds publish more than 100 updates/month, and the median update rate is below 10 updates/month. Figure 2(b) shows that the 10% most active RSS feeds contribute to 75% of the total feed updates. Note that the 80-20 rule is also observed in many other network applications, *e.g.*, subscription counts of RSS feeds [15], view counts of video clips [8], incoming link counts of Web pages [9], and file access frequencies of online streaming servers [11].

Dynamic activity level within a group. The activity level within a group tends to vary over time. Some new groups become active quickly, whereas other groups become dormant after the peak. To illustrate such dynamic behavior, we measure how long a channel stays in the top-100 popular channel lists in YouTube. Figure 1(b) shows that 78% channels disappear from the daily top-100 list in just 2 days after their appearance. Similarly, 80% of the channels disappear from the weekly top-100 list after 4–5 days.

To effectively support such information-centric network services, we seek to design a multicast infrastructure that can provide multicast services at *massive scale* (with a billion users, hundreds of billions of groups, and long-lived group membership), while being *efficient* across a range of group sizes and diverse, time-varying activity levels. We want to realize this goal using *today's* commercial hardware.

B. MAD Approach and Contributions

In this paper, we describe Multicast with Adaptive Dual-state (MAD), a novel architecture that can scalably support a vast number of multicast groups with diverse, time-varying activity, in an efficient and transparent manner on today's commercial hardware. MAD has the following key features.

1. MAD provides **persistence** in group membership by explicitly decoupling the membership state from the forwarding state. MAD uses a distributed state management approach to efficiently store group membership state at very large scale. For each group, a small number of routers form a *membership tree* (rooted at a core router) to maintain the membership state.
2. MAD achieves both **efficiency** in data forwarding and **scalability** in number of groups by treating active groups and inactive groups differently to optimize for different performance objectives. Specifically, messages to an *active group* are handled using any existing multicast protocol (for maximizing forwarding efficiency), whereas messages to an *inactive group* are forwarded along the membership tree (for minimizing state requirement and control overhead). Our specific instantiation of MAD uses the Core Based Tree (CBT) [2] (or a shared tree using PIM-SM [14]) for active groups due to its known efficiency and scalability. We refer to the CBT of an active group as the *dissemination tree*, in contrast to the membership tree.
3. MAD provides **transparency** in the presence of dynamic changes in group activity level. Since group activity can drastically change over time, MAD provides seamless transition mechanisms to promote active groups from inactive groups and vice versa without any end-system participation. Messages are forwarded along the dissemination tree or the membership tree based on the activity level.

Our instantiation of MAD begins as an overlay multicast service for better deployability. However, MAD can directly take advantage of alternative multicast or information dissemination capabilities when they become available. For example, messages sent to an active group can be handled by either IP multicast or peer-to-peer protocols for better forwarding efficiency. Messages sent to an inactive group are forwarded along the membership tree. In doing so, stateless multicast protocols like Xcast [4] can be used for eliminating redundant overlay unicast messages (see §IV-A).

We demonstrate through analysis, simulation and implementation that MAD can support hundreds of billions of groups with hardware that is representative of today's commercial platforms. At the same time, MAD achieves high performance and efficiency, while exploiting the wide range of activity likely to be seen across multicast groups.

II. Related Work and Limitations

Twenty years of networking research has produced many efficient mechanisms for multicast communication (*e.g.*, [2], [4]–[6], [13], [14], [19], [23]). However, these current multicast approaches implicitly couple group membership state (*i.e.*, which end hosts are members of a group) with forwarding state (*i.e.*, how to reach those group members), and use a common approach for multicast distribution for all groups – small or large, active or inactive.

IP multicast-style approaches: IP multicast has focused on efficient forwarding of information to a large active group of recipients, with the goal of efficient lookup for forwarding. IP multicast-style approaches (at the network layer [2], [14] or at the application layer with "overlay multicast" [5], [6]) try to keep a relatively small amount of state (limited number of groups and the associated interfaces downstream with recipients for the group). However, this state is maintained (as soft state) at *every* router on the multicast tree of the group for efficient forwarding. State maintenance is thus expensive — it requires a large number of routers to store the state, and involves considerable control overhead (periodic refresh and pruning) to keep the state current and refreshed. This approach is inappropriate for our problem because:

First, IP multicast-style approaches are appropriate for a relatively small number of groups. Building a cache hierarchy with SRAM and DRAM in routers for keeping state can partially relieve the memory requirements for large numbers of groups. With an increasing number of active groups, either the cache size has to be increased to maintain forwarding performance, or cache misses will degrade system performance [17]. Moreover, we would like the environment to support a range of router sizes, including small routers such as

IP DSLAMs which may only be able to support a few thousand multicast groups. It is important for the architecture to scale to large numbers of groups even with such small routers in the network.

Second, when groups are long-lived, but have little or no activity over long periods of time, maintaining the membership state in IP multicast-style approaches requires a lot of control overhead (relative to the activity) to keep it from being aged-out. If not, data sent out by sources to a relatively inactive group will likely be lost, which we believe is highly undesirable.

Third, when ISPs choose to support non-active groups with unicast (due to the limitation of the number of groups in existing multicast schemes), converting groups from multicast to unicast and vice versa requires additional intervention - allocating a multicast group address and initiating joins for those who are members of a group that becomes active, and tearing down an existing multicast group for a group that becomes inactive.

Multicast forwarding state reduction: There exist a wide range of approaches to reducing multicast forwarding state.

Pruning routers with multicast forwarding state. REUNITE [23] and [12] propose to keep multicast forwarding state only at "branching" routers, while non-branching routers use unicast routing to forward traffic and keep the multicast control table in the control plane. However, the number of branching routers can grow as the size of group increases in above schemes, increasing the multicast state. Also, multicast forwarding tables are stored by soft state, which entails high control overhead for groups with little data activity.

Aggregating multicast forwarding state. Aggregated Multicast [13] aggregates multiple base multicast groups into a single tree to achieve better scalability (at the expense of sending irrelevant content to a subset of the members). Such aggregation is complementary to our approach of separating the multicast state between forwarding and membership states. If desired, MAD can apply aggregation to further reduce multicast forwarding state.

Making multicast stateless. Stateless approaches eliminate the need for routers to maintain multicast forwarding state by storing such state in packet headers instead. For example, Explicit Multicast (Xcast) [4] encodes the list of destination nodes into every packet. Free Riding Multicast (FRM) [19] reduces routing state by caching all source-based tree edges of a group at the sources and embeds a Bloom filter (that encodes all the tree edges) into the message itself. These stateless approaches do not effectively meet the needs of large multicast groups, because they can result in excessively large packet headers. They can also reduce forwarding efficiency, because they require routers to parse the header of every packet at every hop to extract the forwarding state.

P2P content dissemination: P2P solutions have also been developed for large-scale information dissemination (*e.g.*, BitTorrent [3]). Such solutions can achieve high forwarding efficiency in disseminating popular content, However, they are less effective in a publish-subscribe environment where the information sent by the publisher is infrequent but still critical, and has to reach all the subscribers in a timely manner. In such an environment, it would be difficult for a user to quickly find enough peers that can share such content. This would be of particular concern when peers have limited up-link bandwidth or are unreliable. Thus, peer-to-peer content dissemination does not fully meet our goal of disseminating information to a vast number of groups with persistent membership and diverse, dynamic activity levels.

Points of departure: Our design of MAD seeks improvements over these traditional approaches. In contrast to IP multicast-style approaches, we wish to minimize the amount of control overhead associated with keeping state up over a long time, especially when groups are inactive. However, for active groups, we wish to take advantage of the structures that IP multicast designs have adopted. Thus, MAD seeks *the best of both worlds* — forwarding efficiently (a la IP multicast) when information is frequently generated, while also enabling the membership of a group to scale to large numbers where the membership may be long-lived.

III. MAD Overview

MAD environment: Publish-subscribe multicast services that MAD is targeting must support user profile management, authentication and fine-grained access control, etc. In this paper, to focus on the routing aspect of the problem (which is challenging in itself), we explicitly decouple user management from the multicast service. The MAD multicast service overlay consists of *logical* overlay routers, which reside on (or are *owned by* a provider of) the *physical* overlay routers. User management functionality is concentrated in the external *subscription manager*, which maintains subscriptions for all users connecting at a given MAD site, and initiates or cancels group subscriptions with corresponding logical overlay routers on behalf of the end users. Thus, every logical overlay router serves a single aggregated local subscriber representing all users assigned to it by the subscription manager. From the perspective of a MAD overlay router, no knowledge of end users is required and the only entities an overlay router communicates with are other MAD overlay routers and its single aggregated local subscriber.

MAD example: To illustrate the key ideas behind MAD, we consider an example in Figure 3(a), which shows a set of routers that are part of a traditional IP multicast tree. This example is for a single multicast group, with a set of 5 users subscribed to the group. With IP multicast-style approaches (*e.g.*, PIM-SM [14], CBT [2]), every intermediate router on the path from the root (**A**) to the first-hop routers that users are connected to has to maintain state for this group. In this example, there are 11 routers that have to maintain state.

MAD's membership tree (MT) protocol significantly reduces state by limiting the number of routers that have to keep multicast group state. As shown in Figure 3(b), the membership tree consists of only four nodes: **A**, **B**, **C** and **E**. The membership tree itself is shown in Figure 3(c). The core (*i.e.*, root) of the membership tree is selected to be **A**, based on the hash of the group ID. To limit its depth, the membership tree is constructed on top of a *base tree* rooted at the core. A base tree is a balanced *k*-ary tree that comprises all logical overlay routers. For each node, MAD defines a single base tree with this node as its root (see §IV-A). All groups that take this node as the core then use this base tree as the basis to construct their membership trees.

The membership tree is constructed as follows. When a subscriber wants to join a group, it issues a join message, which gets forwarded towards the core along the base tree rooted at the core until it reaches the first node already on the membership tree. An *en route* logical overlay router joins the membership tree whenever the subtree rooted at this node in the base tree has at least a minimum number (**2** in this example) of first-hop routers with attached subscribers.

Each node on the membership tree keeps the following state: (i) *mtChild* – children of this node in the base tree that belong to the MT (encoded as a bit-vector), and (ii) *mtFHs* – a list of first-hop (FH) routers with attached subscribers that are downstream of this node in the base tree.

In our example, when the first subscriber S_1 (attached to **M**) joins, its join message is propagated to the core **A** and the core adds **M** to its *mtFHs*. Subsequently, when another subscriber S_2 (attached to **O**) joins, **A** sees that the subtree rooted at **C** in the base tree now has at least **2** FH routers with subscribers (*i.e.*, **M** and **O**). So **A** informs **C** to create membership state. **C** then updates its *mtFHs* to include **M** and **O**. Meanwhile, **A** sets a bit in its *mtChild* to indicate subscribers downstream of child **C**. After subscribers S_3 (attached to **J**) and S_4 (attached to **K**) join, routers **B** and **E** create membership state, respectively. Finally, after subscriber S_5 (attached to **I**) joins, only 4 routers (**A**, **B**, **C**, and **E**) maintain membership state. Their final state is shown in Figure 3(d).

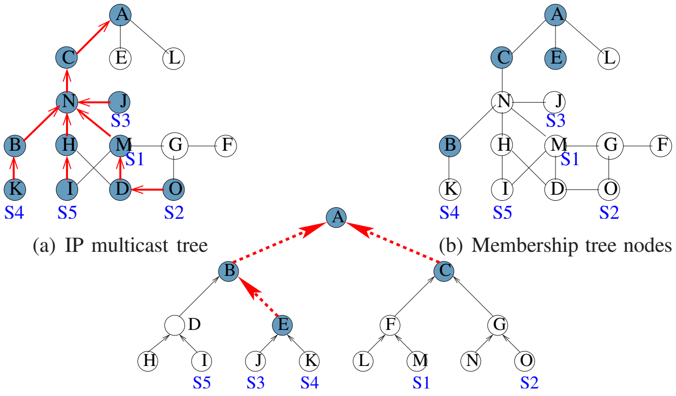
Thus, even this limited topology shows that we have fewer (only 4) routers maintaining membership state compared to IP multicast-style approaches that have more (11) such routers.

On the other hand, IP multicast-style approaches have better forwarding efficiency. Specifically, messages delivered to a group can be forwarded either using the IP multicast tree in Figure 3(a) or along the membership tree in Figure 3(c) (following *mtChild* and *mtFHs*). The former is clearly more efficient than the latter. To maximize overall efficiency, MAD uses IP multicast-style dissemination tree to deliver messages for active groups, and membership tree based forwarding to deliver messages for inactive groups. MAD also provides seamless transition mechanisms to switch between dissemination tree and membership tree as the level of group activity varies over time. Although we have chosen to classify a group as being active based on the "absolute" activity level of individual groups, MAD may choose to classify groups as being active based on the relative activity level of groups, so that the top k% of the groups are classified as being active (to avoid particular groups unnecessarily generating traffic in an attempt to receive better forwarding efficiency).

IV. MAD Protocol Design

The MAD protocol is designed to be modular. It consists of five components: (i) the membership tree sub-protocol, (ii) the dissemination tree sub-protocol, (iii) state transition mechanisms, (iv) failure recovery, and (v) mechanisms for operating across domain boundaries.

Preliminaries: We first introduce some notations before presenting the details of MAD protocols. The MAD multicast service overlay consists of L *logical* overlay routers; assume $L = 2^b$ is a power of two. Each logical overlay router is uniquely identified by a b-bit router ID (ranging from 0 to $L-1$). We use $FH(s)$ to denote the first-hop router that a subscriber s is connected to.



(a) IP multicast tree

(b) Membership tree nodes

(c) Membership tree and its base tree

Nodes on MT	Membership tree state	
	mtChild (children in MT)	*mtFHs* (FHs with subscribers)
A	{**B**, **C**} (encoded as **11**)	{}
B	{**E**} (encoded as **01**)	{**I**}
C	{} (encoded as **00**)	{**M**, **O**}
E	{} (encoded as **00**)	{**J**, **K**}

(d) Final membership tree state

Fig. 3. Examples of MAD trees

Each multicast group g is identified by a unique 128-bit group ID $gid(g)$. Each group g maintains a membership tree $MT(g)$ to record its set of members. If g is active, it also maintains a separate dissemination tree $DT(g)$. $MT(g)$ and $DT(g)$ share a common core (*i.e.*, root) logical overlay router $core(g)$. To balance the load and reduce traffic concentration, we apply a hash function $H(\cdot)$ to map the 128-bit group ID to a random core ID, *i.e.*, $core(g) = H(gid(g))$. A benefit of this hash-based scheme is that it obviates the need for a separate resolution procedure for mapping group ID's to core ID's, which can become expensive with hundreds of billions of groups. Finally, for each logical overlay router ℓ, MAD defines a virtual base tree $BT(\ell)$ rooted at ℓ that includes all L logical overlay routers. $BT(\ell)$ is the basis for constructing $MT(g)$ for every group g with $core(g) = \ell$.

A. Membership Tree Sub-protocol

Given a multicast group g, MAD's membership tree construction protocol is designed to ensure that (i) the membership tree $MT(g)$ only comprises a small number of on-tree nodes that maintain group state (so that the total state requirement is reduced), and (ii) $MT(g)$ stays largely unmodified when there are changes in network performance or overlay unicast routes (so that the control overhead is minimized). To achieve these two design goals, we first statically construct a base tree $BT(\ell)$ for every logical overlay router ℓ, which is a balanced k-ary tree (with each internal node having k children) that is rooted at ℓ and includes all L logical overlay routers. We then construct $MT(g)$ from $BT(core(g))$ by including only those logical overlay routers that have a sufficient number ($S_{\min}$) of downstream first-hop routers with attached subscribers to g.

Abstractly, this is like the formation of a static structure with overlay peer-to-peer networks, such as with Chord [24]. The primary difference here is the maintenance of a static k-

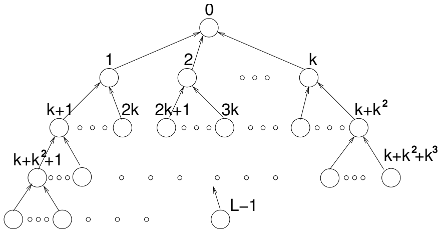


Fig. 4. Base tree $BT(0)$. $BT(\ell)$ is constructed from $BT(0)$ by XORing ℓ with each logical router ID in $BT(0)$.

ary tree that enables rapid computation of a node's parent and children, and avoids maintaining state for the base tree.

Base tree construction: At each logical overlay router ℓ, we construct a balanced k-ary base tree $BT(\ell)$ as follows.

- We first construct $BT(0)$ by sequentially placing logical overlay routers $0, \cdots, L-1$ onto a regular k-ary tree (as shown in Figure 4).
- We then construct $BT(\ell)$ from $BT(0)$ by substituting each logical overlay router r in $BT(0)$ with logical overlay router $r' = \ell \oplus r$, where $\oplus$ is bitwise exclusive or (XOR). For example, the root of $BT(\ell)$ is $\ell \oplus 0 = \ell$, and the set of depth-1 nodes in $BT(\ell)$ are $\ell \oplus 1$, $\ell \oplus 2$, $\ldots$, $\ell \oplus k$.

With the above $BT(\ell)$, for any logical overlay router r, we can compute its parent and children in $BT(\ell)$ as a function of ℓ without requiring any node to maintain any state for $BT(\ell)$. Specifically, we have (i) the parent of r in $BT(0)$ is $\lceil r/k \rceil - 1$, and (ii) the children of r in $BT(0)$ are $rk+1, rk+2, \cdots, rk+k$. To obtain r's parent and children in $BT(\ell)$, we just need to first compute the parent and children of $r' = \ell \oplus r$ in $BT(0)$ and then XOR ℓ with the resulting router IDs.

Membership tree state: Each logical overlay router r on $MT(g)$ keeps the following membership tree state for group g. (i) ***r.g.mtChild***: a k-bit bit-vector, with one bit for each child of r in $BT(core(g))$. The bit is set to 1 when that child is also on $MT(g)$. (ii) ***r.g.mtFHs***: a list of first-hop routers (with attached subscribers) that are downstream of r in the base tree $BT(core(g))$. Our membership tree construction ensures that for any first-hop router $FH(s)$ (with an attached subscriber s), $FH(s)$ appears in exactly one node on $MT(g)$, which is the first node that belongs to $MT(g)$ and lies on the leaf-to-root path from $FH(s)$ to $core(g)$ in $BT(core(g))$.

Joining a membership tree: Messages of type *MtJoinMsg* are sent by subscribers to join a membership tree. When a subscriber s wants to join group g, it sends a *MtJoinMsg* to its first-hop logical overlay router $FH(s)$. $FH(s)$ determines the core for the group as a hash of the group ID $gid(g)$, and then forwards the *MtJoinMsg* towards $core(g)$ along the base tree $BT(core(g))$ (with $FH(s)$ in the message header).

Meanwhile, the decision on node creation is made in a top-down fashion. Unlike protocols like CBT [2] or PIM-SM [14], when a logical overlay router r not yet on $MT(g)$ receives a *MtJoinMsg* for group g, r does not instantiate any state right away. Instead, r simply forwards the *MtJoinMsg* towards $core(g)$ along $BT(core(g))$. Eventually, this message will reach a node that is already on $MT(g)$, denoted by p. Note that it is possible to have $core(g) = p$. Suppose the *MtJoinMsg* received by p comes from p's child n in $BT(core(g))$. Upon receiving this *MtJoinMsg*, p first adds $FH(s)$ to its first-hop router list *p.g.mtFHs*. p then checks to see if it has accumulated $S_{\min}$ first-hop routers (with attached subscribers) from child n. If so, p sends a *MtNodeCreateMsg* to inform n to create membership tree state for group g. The *MtNodeCreateMsg* also includes the list of $S_{\min}$ first-hop routers that are downstream of n in $BT(core(g))$. After n creates the membership tree state, p then removes these $S_{\min}$ first-hop routers from *p.g.mtFHs*, and sets the bit corresponding to n in the bit-vector *p.g.mtChild* to 1, indicating that n is now on $MT(g)$. Note that n may find that all these $S_{\min}$ first-hop routers are downstream of one of its k children in $BT(core(g))$. In this case, n would further inform this child to create state for g and those first-hop routers by sending a *MtNodeCreateMsg*.

Leaving a membership tree: Messages of type *MtLeaveMsg* are used to leave a membership tree. The first-hop router $FH(s)$ for a subscriber s sends a *MtLeaveMsg* towards $core(g)$ along the base tree $BT(core(g))$ until it reaches the first node n that is on $MT(g)$. Upon receiving this *MtLeaveMsg*, n first removes $FH(s)$ from *n.g.mtFHs*. n then checks if *n.g.mtChild* is empty and *n.g.mtFHs* contains fewer than $S_{\min}$ first-hop routers. If so, n determines that it should no longer stay as a node on $MT(g)$. n then sends a *MtNodeDeleteMsg* to its parent in $MT(g)$ with the list of first-hop routers in *n.g.mtFHs* (which will be maintained by this parent henceforth) before n purges the state associated with group g. Note that the parent of n may also decide to delete itself from $MT(g)$ and sends a *MtNodeDeleteMsg* to his own parent (*i.e.*, the grand-parent of n in $MT(g)$).

Membership tree based forwarding: The membership tree can be combined with overlay unicast to deliver both control and data messages as follows.

- When a node wishes to multicast a message M to group g, it first forwards M to $core(g)$ through overlay unicast. This allows us to implement critical functionality such as per group access control, authentication, authorization, accounting and traffic monitoring at the core. Existing protocols such as Scribe [6] take a similar approach.
- After receiving M, $core(g)$ first forwards M to any child in $BT(core(g))$ that belongs to $MT(g)$ (according to bit-vector *core(g).g.mtChild*) through overlay unicast. In addition, $core(g)$ forwards M to all first-hop routers in *core(g).g.mtFHs*. Note that $core(g)$ itself may appear in *core(g).g.mtFHs* when it has attached subscribers. In this case, $core(g)$ also forwards M to its subscription manager through underlay unicast, which then forwards M to the appropriate subscribers that it manages.
- After a child n receives M, it repeats the same procedure above to forward M to its first-hop routers, children and subscription manager (if any).

Avoiding redundant overlay unicast messages: In MAD, overlay unicast is used by each node to forward messages to its children and its list of first-hop routers. To avoid repeated forwarding of a message to the next hop overlay router if this next hop is shared among several overlay unicast routes, MAD supports the option to trade processing overhead for better bandwidth efficiency by packing these "redundant" messages into one message with a variable-length header to specify all the recipients (as in Xcast [4]). Note that if Xcast is available in the underlay, it can directly replace overlay unicast in MAD to further improve forwarding efficiency.

B. Dissemination Tree Sub-protocol

MAD uses a separate dissemination tree for each active group to achieve better forwarding efficiency. The dissemination tree can be maintained using any existing multicast protocol. Our current instantiation of MAD uses the standard Core Based Tree (CBT) [2] protocol for constructing the dissemination tree. For convenience, the dissemination tree and the membership tree share a common core. Since the overlay topology is under our control, we enforce the constraint that each overlay router has no more than 31 neighbors. We use a hash table indexed by the 128-bit group ID is used to store all the dissemination tree state for different groups. The current group status (*e.g.*, whether it is active or inactive) is decided by the group's core in the membership tree.

C. Mode Transition

A major challenge in the design of MAD is how to ensure the smooth transition between membership tree based forwarding (*i.e.*, the inactive mode) and dissemination tree based forwarding (*i.e.*, the active mode). In particular, it is essential to avoid disruption of the multicast data delivery service during mode transition.

Our basic strategy for achieving smooth mode transition is to require every group in the system to always maintain the membership tree even when the group is considered active and the dissemination tree has been constructed. Having an always up-to-date membership tree ensures that during the transition period we can use membership tree based forwarding to deliver messages reliably to all the group members, with very little additional control overhead.

When a group transitions from being active to inactive, the transition is achieved simply by not forwarding on the dissemination tree and "tearing it down". The key however is to have an efficient transition from inactive to active mode while ensuring no data is lost. Every new group g initially stays in the *inactive* mode. As group activity becomes high enough, $core(g)$ may decide to improve forwarding efficiency by creating a separate dissemination tree $DT(g)$. We first deliver data messages over both $MT(g)$ and $DT(g)$ during the transition (*i.e.*, in *transient* mode) and stop delivering data to a subtree of $MT(g)$ only after the root of the subtree is certain that all *existing* group members in the subtree are able to receive data from the dissemination tree.

While we have worked through the details of mode transition in the prototype implementation we have evaluated in this paper, we elide the details here due to limited space.

D. Failure Recovery

MAD handles failures through replication, in a manner similar to other overlay based approaches [6], [20]. We exploit the advantage of being able to establish connectivity in the overlay dynamically, in response to a failure. The main concern we address here is the careful management of state specific to MAD. Specifically, for each physical overlay router in our system, we designate a set of physical overlay routers as its *shadow* routers. To minimize replicated state, we only store state related to leaf nodes of the membership tree. Once the membership tree is repaired, it can perform normal multicast data delivery. For each logical overlay router ℓ owned by a physical overlay router p, the shadow routers of p only need to replicate the user subscription state (*i.e.*, *mtFHs*) for all enrolled groups. The replicated state is saved in the stable storage (*e.g.*, hard disk drive) of all the shadow routers of p. There is a keep-alive message exchange between a physical overlay router p and its shadow routers p' for fast failure detection and recovery. For each logical overlay router ℓ that is previously owned by p, p' needs to recover the role of ℓ in every group that ℓ is involved in. Each child ℓ of p may discover p' through a directory service lookup that maintains an up-to-date list of shadow routers. Such a directory service may also be implemented in a distributed manner using DHTs.

Leaf: ℓ may be a leaf node in the membership tree $MT(g)$ for group g. To take over such a role, p' sends *MtJoinMsg* towards the core of group g along base tree $BT(g)$. We save bandwidth by aggregating multiple *MtJoinMsg* with the same core into a single message with a list of group IDs.

On-tree node: ℓ may be an internal node of either a membership tree or a dissemination tree. MAD takes advantage of on-tree nodes in both *MT* and *DT* sending heartbeat messages down the tree. Upon failure detection, the child repairs the tree by sending a join message (*i.e.*, either *MtJoinMsg* or *DtJoinMsg*) to its parent.

Core: ℓ may be the core for a group. After a failure, p' starts receiving join messages from children of ℓ in group g. p' infers the *mode* information from the received *DtJoinMsg*.

It is important to note that MAD only involves system-wide keep-alive messages (between each physical overlay router and its shadow routers) as opposed to per-group keep-alive messages. So the control overhead due to keep-alive messages is independent of the number of multicast groups. In contrast, IP multicast style protocols like CBT require per-group keep-alive messages to retain forwarding state. Our evaluation in §V suggests that such per-group control overhead can become quite expensive when there are a large number (*e.g.*, billions) of groups and a large number of them are inactive.

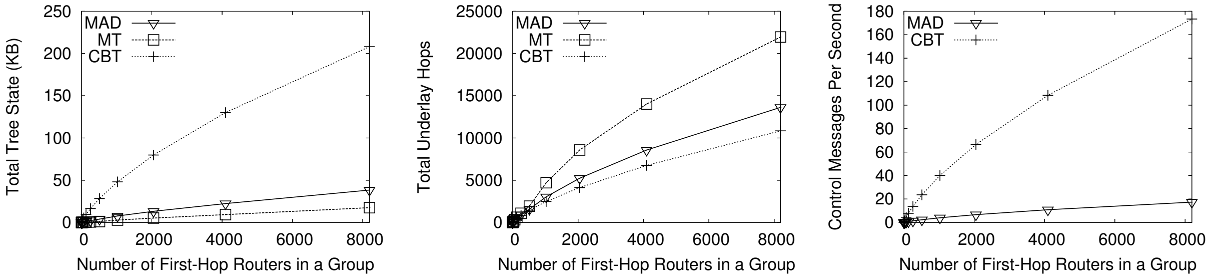
E. MAD across Administrative Domains

MAD groups a set of routers in the same region or network domain (*e.g.*, university network, corporate network, and AS) to form a "MAD domain". MAD domains serve two goals: (i) enable MAD to operate across multiple administrative domains, and (ii) handle heterogeneity and load imbalance by promoting autonomous decisions in local networks. For more information, please read our detailed version [10].

V. Scaling of MAD Trees

In this section, we conduct extensive simulations on realistic network topologies to examine the state requirement of MAD trees and the tradeoff between state reduction and forwarding cost. To gain more insights into the scaling of MAD trees, we also analytically derive the state requirement.

In our model, rather than the number of subscribers, the number of distinct subscription managers that are involved in a group (essentially the number of FH routers) reflects the scaling of the system. Thus, our results examine the scaling based on the number of FH routers in the group or system.



(a) Total tree state for a group
(b) Total underlay hops
(c) Control overhead

Fig. 5. Scaling of MAD trees on topology *pow-16k*.

A. Simulation Evaluation

Simulator: To evaluate MAD, we developed a simulator (with 6,000 lines of C/C++ code) that achieves scalability by avoiding the simulation of packet-level events.

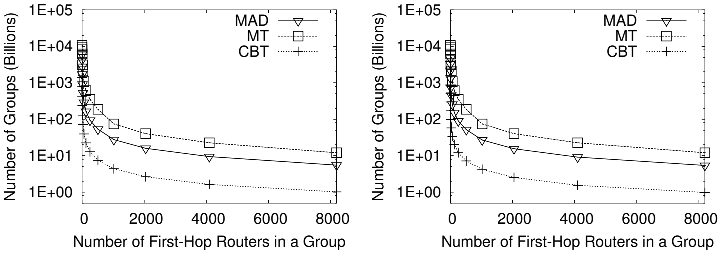
Topologies: Our simulator constructs overlay network topologies as follows: (i) generate an underlay network topology that comprises 16,000 routers in all; (ii) use shortest hop-count routing to obtain the underlay distance (*i.e.*, hop count) between each pair of routers; and (iii) construct the overlay network topology as the union of m edge-disjoint Minimum Spanning Trees (MSTs) for the logical full-mesh (*i.e.*, clique) over the 16,000 routers. The underlay topology we consider is either a power-law topology (*pow-16k*), or a transit-stub topology with stub (access) nodes and transit nodes (*ts-16k*).

Simulation setup: We randomly form 100 multicast groups each with a fixed group size. We then vary this fixed multicast group size with respect to the number of first-hop routers. For each group, we compute the required state, forwarding cost, and control overhead of CBT and MT as follows.

- We compute the total tree state stored by all on-tree nodes. For CBT, an on-tree node stores a 128-bit group ID and a 32-bit bit-vector *dtChild* (which specifies all interfaces with members downstream). For MT, an on-tree node stores a 128-bit group ID, a 16-bit bit-vector *mtChild* (which indicates whether each child in the base tree belongs to MT), and a list of first-hop overlay routers in the subtree rooted at this node (*mtFHs*).
- The forwarding cost for a message is measured by the total number of underlay hops that the message traverses.
- The control overhead is measured by the total number of keep-alive (*i.e.*, hello) messages sent by the group in a second. We use the default keep-alive message interval of 60 seconds in CBT [2], *i.e.*, each node in a CBT sends a keep-alive message to its parent once every 60 seconds.

Finally, to compute the state requirement, forwarding cost and control overhead of the MAD protocol, we assume that 10% groups are active, and that they contribute to 75% of the data traffic. These fractions are chosen based on the publishing behavior of RSS feeds as shown in Figure 2.

Simulation results: Figure 5 compares the state requirement, forwarding cost, and control overhead for *pow-16k*, where every data point is the average over 100 random groups (of the same size). The results for *ts-16k* are quantitatively similar and are omitted in the interest of brevity. Figure 5(a) compares the total tree state required by CBT, MT, and MAD. By combining MT with CBT, MAD achieves nearly an order of magnitude state reduction over CBT. Figure 5(b) shows the forwarding cost of CBT, MT, and MAD. The total forwarding cost for MAD is very close to CBT (both in delay and number of hops traversed) and significantly outperforms MT as the size of the group increases. Figure 5(c) shows the control overhead of CBT and MAD (measured by the number of keep-alive messages per second from each group). MAD achieves an order of magnitude reduction in control overhead over CBT, because 90% groups are inactive and only have to maintain the MT, which requires no per-group keep-alive messages.



(a) pow-16k
(b) ts-16k

Fig. 6. Max # of groups that 2^{16} routers (each with 3GB MEM) can hold.

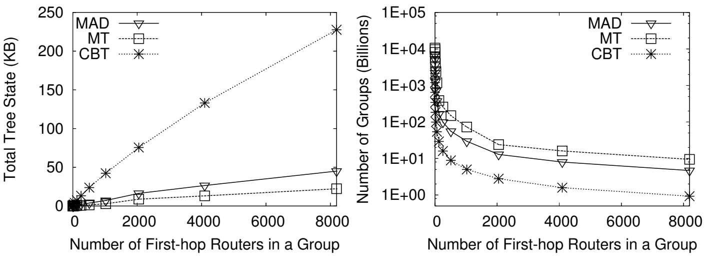
Figure 6 shows the maximum number of groups that an overlay with 2^{16} overlay routers (each with 3GB memory) can hold. MAD can easily support hundreds of billions of groups on today's commercial hardware platform. For example, with a group size of 16, MAD supports 913 billion groups for *pow-16k*, and 750 billion groups for *ts-16k*, yielding a factor of 7–9 improvement over CBT. Note that such improvement is close to optimal, because in our simulation 10% groups are active and maintain both MT and CBT. The state reduction is thus bounded by $\frac{state^{\text{CBT}}}{0.1\times state^{\text{CBT}}+state^{\text{MT}}} \leq 10$.

B. Formal Analysis of State Requirement

Consider an overlay with L logical overlay routers. Given a multicast group g that has F randomly selected first-hop routers to which subscribers are connected, below we analytically derive the expected state requirement for both $CBT(g)$ and $MT(g)$ with respect to F.

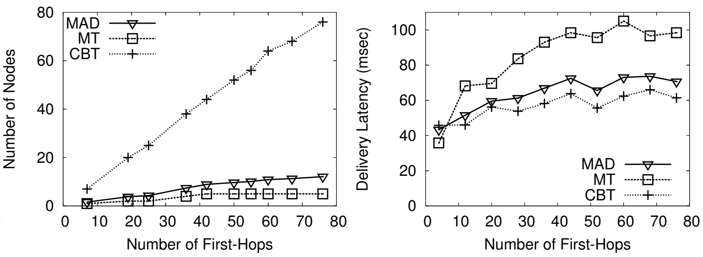
Membership tree state requirement: Since all the base trees constructed in §IV-A are isomorphic, without loss of generality we can assume that $core(g) = 0$. Therefore, $MT(g)$ is constructed based on the base tree $BT(0)$, which is illustrated in Figure 4.

For any logical overlay router i, let N_i be the number of nodes that are in the subtree of $BT(0)$ rooted at i (including node i itself); and let random variable X_i denote the number of first-hop routers with subscribers that are in the same subtree.



(a) Total tree state for a group (b) Max # of groups that 2^{16} routers can hold

Fig. 7. Analytical results on the scaling of MAD trees.



(a) No. of nodes keeping tree state (b) Avg. message delivery latency

Fig. 8. Efficiency of MAD

Assuming that the F first-hop routers with attached subscribers are selected uniformly at random from all L logical overlay routers, then X_i has a hypergeometric distribution: $\Pr(X_i = n) = \frac{\binom{F}{n}\binom{L-F}{N_i-n}}{\binom{M}{N_i}}$. The mean μ_i and variance σ_i^2 of X_i are given by $\mu_i = N_i\frac{F}{L}$ and $\sigma_i^2 = \frac{N_i(L-N_i)}{L-1}\frac{F}{L}\frac{L-F}{L}$.

In order for logical overlay router i to become a member of $MT(g)$, we need to have $X_i \geq S_{\min}$. The probability for this to occur is bounded by Chebyshev's Inequality:

$$\Pr(X_i \geq S_{\min}) \leq \left[\frac{\sigma_i}{\max(\sigma_i, S_{\min}-\mu_i)}\right]^2 \triangleq P_i^{\mathrm{MT}}$$

where μ_i and σ_i^2 are the mean and variance of X_i (see above).

The expected number of nodes on $MT(g)$ is thus bounded by $\sum_{i=0}^{L-1} P_i^{\mathrm{MT}}$. Each node n on $MT(g)$ stores a 128-bit group ID (as the key for the forwarding table), a 16-bit field *mtChild*, and a list of 16-bit first-hop router IDs (*mtFHs*). Since each 16-bit first-hop router ID is stored only once, the total amount of state devoted to *mtFHs* is $16 \times F$ bits. So the expected number of bits for the entire membership tree state is:

$$state^{\mathrm{MT}} \leq 16 \times F + (128+16) \times \sum_{i=0}^{L-1} P_i^{\mathrm{MT}}$$

Dissemination tree state requirement: For simplicity, we only consider the state requirement of $CBT(g)$ in the special case where all the unicast routes destined to $core(g)$ together form a balanced k-ary tree isomorphic to $BT(0)$. This is likely to underestimate the state requirement of CBT in the more general case where the unicast routes do not have such a regular underlying structure. However, our results clearly show that even in this special case, MT and MAD achieve much better state efficiency than CBT.

In the special case we consider, let N_i and X_i denote the same as in the analysis for $MT(g)$ (see above). A logical overlay router i becomes an on-tree node of $CBT(g)$ whenever $X_i > 0$. The probability for this to occur is given by

$$P_i^{\mathrm{CBT}} \triangleq 1 - \Pr(X_i = 0) = 1 - \frac{\binom{F}{0}\binom{L-F}{N_i-0}}{\binom{L}{N_i}} = 1 - \frac{\binom{L-F}{N_i}}{\binom{L}{N_i}}$$

The expected number of nodes on $CBT(g)$ is $\sum_{i=0}^{L-1} P_i^{\mathrm{CBT}}$. Each node n on $CBT(g)$ stores a 128-bit group ID (as the key for the forwarding table) plus a 32-bit bit-vector *n.g.dtChild*. So the expected number of bits for the dissemination tree is:

$$state^{\mathrm{CBT}} = (128+32) \times \sum_{i=0}^{L-1} P_i^{\mathrm{CBT}}$$

Numerical results: We numerically compute the state requirement for MT and CBT (with $k = 16$) for different sizes of the group and the overlay network. As shown in Figure 7, MAD achieves an order of magnitude state reduction over CBT. Moreover, depending on the group size, MAD can easily support hundreds of billions of groups on today's commercial hardware. These results are consistent with our simulation results on more realistic topologies (*i.e.*, Figure 5 and Figure 6) and demonstrate the MAD's ability to reduce group state by decoupling membership and dissemination.

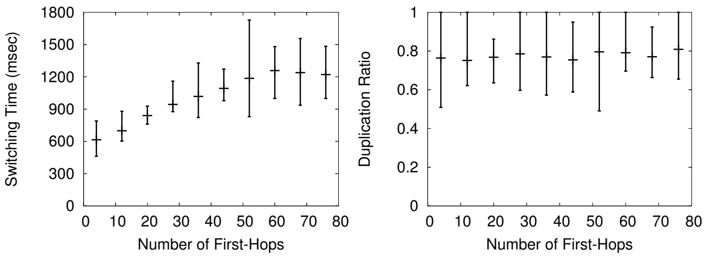
VI. Evaluation of Implementation

We implemented the MAD protocol on top of *FreePastry 1.4.4* [21]. Below we evaluate the state efficiency, forwarding efficiency, and mode transition cost of our MAD prototype.

Experimental setup: We conducted experiments on the Emulab testbed [25]. Our experiments involve 105 Emulab nodes that range from systems with an Intel Pentium III with 256MB RAM to a Xeon with 2GB RAM. We use the *sprintlink-us* network topology available from Rocketfuel [22], with link latencies inferred in [16] and zero link loss. To control the total number of nodes in the network, we vary the number of routers in each city (PoP). Routers in the same city are connected via a LAN with 0 latency. For each group, we select first-hop routers to join the group uniformly at random from the entire set of nodes in the experiment. Similar to our simulation, we assume that 10% groups are active and that they contribute to 75% of the data traffic. Finally, we use the default settings of *FreePastry 1.4.4*.

State efficiency: We first look at the results from experiments to show the benefit of the membership tree (MT) over the dissemination tree (CBT) with respect to the total state stored at on-tree routers. Figure 8(a) shows the number of nodes that maintain state for a group, as a function of the number of subscribed first hop routers. With our design of the membership tree, only a small number of nodes need to maintain state for the group, and this number grows slowly. With CBT, the number of on-tree routers grows much more rapidly as more first-hop routers join the tree. MT thus achieves significant state reduction over CBT. The state efficiency of MAD is close to MT because 90% of all groups are inactive and maintain only the efficient MT.

Forwarding efficiency: Figure 8(b) shows the average latency of delivering a data message from the core to all the member routers in a tree. CBT achieves much lower message delivery latency than MT. This is not surprising because MT is designed primarily for state efficiency, not for forwarding efficiency. Meanwhile, the delivery latency of MAD is very close to that of CBT, because most of the data traffic comes from active groups and is thus delivered via CBT in the MAD



(a) Transition latency

(b) Message duplication ratio

Fig. 9. Cost of mode transition from *MT* to *CBT*

protocol — recall that we assume active groups contribute to 75% of all the data traffic.

Mode transition cost: We measure the cost of mode transition from *MT* to *CBT* (as an inactive group becomes active) in terms of (i) the transition latency (*i.e.*, the time it takes for all nodes in a group to complete mode transition) and (ii) the message duplication ratio (*i.e.*, the ratio between the number of duplicate messages and the total number of distinct messages received during the transition period). Recall that mode transition from *CBT* to *MT* is almost instantaneous and does not result in any duplicated messages (see §IV-C).

Figure 9(a) shows the average, minimum, and maximum transition latency from *MT* to *CBT* as the number of first-hop routers in the group increases. The transition latency is quite low — the average transition latency is close to 1 second and the worst-case transition latency is only 1.8 seconds.

Figure 9(b) shows the average, minimum, and maximum message duplication ratio during mode transition. The message duplication ratio is less than 1 because as soon as a first-hop router receives the first message from the *CBT*, it informs its parent in the *MT* to suppress all future duplicate messages. Therefore, during the entire transition period, only those messages that are sent before the suppression occurs will be received in duplicate. Since the transition latency is low, the overhead due to message duplication is acceptable (especially given the increased forwarding efficiency after the mode transition completes).

Summary: Our experimental results clearly demonstrate that MAD achieves both the high state efficiency of *MT* and the high forwarding efficiency of *CBT*. Meanwhile, such a benefit comes at a low cost — the mode transition between *MT* and *CBT* only takes 1–2 seconds and the overhead due to message duplication during the transition is acceptable.

VII. Conclusions

In this paper, we present Multicast with Adaptive Dual-state (MAD), a novel architecture for providing efficient multicast service at massive scale. The key to its scalability and efficiency is the decoupling of group membership and forwarding state, which allows us to optimize for different objectives for active and inactive groups. Group membership is maintained scalably in a distributed fashion using a hierarchical membership tree (MT). Inactive groups forward data over their membership trees. Active groups use IP multicast-style dissemination trees (DT) for efficient data forwarding. MAD provides seamless and efficient mechanisms for a group to transition between DT-based and MT-based forwarding as its activity level changes.

We examined the scaling characteristics of the MAD protocol through analysis, simulation and a prototype implementation. Compared with IP-multicast style approaches (*e.g.*, CBT), MAD achieves nearly an order of magnitude reduction in state requirement and control overhead. As a result, MAD can support hundreds of billions of multicast groups with long-lived membership on today's commercial hardware platform. Meanwhile, MAD achieves comparable forwarding efficiency and low message delivery latency, through the use of DT-based forwarding for active groups. Thus, MAD achieves the best of both worlds – scalability and persistence in group membership by using MT, and efficient data forwarding by using DT.

References

[1] AT&T's fast IPTV growth may have a downside. Multichannel News. http://www.multichannel.com/article/ca6493387.html.

[2] A. J. Ballardie. Core based trees (CBT version 2) multicast routing: Protocol specification. RFC-2189, 1997.

[3] BitTorrent. http://www.bittorrent.com.

[4] R. Boivie, N. Feldman, Y. Imai, W. Livens, D. Ooms, and O. Paridaens. Explicit multicast (Xcast) basic specification. IETF draft, 2000.

[5] A. Bozdog, R. van Renesse, and D. Dumitriu. Selectcast: a scalable and self-repairing multicast overlay routing facility. In *Proc. ACM workshop on survivable and self-regenerative systems*, 2003.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE J. on Selected Areas in Comm.*, 2002.

[7] E. Castronova. Network technology, markets and the growth of synthetic worlds. In *Proc. NetGames*, May 2003.

[8] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I tube, you tube, everybody tubes: Analyzing the world's largest user generated content video system. In *Proc. IMC*, 2007.

[9] J. Cho, and S. Roy. Impact of search engines on page popularity. In *Proc. WWW*, 2004.

[10] T. Cho, M. Rabinovich, K.K. Ramakrishnan, D. Srivastava, and Y. Zhang. Enabling Content Dissemination Using Efficient and Scalable Multicast (extended version). http://www.cs.utexas.edu/~khatz/papers/madev.pdf

[11] C. P. Costa, I. S. Cunha, A. Borges, C. V. Ramos, M. M. Rocha, J. M. Almeida, and B. Ribeiro-Neto. Analyzing client interactivity in streaming media. In *Proc. WWW*, 2004.

[12] L. HMK Costa, S. Fdida, and O. CMB Duarte. Hop-by-hop multicast routing protocol. In *Proc. SIGCOMM*, 2001.

[13] J. Cui, M. Geria, K. Boussetta, M. Faloutsos, A. Fei, J. Kim, and D. Maggiorini. Aggregated multicast: A scheme to reduce multicast states. IETF draft, 2002.

[14] B. Fenner, M. Handley, H. Holbrook, and I. Kouvelas Protocol Independent Multicast-Sparse Mode (PIM-SM): Protocol Specification. RFC4601, 2006.

[15] H. Liu, V. Ramasubramanian, and E. G. Sirer. A measurement study of RSS, a publish-subscribe system for Web micronews. In *IMC*, 2005.

[16] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson. Inferring link weights using end-to-end measurements. In *Proc. IMW*, 2002.

[17] JUNOS 9.2 Multicast Protocols Configuration Guide. http://netscreen.com/techpubs/software/junos/junos92/swconfig-multicast/.

[18] M. Medved. EBay auction counts. 2008. http://www.medved.net/cgi-bin/cal.exe?EIND

[19] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. In *Proc. SIGCOMM*, 2006.

[20] A. Rowstron and P. Druschel. Store management and caching in PAST, a large-scale, persistent peer-to-peer storage utility. In *Proc. SOSP*, 2001.

[21] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *Proc. IFIP/ACM Middleware*, 2001.

[22] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring ISP topologies with Rocketfuel. *IEEE/ACM Trans. Netw.*, 2004.

[23] I. Stoica, T. S. E. Ng, and H. Zhang. REUNITE: A recursive unicast approach to multicast. In *Proc. INFOCOM*, 2000.

[24] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup protocol for Internet applications. In *Proc. SIGCOMM*, 2001.

[25] B. White, J. Lepreau, L. Stoller, R. Ricci, S. Guruprasad, M. Newbold, M. Hibler, C. Barb, and A. Joglekar. An integrated experimental environment for distributed systems and networks. In *Proc. OSDI*, 2002.

[26] YouTube. http://www.youtube.com