

SAAR: A Shared Control Plane for Overlay Multicast

Animesh Nandi^{‡◊} Aditya Ganjam[†] Peter Druschel[◊] T. S. Eugene Ng[‡]
Ion Stoica[§] Hui Zhang[†] Bobby Bhattacharjee^Φ

[◊]Max Planck Institute for Software Systems [‡]Rice University

[†]Carnegie Mellon University

[§]University of California, Berkeley

^ΦUniversity of Maryland

Technical Report 2006-2, Max Planck Institute For Software Systems, Updated on Feb 2007

Abstract

Many cooperative overlay multicast systems of diverse designs have been implemented and deployed. In this paper, we explore a new architecture for overlay multicast: we factor out the control plane into a separate overlay that provides a single primitive: a configurable anycast for peer selection. This separation of control and data overlays has several advantages. Data overlays can be optimized for efficient content delivery, while the control overlay can be optimized for flexible and efficient peer selection. Several data channels can share a control plane for fast switching among content channels, which is particularly important for IPTV. And, the control overlay can be reused in multicast systems with different data plane organizations.

We designed and evaluated a decentralized control overlay for endsystem multicast. The overlay proactively aggregates system state and implements a powerful anycast primitive for peer selection. We demonstrate that SAAR's efficiency in locating peers reduces channel switching time, improves the quality of content delivery, and reduces overhead, even under dynamic conditions and at scale. An experimental evaluation demonstrates that the system can efficiently support single-tree, multi-tree and block-based multicast systems.

1 Introduction

Cooperative endsystem multicast (CEM) has become an important paradigm for content distribution in the Internet [11, 35, 8, 2, 29, 46, 27, 7, 49]. In CEM, participating endsystems form an overlay network and cooperatively disseminate content. As a result, the resource cost of disseminating content is shared among the participants. Unlike server-based approaches, the content source need not provide bandwidth and server resources proportional to the number of receivers; unlike IP multicast [14], no network layer support is required; and unlike commercial content-distribution networks [1], no contract with a provider is needed.

Numerous CEM systems are being proposed and deployed by research, industry and open source communities. The systems cover a range of designs, using single tree-, multi-tree- or mesh-based data dissemination approaches and various overlay maintenance algorithms.

Common to all CEM systems is the problem of selecting overlay neighbors for data dissemination. We will show that the policies and mechanism used to make this selection significantly affect the performance of CEM systems. An ideal peer selection mechanism can support sophisticated selection policies, enabling high-quality data paths and good load balance, while accommodating participants with heterogeneous capabilities. An efficient peer selection mechanism can scale to large groups and large numbers of groups. Lastly, a responsive mechanism allows the system to more rapidly respond to failures and node departures, and it allows nodes to quickly join and switch content channels. Fast channel switching, in particular, is critical to emerging IPTV applications [23].

We show that peer selection for CEM systems can be performed using an *anycast* primitive, which takes as arguments a constraint and an objective function. Among the participating nodes that satisfy the constraint, the primitive selects one that maximizes the objective function. Such an anycast primitive offers a single, unified mechanism for implementing diverse data plane policies.

Consider the following example of a simple data dissemination tree. Each node n needs to select a parent. The constraint would require that a prospective parent is not a descendant of n and has spare forwarding capacity. Among the eligible nodes, the objective function might minimize the loss rate or the distance of the parent from the root. Much more complex data plane structures can be expressed in this way, e.g., multiple interior-node-disjoint trees as in SplitStream [7].

We have designed and evaluated SAAR, a *control overlay* for CEM systems. SAAR provides a powerful anycast primitive for selecting peers in one or more separate data dissemination overlays. SAAR provides several key benefits:

- *SAAR separates control and data dissemination into different overlay networks.* As such, it avoids a tradeoff between data and control efficiency. We show that the benefits of this separation outweigh the costs of maintaining a separate control overlay. First, the SAAR control overlay is optimized for efficient peer selection. When compared to current CEM systems, SAAR can locate more appropriate peers, and can do so faster. Rapid peer selection results in faster channel join and switching times; more appropriate peer selection improves data paths and delivery quality. Second, the data overlay is not constrained by a control overlay structure, and can therefore be optimized solely for efficient data dissemination and load balance, subject to application policy.

- *SAAR can support different data plane structures* (e.g. tree, multi-tree, mesh-based). Specific structures can be achieved by defining appropriate constraints and objective functions for anycast. Thus, SAAR separates the common control mechanism from the specific policies for maintaining a data overlay. As a reusable control overlay, SAAR simplifies the design of CEM systems.

- *A single SAAR overlay can be shared among many data overlay instances.* SAAR allows nodes to remain in the control overlay independent of their data channel membership. Control overlay sharing allows nodes to quickly join a channel and to rapidly switch between channels, which is critical for applications like IPTV [23].

The implementation of SAAR is layered on a structured overlay network [38, 8]. For each data overlay instance, a tree is embedded in this structured control overlay that connects the members of that data overlay. State information for members of a data channel is aggregated and disseminated within the corresponding tree. An anycast traverses the tree to locate peers for the data overlay, subject to the constraint and objective function. The aggregated state information is used to guide the search.

The rest of this paper is organized as follows. Section 2 briefly reviews existing CEM systems and other related work. Section 3 presents our proposed architecture and the design of SAAR. Section 4 describes how different data plane organizations can be built using SAAR. Section 5 presents an experimental evaluation of our SAAR prototype. We conclude in Section 6.

2 Background and related work

In this section, we consider existing CEM systems and other related work. The data planes of CEM systems can be classified as either path-based (single tree and multiple tree) or block-based. Path-based systems maintain one or more loop free paths from the content source to each member of a group. ESM [10], Overcast [25] and NICE [2] form a single tree, while SplitStream [7] and Chunkyspread [43] form multiple trees. In Bullet [27],

CoolStreaming [49] and Chainsaw [30], the streaming content is divided into fixed-size blocks and group members form a mesh structure. Mesh neighbors exchange block availability information and swap missing blocks. Next, we discuss existing CEM systems from the perspective of the type of overlay network they utilize.

Unstructured overlay CEM systems construct an overlay network that is optimized primarily for data dissemination. Overlay neighbors are chosen to maximize the quality of the content delivery (i.e., minimize packet loss, delay and jitter), to balance the forwarding load among the overlay members, and to accommodate members with different amounts of network resources. Typically, a separate overlay is constructed for each content instance, consisting of the set of nodes currently interested in that content. The control plane is then implemented within the resulting overlay. Although these systems enable efficient data dissemination, the overlay is not optimized for efficient control.

Overcast [25], Host Multicast [48] and End System Multicast (ESM) [10] form a single dissemination tree. In the former two systems, nodes locate a good parent by traversing the tree, starting from the root. These protocols do not scale to large groups, since each member must independently explore the tree to discover a parent, and the root is involved in all membership changes. ESM uses a gossip protocol to distribute membership information among the group members. Each node learns a random sample of the membership and performs further probing to identify a good parent. The protocol is robust to node departures/failure but does not scale to large group sizes, where the membership information available to a given node tends to be increasingly partial and stale.

Chunkyspread [43] uses a multi-tree data plane embedded in an unstructured overlay, using a randomized protocol to select neighbors. The selection considers the heterogeneous bandwidth resources of nodes and assigns them an appropriate node degree in the overlay.

Bullet [27], CoolStreaming [49] and Chainsaw [30] use block-based data dissemination in an unstructured mesh overlay. Bullet has separate control and data planes. The control plane is not shared among multiple data channels and it was not designed to support different data plane organizations.

Structured overlay CEM systems use a structured overlay network [38, 34, 41, 37]. The key-based routing primitive [13] provided by these overlays enables scalable and efficient neighbor discovery.

In general, data is disseminated over existing overlay links. This constraint tends to make it more difficult to optimize data dissemination and to accommodate nodes with different bandwidth resources [3]. Group membership changes, on the other hand, are very efficient and the systems are scalable to very large groups and large numbers of groups in the same overlay.

Scribe [8], Subscriber/Volunteer(SV) trees [15] and SplitStream [7] are examples of CEM systems based on structured overlays. Scribe embeds group spanning trees in the overlay. The trees are then used to anycast or multicast within the group. Due to the overlay structure, some nodes may be required to forward content that is not of interest to them. SV trees are similar to Scribe, but ensure that only interested nodes forward content.

SplitStream uses multiple interior-node-disjoint dissemination trees that each carry a slice of the content. Compared to single-tree systems, it better balances the forwarding load among nodes and reduces the impact of node failures. SplitStream has an anycast primitive to locate parents with spare capacity in the desired trees when none can be found using Scribe. In SplitStream, however, this primitive is used only as a last resort, since it may add non-overlay edges and may sacrifice the interior-node-disjointness of the trees.

NICE [2] is not based on a structured overlay as we defined it. Nevertheless, it shares the properties of efficient control but constrained data dissemination paths. Nodes dynamically organize into a hierarchy, which is then used to distribute the data.

Other related work: Anycast was first proposed in RFC 1546 [31]. GIA [26] is an architecture for scalable, global IP anycast. Both approaches share the drawbacks of network-layer group communication. That is, they require buy-in from a large fraction of Internet service providers to be effective at global scale, and they cannot easily consider application-specific metrics in the server selection. Application-layer anycasting [4] defines anycast as an overlay service.

Anycast within a structured overlay network has been used in several systems for decentralized server selection [9, 24, 40, 17]. Scribe [9] and DOLR [24] deliver anycast requests to nearby group members. Unlike SAAR, they provide only a coarse-grained overload protection mechanism by requiring overloaded group members to leave the group temporarily. *i3* [40] provides fine-grained load balancing of anycast requests among the group members, but is not designed for efficient server selection based on multiple metrics like load, location and server state. Server selection in Oasis [17] is primarily optimized for locality, but also incorporates liveness and load. Oasis does not optimize the anycast based on proactive aggregation of state information. Unlike these systems, SAAR provides general and efficient anycast for peer selection in CEM systems.

Several systems use structured overlays for efficient request redirection [16, 45, 12]. CoDeeN [45], a cooperative CDN, distributes client requests to an appropriate server based on factors like server load, network proximity and cache locality. Coral [16] is a peer-to-peer web-content distribution network that indexes cached web

pages and redirects client requests to nearby peers that have the desired content cached.

SDIMS [47] (influenced by Astrolabe [36]) aggregates information in large scale networked systems and supports queries over the aggregated state of a set of nodes. Internally, SDIMS relies on aggregation trees embedded in a structured overlay to achieve scalability with respect to both the number of nodes and attributes. SAAR implements a subset of SDIMS's functionality, which is specialized for the needs of a CEM control plane.

ChunkCast [12] provides a shared control overlay, in which it embeds index trees for objects stored in the overlay. An anycast primitive discovers a nearby node that holds a desired object. ChunkCast is intended for block dissemination in a swarming file distribution system, and not for streaming multicast. Its anycast primitive is specialized for this purpose, and not for peer selection in a CEM system.

Pietzuch et al. [32] observe that structured overlays do not produce a good candidate node set for service placement in Stream-based overlay networks (SBONs). This is closely related to our observation that structured overlay CEM systems have constrained and sub-optimal data distribution paths.

Opus [5] provides a common platform for hosting multiple overlay-based distributed applications. Its goal is to mediate access to wide-area resources among multiple competing applications, in a manner that satisfies each application's performance and reliability demands. SAAR, on the other hand, provides a control overlay and an anycast peer selection service for a specific application, CEM. Thus, Opus and SAAR address largely complementary problems.

3 Design of SAAR

We begin with an overview of the SAAR control plane and describe its design in detail. Figure 1 depicts the SAAR architecture.

Our architecture for CEM systems separates the control and data planes into distinct overlay networks. There are no constraints on the structure of the data plane: it can be optimized for efficient data dissemination, can accommodate heterogeneity and includes only nodes that are interested in the content. The control overlay can be shared among many data plane instances, each disseminating a different content type or channel.

SAAR uses a decentralized control plane based on a structured overlay network. Its anycast primitive supports efficient and flexible selection of data dissemination peers. The SAAR overlay performs efficient, proactive state dissemination and aggregation. This aggregate state is used to increase the efficiency of the anycast primitive.

All nodes that run a particular CEM system participate in the SAAR control overlay, regardless of which

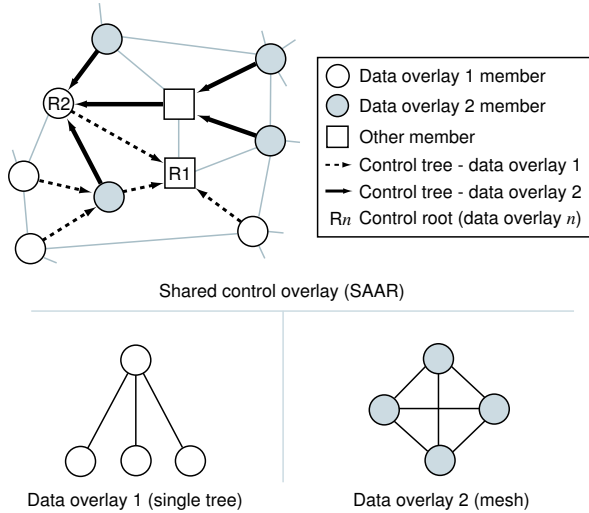


Figure 1: SAAR architecture: Each node is a member of the control overlay and may be part of one or more data overlays. The members of a given data overlay are part of a tree embedded in the control overlay. Nodes use the SAAR anycast primitive to locate data overlay neighbors.

content they are currently receiving. This enables rapid switching between content channels. Even nodes that do not currently receive any content may choose to remain in the control overlay. In this “standby” mode, a node has low overhead and can join a data overlay with very low delay. As a result, membership in the control overlay is expected to be more stable and longer-term than the membership in any data overlay. Additionally, the sharing of state information across data overlays can reduce overhead, e.g., when a node is in more than one data overlay because it receives several content channels.

Group abstraction: The key abstraction provided by SAAR is a *group*. A group represents a set of nodes that are members of one data overlay. The group’s control state is managed via a spanning tree that is embedded in the control overlay and rooted at a random member of the control overlay. Due to the SAAR overlay structure, the spanning tree may contain interior nodes that are not part of the group. The group members may choose to form any data overlay structure for data dissemination.

A set of *state variables* is associated with a group. Each group member holds an instance of each state variable. Typical examples of state variables are a node’s forwarding capacity, current load, streaming loss rate, tree-depth in a single-tree data plane, etc.

SAAR can aggregate state variables in the spanning tree. Each state variable g is associated with an *update propagation frequency* f_{up} , a *downward propagation frequency* f_{down} and an *aggregation operator* A . The values

of a state variable are periodically propagated upwards towards the root of the group spanning tree, with frequency at most f_{up} . (The propagation is suppressed if the value of a variable has not changed). At each interior node, the values received from each child are aggregated using the operator A . The aggregated value at the root of the spanning tree is propagated down the tree with frequency at most f_{down} . State variables for which no aggregation operator is defined are propagated only one level up from the leaf nodes.

The aggregated value of g (using aggregation operator A) at an intermediate node in the spanning tree is denoted by g^A . For example, the value of g_{cap}^{SUM} at the root of the spanning tree would denote the total forwarding capacity of the group members.

Anycast primitive: SAAR provides an *anycast* operation that takes as arguments a *group identifier* G , a *constraint* p , an *objective function* m , and a *traversal threshold* t .

The primitive “inspects” group members whose state variables satisfy p and returns the member whose state maximizes the objective function m among the considered members. To bound the anycast overhead, at most t nodes are visited during the tree traversal. Note that a search typically considers many more nodes than it visits, due to the propagation of state variables in the tree. If $t = \perp$, the first considered node that satisfies the predicate is selected.

The predicate p over the group’s state variables specifies a constraint on the neighbor selection. Typically, the constraint is chosen to achieve the desired structure of the data overlay. A simple example predicate $p = (g_{load} < g_{cap})$ would be used to locate a node with spare forwarding capacity.

The anycast selects, among the set of nodes it inspects and that satisfy p , a node whose state variables maximize the objective function m . m is an expression over the group’s state variables and evaluates to a numeric value. For example, using the state variable g_{depth} to denote the tree depth of a member in a single-tree data plane, the objective function $m = 1/g_{depth}$ would select a node with minimum depth in the tree, among the considered nodes that satisfy the predicate $p = (g_{load} < g_{cap})$.

The anycast primitive performs a depth-first search of the group spanning tree, starting at the requester node. It uses a number of optimizations. If the aggregated state variables of a group subtree indicate that no member exists in the subtree that is both eligible (i.e., satisfies the constraint) and superior (i.e., achieves a better value of the objective function than the current best member), then the entire subtree is pruned from the search. Similarly, if the aggregated state variables of the entire tree (propagated downward from the root) indicate that no eligible and superior member exists in the tree, then the anycast terminates immediately.

create (G , set of $(g_v, A_v, f_{up}^v, f_{down}^v)$).	Creates a group with its group variables, their aggregation operators and propagation frequencies.
join (G)	This function is called by a node that wishes to join the group G .
anycast (G , p , m , t)	This function is called by a node to select a member of the group G . p is the constraint, m is the objective function, t is the maximal number of nodes visited.
update (G , set of g_v)	Called by a node to update the group with the current values of its state variables.
groupAggregateRequest (G , g_v)	Returns the value of the aggregated state variable g_v at the root of the spanning tree.
leave (G)	Called by a node that wishes to leave the group G .

Table 1: SAAR API

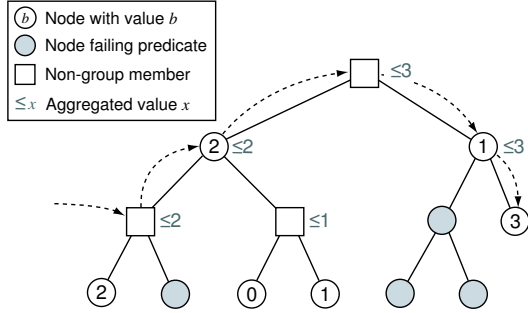


Figure 2: Anycast traversal example: Given an anycast request issued at the leftmost interior node in the group spanning tree, the anycast traverses the tree in a depth-first search. The search only visits subtrees with members that satisfy the predicate and whose value exceeds that of the current best known member.

Since the SAAR overlay construction is proximity-based, the group members are inspected roughly in order of increasing delay from the requester node n . Therefore, the result is chosen from the nodes with least delay to n , among the nodes that satisfy the constraint. This bias can be removed by starting the anycast traversal instead from a random member of the SAAR control overlay¹.

There is an inherent trade-off between accuracy and scalability in the distributed selection of overlay neighbors. Accuracy is maximized when decisions are based on complete and current information. To maximize accuracy, either (1) all nodes maintain current information about many nodes in the system, or (2) an anycast visits many nodes in the system for each peer selection. Neither approach is scalable. SAAR mitigates this tension by propagating aggregated state and by limiting the scope of anycast tree traversals based on this state. Also, in SAAR, accuracy and overhead can be controlled by bounding the anycast overhead (threshold t), and by changing the propagation frequencies of the state variables.

Example anycast traversal: Figure 2 shows an example group spanning tree. A new node wants to join the data overlay and seeks a parent that maximizes the value of an integer state variable among the nodes that satisfy

¹Here, we route the anycast message to the node responsible for a randomly drawn key in the underlying structured overlay network.

a given constraint. There are six members that satisfy the constraint. Given an anycast request issued at the leftmost interior node in the spanning tree, the anycast traverses the tree in a DFS, pruning subtrees that contain no eligible members with a value of the variable that exceeds that of the current best known member. In the example shown, the anycast stops after visiting five nodes, and yields the rightmost leaf node with the value 3. Had the anycast been invoked with a value of $t < 5$, then the anycast would have stopped after visiting t nodes, and yielded the leftmost leaf node with value 2.

SAAR API: The SAAR API contains functions to create, join and depart groups, to locate a neighbor, retrieve pre-computed aggregated group state, and to update the control plane with a member’s group variables. The operations are listed in Table 1.

3.1 Implementation

The implementation of SAAR uses Scribe [8, 9] to represent groups and to implement the anycast primitive. Scribe is a group communication system built upon a structured overlay network. Each group is represented by a tree consisting of the overlay routes from group members to the node responsible for the group’s identifier.

Due to proximity neighbor selection (PNS) in the underlying Pastry overlay [6, 21], the spanning trees are proximity-based, i.e., nodes within a subtree tend to be close in the underlying Internet. An anycast primitive walks the tree in depth-first order, starting from a node that is close to the anycast requester until an appropriate node is found [9].

Scribe does not have group variables and does not aggregate or propagate state in a group tree. Scribe’s anycast primitive does not take a constraint or objective function. Our implementation adds support for these facilities.

Nodes in a SAAR group tree aggregate and propagate state variables up and down the tree, similar to SDIMS [47]. To reduce message costs, update messages are propagated periodically, they are combined on a given overlay link across state variables and across group trees, they are piggy-backed onto other control traffic when possible, and they are multicast down the tree. During an anycast, a DFS traversal of the group tree prunes entire subtrees based on the aggregated state of

the subtree. In addition, we implemented the following optimizations:

Using network coordinates: SAAR employs virtual network coordinates to guide the depth-first search in the group trees. We used NPS [28] with a 5-D coordinate system in our implementation. Specifically, network coordinates are used to visit an interior node’s children in order of increasing distance to the anycast requester. Moreover, node coordinates can be exported as state variables; thus, they can be used to guide the selection of nodes based on their location.

Multiple spanning trees: To increase the robustness to churn in the control overlay, SAAR maintains multiple interior-node-disjoint spanning trees connecting the members of each group. Thus, the failure or departure of a node may cause a subtree disconnection and subsequent repair in at most one tree. By starting multiple traversals in different trees in parallel, anycast operations can be served in a timely fashion even while one of the spanning trees is being repaired. Interior-node-disjoint trees are constructed in Scribe simply by choosing group ids that do not share a prefix, as in SplitStream [7].

Our SAAR prototype was implemented based on the FreePastry implementation of Scribe [18]. Implementing SAAR added 4200 lines of code. Implementing a single-tree, multi-tree, and block-based CEM based on SAAR, as described in the following section, added another 1864, 2756 and 3299 lines of code, respectively.

4 Using the SAAR Control Overlay

This section describes how the SAAR control overlay can be used to construct CEMs with single-tree, multiple-tree, and block-based data overlays.

4.1 Single-Tree Multicast

To implement a single-tree multicast protocol using SAAR, each data overlay instance is associated with a SAAR group. Data overlay neighbors are selected such that (i) the neighbor has spare forwarding capacity and (ii) adding the neighbor does not create a loop in the data path. In addition, the control plane should preferentially select neighbors that (i) experience low loss, (ii) are near the requester and (iii) have low depth in the tree. These requirements can be expressed via the constraint and the objective function arguments to a SAAR anycast. Assuming the data stream has rate BW_{stream} and a node’s forwarding bandwidth is BW_{node} , we define the forwarding capacity of a node as $D = BW_{node}/BW_{stream}$.

The group associated with our single-tree data plane uses the following state variables, constraint and objective function:

- **State Variables:** $g_{cap} = D$ is the maximum number of children a node can support; g_{load} is the current number of children, g_{path} is a list of node identifiers on the node’s path to the root, g_{depth} is the length of the node’s path to the root, g_{loss} is the streaming loss rate, and g_{pd} is the path delay from the root. No aggregation operator is defined for g_{path} .

- **Constraint:** A requesting node r selects a prospective parent that has free capacity, will not cause a loop and has a loss rate less than a threshold L , using the predicate:

$$(g_{load} < g_{cap}) \wedge (r \notin g_{path}) \wedge (g_{loss} < L)$$

Alternatively, the term $(g_{loss} < r_{loss})$ can be used to select a parent that has lower loss than the requester.

- **Objective Function:** The objective function is either $1/g_{depth}^{MIN}$ or $1/g_{pd}^{MIN}$, which minimizes depth and path delay, respectively, as motivated by the findings in [39].

The source of a multicast event creates a new group and then joins it. An interested node calls the anycast method to locate a parent. Once it receives data, it joins the group, allowing the control plane to select it as a potential parent. The node uses the update method to inform the control plane of the current values of its state variables. To leave, a node disconnects from its parent and leaves the group. When a node fails or leaves, its children select a new parent using the anycast method.

Periodic data plane optimization: When a node joins or recovers from a disconnect, it uses a traversal threshold $t = \perp$ to find an eligible parent as quickly as possible.

The system gradually improves the tree’s quality by periodically (e.g. every 30 seconds) anycasting with a traversal threshold of $t = 2\log_k N$, where N is the approximate size of the group and k is the arity of the control tree. Assuming that the information from the lower levels of the tree has already been aggregated, this anycast considers information about at least $c = k^{\frac{-1+\sqrt{1+8t}}{2}}$ nodes. Suppose the eligible nodes (i.e., the ones satisfying the anycast constraint), constitute ‘ s ’ fraction of the total nodes in the group, and assume that these eligible nodes are distributed uniformly randomly in the control tree. Then, the probability of a single anycast finding a peer in the best ‘ b ’ fraction of eligible nodes (when the eligible nodes are sorted by decreasing value of the objective function) is greater than $(1 - (1 - s \cdot b)^c)$. A value of $t = 2\log_k N$ ensures that a “good” node is found with high probability. With a system size of at least 1024, a control tree arity of $k = 16$, for instance, a single anycast locates a peer in the top 10th percentile ($b = 0.1$) of eligible nodes with three-nines probability when $s = 1.0$ (i.e when all nodes are eligible), and with a probability of 92.7% when $s = 0.1$ (i.e when only 10% of the nodes are eligible).

Preemption: If a node r with a forwarding capacity $r_{cap} > 0$ is disconnected and cannot locate a new parent, r uses an anycast to locate a parent that has a child

with no forwarding capacity. (Such a child must exist, else there would be leaf nodes with spare capacity). This is done using boolean group variable g_{zdc} , which is true when a node has a zero-degree child. The anycast takes the modified predicate:

$$(g_{zdc} \vee (g_{load} < g_{cap})) \wedge (r \notin g_{path}) \wedge (g_{loss} < L)$$

Once such a parent is located, the node preempts the zero-degree child, attaches to the selected parent and adopts the preempted node as its child.

4.2 Multi-Tree Multicast

Next, we built a multi-tree CEM system similar to SplitStream [7] using SAAR. SplitStream was designed to more evenly balance the forwarding load and to reduce the impact of node and network failures, relative to a single-tree CEM. The content is striped and disseminated using k separate, interior-node-disjoint distribution trees, where each stripe has $1/k$ -th of the stream bandwidth. The constraint, objective function and state variables are the same as in the single-tree CEM. However, there is an instance of each variable per stripe. We use a single SAAR group per multi-tree data overlay, with the following state variables: $g_{coord}[i]$, $g_{cap}[i]$, $g_{load}[i]$, $g_{path}[i]$, $g_{loss}[i]$, $g_{depth}[i]$, $g_{pd}[i]$, $i \in [0, k - 1]$.

A node forwards data (i.e., accepts children) only in its *primary* stripe ps . This construction ensures interior-node-disjoint stripe trees: a node is an interior node in at most one stripe tree and a leaf in all other stripe trees. Thus, a node with forwarding capacity D (defined in Section 4.1) has

$$g_{cap}[ps] = D * k \text{ and } g_{cap}[i] = 0, \forall i \neq ps.$$

In SplitStream, the primary stripe selection is fixed by a node's identifier to allow the efficient construction of interior-node-disjoint stripe trees. This can lead to a resource imbalance when the node forwarding capacities are heterogeneous. In the SAAR-based implementation, nodes can choose their primary stripe so as to balance the available forwarding capacity in each stripe. To do this, a joining node selects as its primary the stripe with the least total forwarding capacity at that time². A node uses the aggregated value of the state variables g_{load} and g_{cap} and chooses ps to be the i that minimizes

$$(g_{cap}[i]^{SUM} - g_{load}[i]^{SUM}).$$

Even with this adaptive choice of a primary stripe, it is still possible that the departure of a node causes a stripe to be momentarily left with no forwarding capacity, until another node joins. As in SplitStream, a number of tree transformations are possible in this case [7], which can be easily expressed in SAAR. As a last resort, a child relaxes the predicate to select a parent with forwarding capacity in a different stripe, at the expense

²Note that with a coding scheme like MDC [20], the stripes are equivalent from the perspective of the application.

of interior-node-disjointness. The system behaves like SplitStream in this respect, except that flexible primary stripe selection significantly reduces the likelihood of stripe resource exhaustion.

Moreover, the SAAR-based implementation can support any number k of stripes, allowing the choice to match the needs of the application coding scheme. To achieve good load balance in SplitStream, on the other hand, the number of stripes must correspond to the routing base in SplitStream's underlying Pastry overlay, which is a power of 2. The flexible choice of k , and the flexible primary stripe selection, are two examples where the power of SAAR's anycast primitive makes it possible to relax constraints on the data plane construction in the original SplitStream implementation.

The constraint and objective function used to locate parents now apply on a per stripe basis. For example, to locate a parent in stripe s , the corresponding predicate is

$$(g_{load}[s] < g_{cap}[s]) \wedge (r \notin g_{path}[s]) \wedge (g_{loss}[s] < L).$$

4.3 Block-Based Multicast

In block-based CEMs [49, 30], a random mesh connects the members of the data overlay, and a swarming technique is employed to exchange blocks amongst the mesh neighbors.

We use SAAR to select and maintain mesh neighbors, rather than the commonly used random walk [44] or gossiping [19] techniques. In Section 5.4, we will briefly describe the swarming algorithm (based on existing literature) we have implemented in our block-based prototype.

Mesh Neighbor Selection: A member n with forwarding capacity D (defined in Section 4.1) maintains between M (e.g., 4 as in Coolstreaming [49]) and $M * D$ neighbors. In steady state, a node expects to receive $1/M$ of the total stream bandwidth from a particular neighbor; thus the minimum number of neighbors is M . Nodes use SAAR anycast to maintain M neighbors of good quality and accept up to $D * M$ neighbors.

To ensure that the mesh construction has sufficient path diversity, we anycast with $t = \perp$ but start from a random group member, so that the selected node is not necessarily near the requester. In addition, each node periodically locates fresh neighbors, even if it has M neighbors of good quality. We have observed that without this periodic update, nodes that joined early tend to have their neighbor capacity exhausted and thus they lack links to nodes that joined much later, resulting in a low path diversity and high depth.

A SAAR group associated with a block-based data plane uses the following state variables and constraint (no objective function is used):

- *State Variables:* g_{cap} is the maximum number of neighbors ($D * M$), g_{load} is the current number of neighbors, g_{loss} is the loss rate.

- *Constraint:* The predicate is $(g_{load} < g_{cap}) \wedge (g_{loss} < L)$.

Note that the loop-freedom constraint needed in tree-based systems is not present.

5 Experimental Evaluation

We begin with a description of the experimental setup. With the exception of the Planetlab [33] experiments in Section 5.5, we use Modelnet [42] to emulate wide-area delay and bandwidth in a cluster of PCs connected by Gigabit Ethernet, each with a 2.6 Ghz CPU and 4 GB of main memory. We chose Modelnet because it allows a meaningful comparison of various systems and protocols, as we can deterministically reproduce the network conditions for each experiment.

Using up to 25 physical cluster nodes, we emulate a network with 250 stubs. (The Modelnet core ran on a separate cluster node.) The delays among the stubs were randomly chosen from the King [22] data set of measured Internet delay data. Four client nodes are attached to each stub network for a total of 1000 client nodes. The client nodes are connected via 1 ms links to their respective stubs. Neither the access links nor the stub network were the bottleneck in any of the experiments. Similarly, we ensured that the CPU was not saturated during the experiments on any of the cluster nodes.

We emulated overlays of 250–900 virtual nodes. To emulate an overlay of size n , we randomly selected n client nodes to participate in the overlay. The forwarding capacity (as defined in Section 4.1) of virtual nodes was limited via their node degrees. The node degrees are heterogeneous and follow the measured distribution from the Sripanidkulchai et al. study of live streaming workloads [39].

However, we use a minimum node degree of one in the experiments to ensure that some forwarding capacity is always available during random node joins and departures. Also, we impose a maximum degree cap to achieve a given mean Resource Index (RI), where mean RI is the ratio of the total supply of bandwidth to the total demand for bandwidth. Unless stated otherwise, we use degree caps of (MIN=1,MAX=6) to achieve a mean RI=1.75. The degree distribution after enforcing the caps is as follows: approximately 76.85% of degree 1, 9.5 % of degree 2, 0.34% each of degree 3, 4 and 5, and 12.4% of degree 6. Unless stated otherwise, the multicast source had a degree of 5.

In the Modelnet experiments, we streamed data from a single source node at a constant rate of 32 Kbps. We chose this low rate to reduce the load on the Modelnet

emulation. This does not affect the results, since we are interested in control efficiency and its impact on the quality of the data streams. Since the streaming rate is identical in all systems and we are primarily interested in control overhead, we exclude data packets when measuring message overheads. We evaluate the performance of the various systems using the metrics described in Table 2.

In all experiments, a single SAAR control overlay is used that includes all participating nodes, irrespective of their data overlay membership. We use a single spanning tree per SAAR group in scenarios without control overlay churn, and two trees per group otherwise. All reported results are the averages of at least 2 runs. Error bars, where shown, indicate the minimal and maximal measured values for each data point. In cases where no error bars are shown in the plots, the deviation among the runs was within 3%.

5.1 Effectiveness of SAAR Anycast

Our first set of experiments evaluate the performance of SAAR’s anycast primitive. No data was streamed in these experiments.

Locality-awareness: We evaluate the anycast primitive’s ability to find peers with low delay. We run SAAR with a traversal threshold t of \perp and 2, respectively. To isolate the effects of using NPS coordinates during tree traversal, we evaluate SAAR with and without NPS coordinates (SAAR-NO-NPS), and compare its performance against a centralized system where peers are chosen either randomly (CENTRAL-Random) or using NPS coordinates (CENTRAL-NPS). CENTRAL-Global reflects the optimal greedy peer selection based on global knowledge.

We use a 250 node overlay and 10 groups. Peers subscribe to each group with a probability of 0.1, resulting in an expected group size of 25 peers. Figure 3 shows that SAAR’s ability to select nearby peers comes close to that of a centralized solution that uses NPS coordinates. Using NPS in the tree traversal significantly improves the results, though even the result without NPS (corresponding to a plain Scribe anycast) is significantly better than random peer selection.

Load awareness: Next, we evaluate SAAR’s ability to quickly select peers with available bandwidth under conditions with a low Resource Index (RI), where there are few nodes with spare bandwidth. Figure 4 compares SAAR with a centralized peer selection service, while building a single-tree data overlay of $N = 350$ nodes. The centralized peer selection service was placed on a node such that the average delay to the remaining nodes in the underlying 250-node stub network was minimized.

We experimented with RI=1.01, in which all nodes have exactly degree 1, (except the source, which has degree 5) and another setting of RI=1.23 with degree cap

Channel Join Delay	The delay from the instant a node joins a multicast event until it receives 90% of the stream rate.
Tree Depth	The depth of a node in the dissemination tree.
Continuity Index	The fraction of the unique data packets streamed during a node's membership that were received by the node.
Datastream Gap	The time during which a node fails to receive data due to the departure of a node in the data plane.
Node Stress	Total number of control messages (not data messages) sent and received per second, per node.

Table 2: Evaluation metrics.

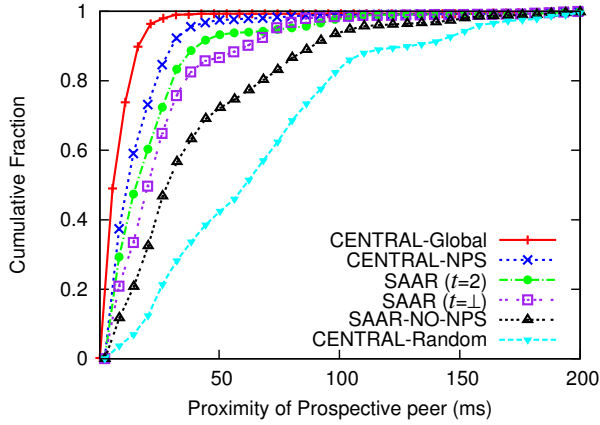


Figure 3: Locality awareness

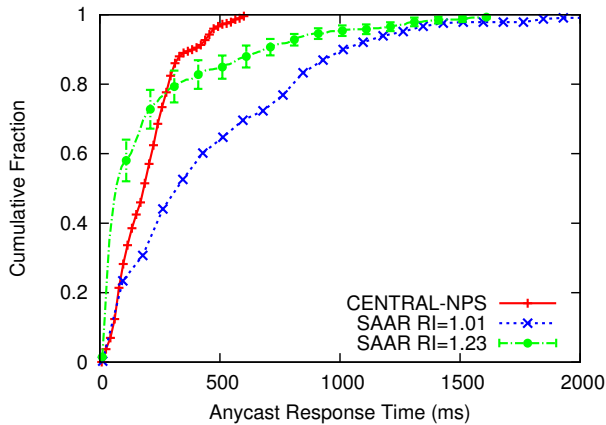


Figure 4: Load awareness

(MIN=1,MAX=2). Even under the harsh RI=1.01 setting, SAAR anycast can select a peer within 1 second in 90% of the cases. This is because SAAR's anycast tree traversal prunes subtrees with no capacity based on aggregated information. When a moderate amount of spare bandwidth is available (RI=1.23), 78% of the anycast response times are even lower than those of the centralized server, because SAAR's anycast can usually find a peer by contacting a node that is closer than that server.

During the experiment, the average/median/99th percentile number of tree nodes visited during an anycast was 3.2, 3, and 4 with RI=1.01, and 2.3, 2, and 4 with RI=1.23. The 95th percentile and the maximum of the total message overhead during the experiment was less than

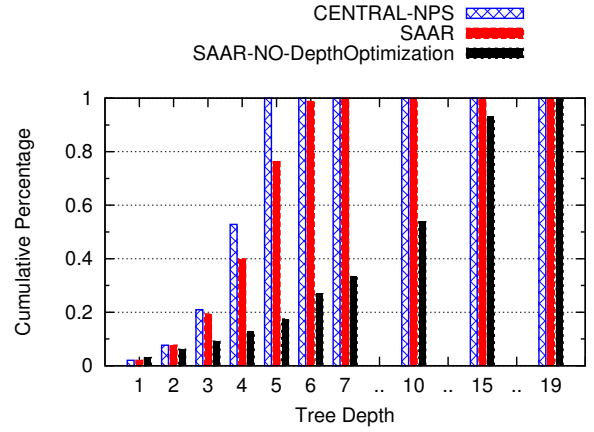


Figure 5: Tree depth optimization

4 msgs/sec and 18 msgs/sec, respectively with RI=1.01; it was less than 3 msg/sec and 12 msgs/sec with RI=1.23. The maxima were reached at the root node of the group tree in each case.

Tree depth optimization: The next experiment shows that SAAR anycast can optimize for metrics like tree depth effectively. We compare the achieved tree depth with that of a centralized membership server. 250 nodes join a channel over a period of 120 seconds. We set the maximum tree traversal threshold $t = 4$. During the traversal, we use aggregated state variables to prune subtrees in which the depth of the minimum depth peer with available capacity is greater than the best choice we have so far. Figure 5 shows that the tree depths resulting from using SAAR anycast are almost as good as those obtained with the centralized server. For comparison, we also included the result for the case when SAAR is being used without an objective function to minimize tree depth. Moreover, the anycasts had low traversal overhead. The 95th percentile and the maximum of the total message overhead during this experiment was less than 3 msgs/sec and 34 msgs/sec, respectively.

In summary, the decentralized SAAR control plane can effectively and efficiently select nearby nodes, nodes with spare capacity, and select nodes subject to metrics like tree depth.

Control overheads: Next, we present results of a simple analysis of SAAR's control overhead. Assume that we construct a SAAR overlay with N nodes using base- k routing in the underlying Pastry network. State variables

are propagated once every second. There are G groups in the system, and the average group size is g . Define $T = g * G * \log_k N$; T is an upper bound on the number of edges in *all* the control trees in the system.

The aggregation analysis considers two cases (when $T \leq kN$ and when $T > kN$). If $T \leq kN$, then an upper bound on the average number of control messages sent and received per node per second due to state aggregation $S = 2 * \frac{T}{N}$. If $T > kN$, then $S = 2 * (k - 1) * \log_k \frac{T}{N}$. Now let, on average, there be a anycasts per second per group in the system. The upper bound on the average number of anycast messages per node per second is simply $2 * \frac{\log_k N}{N} * a * G$.

Consider a large system with 10^6 nodes, a small number of large groups (ten groups of 10^5) and many small groups (10^5 groups of ten) in a SAAR overlay with $k = 16$. For an average node, the aggregation overhead in this case is no more than 20 msgs/sec. Even if we assume that each group turns over every 5 minutes, then the anycast overhead is less than $\frac{1}{7}$ msgs/sec for the average node.

In another configuration, assume that every node is a member of one group. Irrespective of the size distribution of the groups, $g * G = N$. Here, the average aggregation overhead is no more than 10 msg/sec and the corresponding anycast overhead is less than $\frac{1}{15}$ msg/sec in this case.

These results are consistent with our measured average node stress results. We note that the stress at nodes near the root of a control tree are significantly higher than average in our implementation. To some extent, this is inherent in any tree-based control structure. The difference between maximal and average node stress could be reduced by limiting the degree of nodes, at the expense of somewhat deeper trees. Also, the node stress tends to balance as the number of groups in the same SAAR overlay increases, because the group tree roots (and thus the nodes near the root) are chosen randomly.

5.2 SAAR for Single-Tree Data Overlays

Next, we show the benefits of using the SAAR control overlay in the design of a single-tree CEM. We compare the performance of the native single-tree ESM system [10] with a modified implementation of ESM using SAAR.

350 nodes join a single-tree CEM and continue to leave/rejoin the multicast channel with an exponentially distributed mean session time of 2 minutes and a minimum of 15 seconds. To achieve a large mean group size, the nodes rejoin the same multicast channel after an off-line period of 10 sec. The experiment lasts for 1 hour. We compare the performance of four systems below. All systems attempt to minimize the tree depths while locating parents.

Native-ESM: We use ESM [10] as an example of a single-tree CEM based on an unstructured overlay. A single overlay is used for control and data. The overlay is optimized for data dissemination; state information is disseminated epidemically to enable peer selection.

Scribe: We use Scribe as an example of a single-tree CEM based on a structured overlay. A single overlay is used for control and data. Scribe’s standard pushdown policy is used to enforce the degree bounds at the intermediate nodes in the Scribe tree [8].

SAAR-ESM: A version of ESM that uses a shared SAAR overlay for control. Nodes remain in the SAAR control overlay with an exponentially distributed session time with a mean of 30 minutes, for the experiment duration. As before, peers switch between data overlays with mean session time of 2 minutes and a minimum of 15 seconds. As our results show, nodes have an incentive to remain in the control plane longer than in any particular data channel, because it enables them to join and switch between channels much faster while the overhead is very low.

SAAR-ESM-Unshared: To isolate the benefits of a more stable control plane membership, we make nodes join/leave the SAAR control overlay whenever they join/leave a multicast group in this system. Otherwise, the system is identical to SAAR-ESM.

Figure 6 shows the results of our experiments. Among all systems, SAAR-ESM achieves easily the best results for join delay, continuity and node stress. SAAR-ESM-Unshared appears to beat SAAR-ESM in terms of tree depth. This comparison turns out to be misleading, however, because the average steady-state group size achieved by SAAR-ESM-Unshared during the experiment is only 55% that of SAAR-ESM’s, due to the large difference in join delays in combination with churn. The average group sizes in the experiment are 225 (Native-ESM), 180 (Scribe), 290 (SAAR-ESM) and 160 (SAAR-ESM-Unshared), respectively.

The long tail in the SAAR-ESM join delay distribution corresponds to the initial joins when a node is not yet a member of the control overlay. Subsequent joins exhibit very low join delay: 99.8% of such joins had a delay of less than 1.5 sec. Native-ESM exhibits higher join delay and a lower continuity index than SAAR-ESM. Additional results (not shown) show that this gap widens with higher churn or larger groups. This is because in Native-ESM, state information propagates slowly, causing increasing staleness as churn or group size increases.

The results confirm earlier observations that Scribe exhibits deep trees and relatively high join delay under churn or when the node capacities are heterogeneous [3]. One reason is that the overlay structure imposes constraints on the data plane, resulting in Scribe pushdown being the norm rather than the exception. Another reason

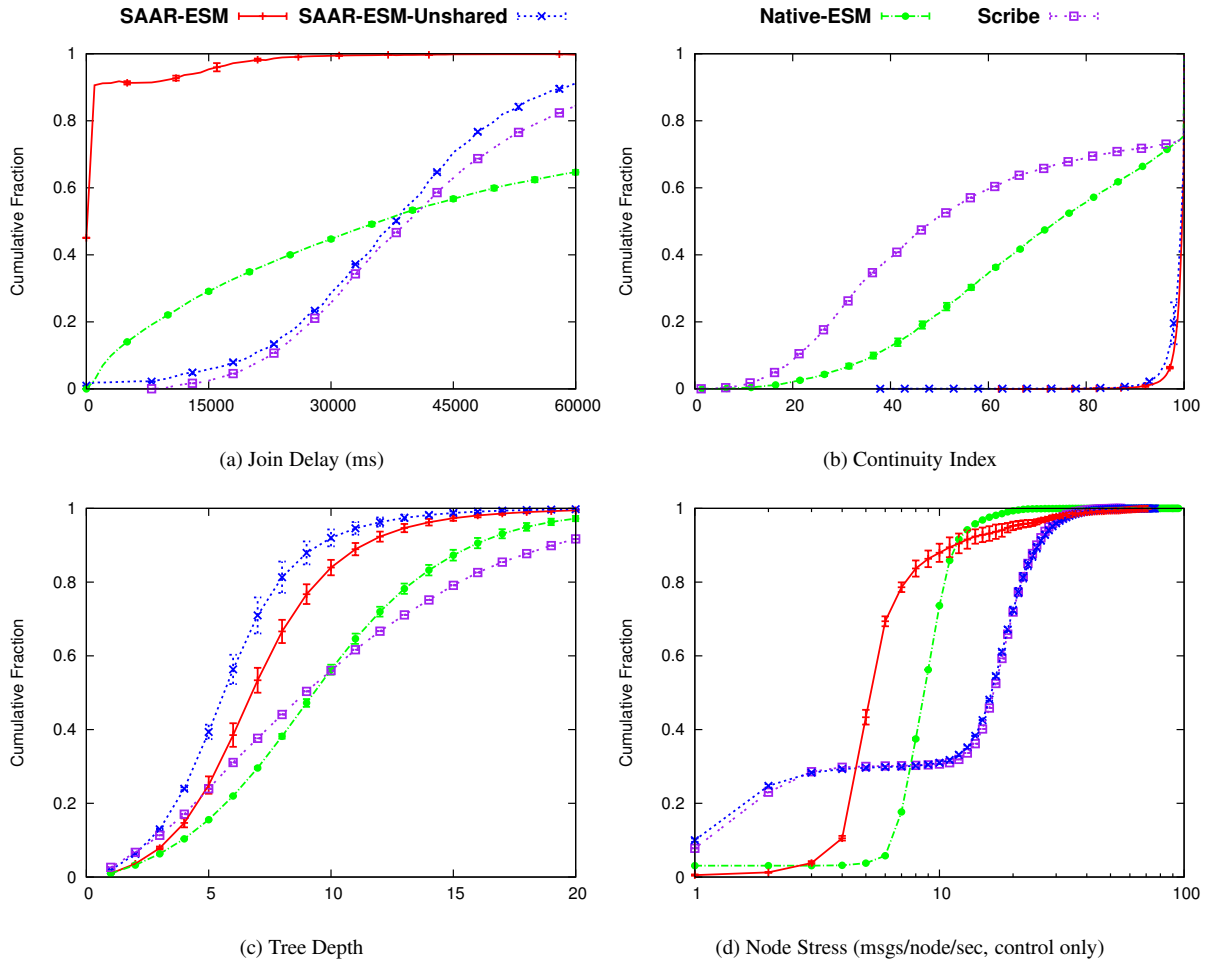


Figure 6: Single-tree CEM performance

is that churn disrupts the coupled control and data overlay in Scribe. The combined effect is higher tree repair time, which leads to a poor continuity index.

The node stress incurred by SAAR-ESM is generally lowest among all systems, except for a longer tail. (The tail is a result of higher node stress near the top of the group tree.) This result indicates that the overall reduction in control churn in the shared SAAR overlay more than outweighs the additional overhead for maintaining a separate control overlay.

Comparing SAAR-ESM and SAAR-ESM-unshared confirms that a shared control overlay leads to lower join delays, better continuity and reduced node stress. Even SAAR-ESM-Unshared, however, yields dramatically better continuity than Native-ESM and Scribe. This speaks to the power of the SAAR anycast primitive, independent of the control plane sharing.

We believe that shielding the control plane from churn due to channel switching, as provided by a shared SAAR

control overlay, is a critical optimization for applications like IPTV [23]. There, join and switching delays are very important and users switch among channels much more frequently than they start/stop their IPTV application.

We performed additional experiments comparing the performance of Native-ESM and SAAR-ESM with different levels of data and control overlay churn, and under flash crowd conditions. We also compared SAAR-ESM with a centralized membership service in terms of scalability.

Lower Membership Churn: We repeated the previous experiment with nodes now leaving/rejoining the multicast channel with a mean session time of 5 minutes. Figure 7 compares the join delays and the continuity index of SAAR-ESM and Native-ESM. The join delays in Native-ESM improved significantly, with a 75th percentile of 15 sec. For SAAR-ESM, as before the long tail in the join delay distribution corresponds to the initial joins when a node is not yet a member of the control overlay. As be-

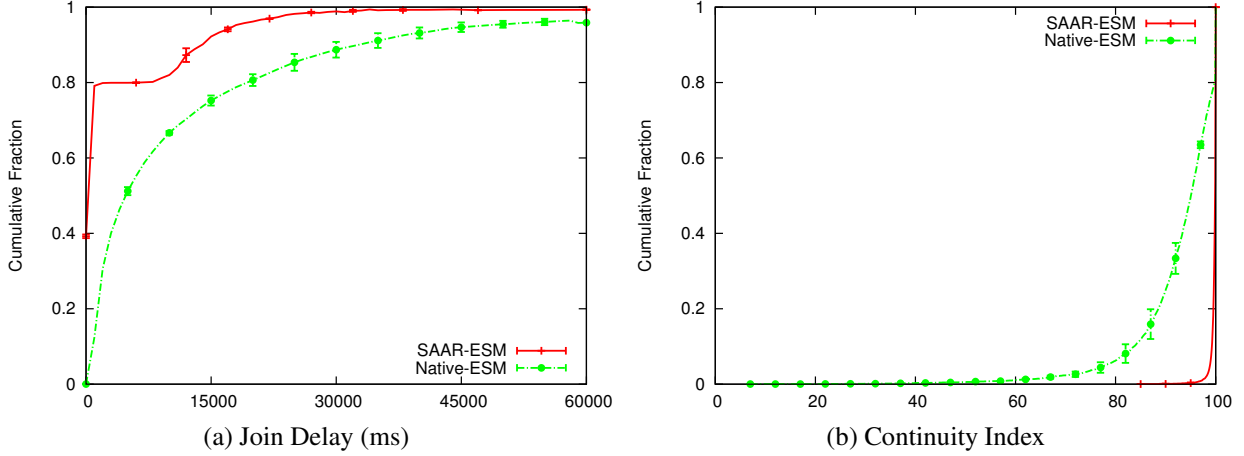


Figure 7: Low membership churn: single-tree CEM

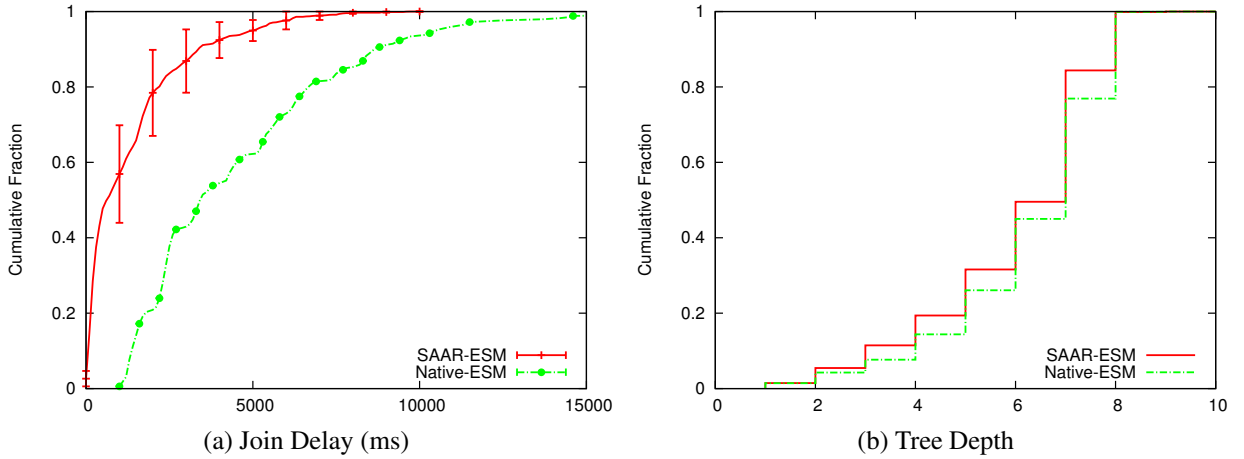


Figure 8: Flash crowds: single-tree CEM

fore, 99.8% of joins where a node was already a part of the SAAR overlay completed within 1.5 secs. However, as compared to the results in Figure 6, the tail is thicker because a greater fraction of joins incur the initial control overlay join delay, since the ratio of the data overlay session time to the control overlay session time (i.e mean of 30 minutes as before) is higher. The continuity index of Native-ESM also improved significantly, with a 75th percentile of over 90. For SAAR-ESM, the 98th percentile continuity index improved to over 98.

These results show that Native-ESM, which was not designed for very high membership churn, performs well under less severe membership dynamics. However, even under these conditions, SAAR lends ESM significantly better performance.

Control overlay churn: We also evaluated SAAR-ESM with different levels of churn in the control overlay, and with one or two spanning trees per SAAR group.

Even at a mean control overlay session time of only 5 minutes (exponential distribution) and an overlay size of 350 nodes, using 2 trees per group yields anycast performance comparable to that of a single tree with no overlay churn. An exception are join events that involve a control overlay join: consistent with the results in Figure 6(a), these have a noticeably higher join delay. The overhead doubles when two trees are used, but the average overhead is still modest at 10 messages/second/node.

Flash crowds: A group of 350 nodes join an event within 15 seconds, and remain in the group for 15 minutes. For SAAR-ESM, the nodes are already part of the control overlay when the experiment starts.

Figure 8 compares the join delays and the tree depth of SAAR-ESM and Native-ESM. SAAR-ESM is able to connect nodes to the data overlay quickly: the 90th percentile of the join delay distribution is less than 4 secs. The corresponding 90th percentile for Native-ESM is more than 8 secs. SAAR-ESM is able to maintain low

tree depths, with an 80th percentile of 7 and a maximum of 8. For comparison, in Native-ESM, the 80th percentile and the maximum tree depths are both 8. In SAAR-ESM, 90% of nodes have a stress of less than 3, while Native-ESM has an average node stress of 21.

These results show that SAAR’s anycast primitive yields significantly lower join delay and lower overhead than Native-ESM under flash crowd conditions, while maintaining comparable tree depth.

Scalability: We compare SAAR-ESM with a version that uses a centralized membership service. We use data overlays of sizes 54, 180, 540 and 900 nodes, all at a mean data session time of 2 minutes (exponential distribution) and a minimum of 15 secs. In SAAR-ESM, all nodes join the control overlay before the start of the experiment and remain in the overlay during the experiment.

The centralized membership service handled 50, 177, 662, 1084 messages/second at the various overlay sizes, showing an expected linear increase in the load. Due to the resulting bottleneck, the 90th percentile peer selection delays of the central membership service increases as 750 ms, 1.1 sec, 2.4 sec, 18 sec, respectively, for the different overlay sizes. For SAAR-ESM, the 90th percentile anycast delay increases from 600ms at 54 nodes to only 1.2 seconds at 900 nodes. The average continuity index achieved with the centralized membership service and SAAR-ESM at a group size of 900 was 80.3 and 97.6, respectively. This clearly demonstrates the scalability of the SAAR control plane.

In summary, the results clearly show that SAAR’s efficient anycast primitive yields ESM superior join delays, better content delivery quality, increased robustness to churn and increased scalability. Moreover, the shared SAAR control overlay dramatically reduces join delays and increases efficiency by reducing membership churn in the control plane. Additionally, these benefits are realized at a lower overhead than Native-ESM and Scribe. Comparing the SAAR-ESM with the native ESM system, we have shown that using a decoupled, shared control plane can achieve the best of both data dissemination quality and control efficiency. SAAR is effective in constructing high quality data overlays under flash crowd scenarios and high data overlay dynamics, while tolerating control plane churn. Finally, unlike a centralized membership service, the decentralized design of SAAR allows it to support data overlays of large size.

5.3 SAAR multi-tree CEM performance

To show the effectiveness of SAAR in supporting multi-tree data planes, we have implemented a prototype multi-tree CEM based on SAAR, as described in Section 4.2. We use five data stripe trees per data overlay. 350 nodes

with heterogeneous degree distribution (mean RI=1.23 and degree caps MIN=1,MAX=2) join the control overlay in the first 10 minutes, and then join the data overlay in the next 5 minutes. They remain in the data overlay for another 10 minutes, and then continue to leave/rejoin the data overlay with a mean session time of 2 minutes (exponential distribution) and a minimum of 15 seconds for the remainder of the experiment. To achieve a large instantaneous group size, nodes re-join the same data overlay 10 seconds after leaving. Nodes do not depart from the control overlay during the experiment, which lasted for approximately one hour. We show that SAAR can effectively balance resources among the stripe trees despite constrained resources and heterogeneous node degrees. We also measure the resulting join delay, continuity index and control overhead.

Figure 9(a) shows the instantaneous group size, as well as the minimum and the maximum of the total forwarding resources among the stripes. The minimum resources are always above the demand, i.e., the group size. The fluctuations in stripe resources result from membership churn. For instance, when a node of degree 6 leaves, the capacity in its primary stripe drops by $6 * 5 = 30$ units. In all cases, however, the imbalance is quickly rectified due to the adaptive primary stripe selection policy for newly joining nodes.

Figure 9(b) plots, for each data sequence number, the minimum number of stripes that 95% of the nodes are able to receive. Every second, the multicast source increments the sequence number. For each sequence number, there are 5 data packets generated, one for each stripe. Thus, a value of 4 stripes received means that 95% of the nodes are able to receive 4 or more stripes.

Figure 9(c) shows the CDF of the join delay of nodes, reflecting how long it took to receive data on different numbers of stripes. Assuming that receiving 4 out of 5 stripes is sufficient to construct the full data stream (e.g. using redundant coding like MDC/Erasure coding), the 95th percentile join delay is 2.6 seconds.

Figure 9(d) shows the CDF of the continuity index among the nodes, calculated with respect to the fraction of data bytes received on all stripes. The average continuity index observed was 99.1. The average node stress (not shown) on the control plane while supporting the multi-tree data overlay is low, with 90% of the nodes handling less than 4 msgs/sec and a maximum node stress of 90 msgs/sec.

We also performed experiments with a higher RI=1.75 and a resulting wider range of node degrees (MIN=1, MAX=6). The results are virtually identical, with a 95th percentile join delay for acquiring 4 out of 5 stripes of 2.2 seconds and an average continuity index of 99.2.

We conclude that SAAR can effectively support a multi-tree data plane design. SAAR can ensure resource balance among the interior-node-disjoint stripe trees in

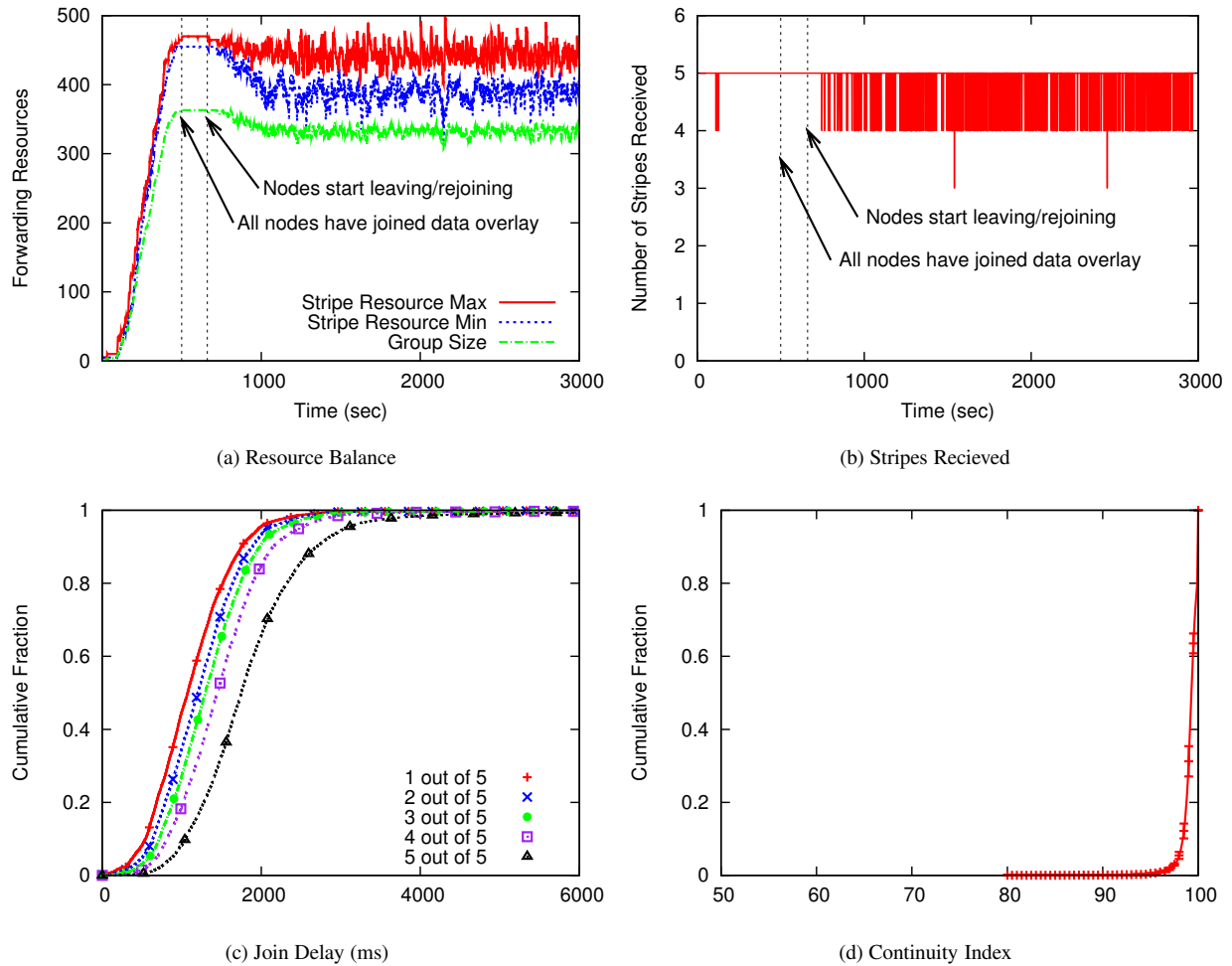


Figure 9: SAAR multi-tree CEM performance

heterogeneous environments. As a result, the resulting CEM system simultaneously realizes the benefits of performance optimized data overlays and the benefits of a multi-tree design in terms of tolerance to loss and membership churn.

5.4 SAAR for Block-based Data Overlays

We built a prototype block-based swarming CEM based on SAAR. The mesh construction, as described in Section 4.3, borrows from Chunkyspread [43]. The swarming block-exchange algorithm we use closely follows Coolstreaming [49]. Briefly, we stripe content into 1 second blocks. Once every second, neighbors exchange their block availability within a sliding window of blocks covering 60 seconds. Missing blocks in this 60 sec buffer are requested randomly from the neighbors in inverse proportion to their bandwidth utilizations. We additionally implemented Request-Overriding as explained in

Chainsaw [30] to ensure that the multicast source sends out every block at least once.

350 nodes with a heterogeneous degree distribution (mean RI=1.23 and degree caps MIN=1,MAX=2) join the control overlay in the first 10 minutes, and then join the data overlay in the next 5 minutes. They remain in the data overlay for another 10 minutes and then continue to leave/rejoin the data overlay with a mean session time of 2 minutes (exponential distribution) for the remainder of the experiment. We enforce a minimum session time of 60 seconds to allow them to fill their initial buffer worth of 60 secs. To achieve a large instantaneous group size, nodes re-join the same data overlay 10 secs after leaving. Nodes do not depart from the control overlay during the experiment, which ran for approximately one hour.

Figure 10 shows the CDF of the continuity index, and the distribution of overlay hops taken per block in the mesh. The average continuity index is 91.6. The average control node stress (not shown) is low, with 90% of

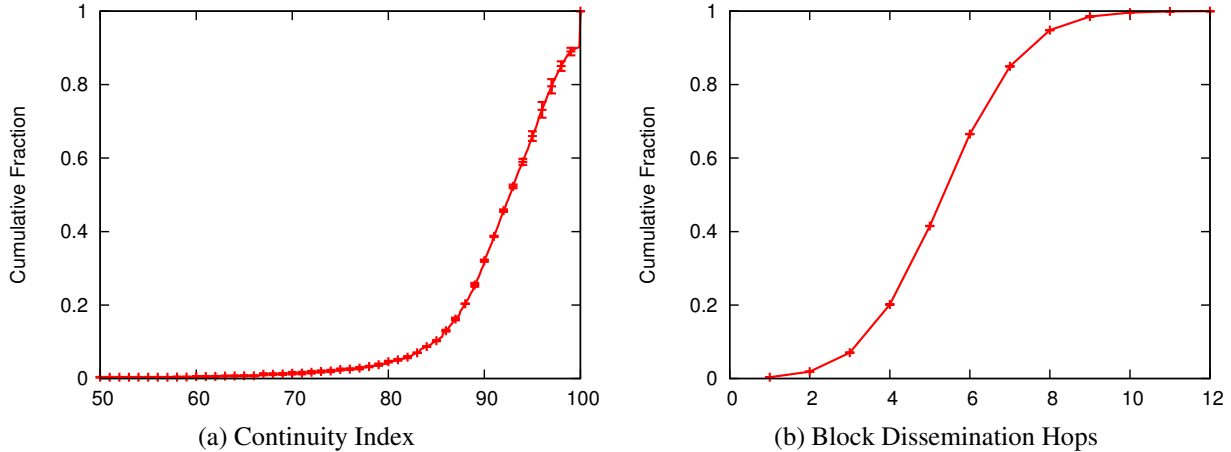


Figure 10: SAAR block-based CEM performance

the nodes handling less than 3 msg/sec and a maximum node stress of 80 msg/sec. Note that the join delay in block-based systems is dominated by the size of the sliding block window, 60 secs in this case.

We also performed experiments with degree caps of (MIN=1, MAX=6, RI=1.75). Here, the average continuity index improved to 96.8, while the distribution of overlay hops was similar.

An additional experiment matches a configuration reported in published results for Coolstreaming [49]. 150 nodes with homogeneous node degrees and RI=1.25 are part of the SAAR control overlay and join a data channel within 1 min. At a mean data overlay session time (exponentially distributed) of 50/100/200 sec, we measured an average continuity index of 89, 94 and 98, respectively. The corresponding results reported for Coolstreaming are 89, 91 and 94, respectively. Thus, our implementation appears to perform on par with Coolstreaming. However, differences in the experimental conditions (Modelnet vs. Planetlab, RI=1.25 vs. unspecified RI, 32 Kbps vs. 500 Kbps streaming rate) do not support a stronger conclusion. In summary, our results show that SAAR can support block-based swarming CEMs effectively.

5.5 Planetlab Experiment

To demonstrate that our SAAR-ESM prototype can realize its benefits when deployed in the Internet, we performed an additional experiment in the Planetlab testbed. We use a single-tree CEM with a streaming data rate of 100 Kbps. Approximately 125 nodes (chosen randomly across all continents among nodes that had reasonable load averages) join a channel in the first 2 minutes and then continue to leave/rejoin the channel with a mean session time (exponential distribution) of 2 minutes and a minimum of 15 seconds, for an experiment duration of

15 minutes. The node-degree distribution was heterogeneous and used caps of (MIN=1,MAX=6), RI=1.75 In SAAR-ESM, the nodes are part of the control overlay at the start of the experiment, and they do not depart the control overlay during the experiment. Two group spanning trees were used in SAAR to mitigate the effects of excessive scheduling delays due to high loads on Planetlab machines, which can affect anycast response times.

Figure 11 compares SAAR-ESM and Native-ESM with respect to join delay and continuity index. SAAR-ESM has a 90th percentile join delay and tree repair time (not shown) of 2.5 seconds, which results in good continuity indices. Under high churn, Native-ESM is not able to locate neighbors fast enough. Therefore, it suffers from higher join delay and tree repair times, which result in a lower continuity index.

The absolute results obtained with SAAR-ESM on Planetlab are not as good as in Modelnet, although the same trends hold. In absolute terms, the 90th percentile anycast response time in one group spanning tree increased from 500 msec in Modelnet to 3.5 seconds on Planetlab, although the number of anycast hops taken was similar. The continuity indices decreased accordingly. We traced the cause to excessive processing delays on Planetlab nodes, where the 50th and 90th percentile load averages³ were approximately 10 and 20, respectively. Planetlab is a shared testbed infrastructure that tends to be heavily oversubscribed. We believe that most deployments in the Internet would likely encounter less loaded nodes, and thus achieve results much closer to our Modelnet results.

Native-ESM appears to be less sensitive to the excessive scheduling delays in Planetlab than SAAR-ESM. The likely reason is its proactive epidemic membership protocol, which maintains a list of multiple candidate

³5 minute average as reported by 'uptime'

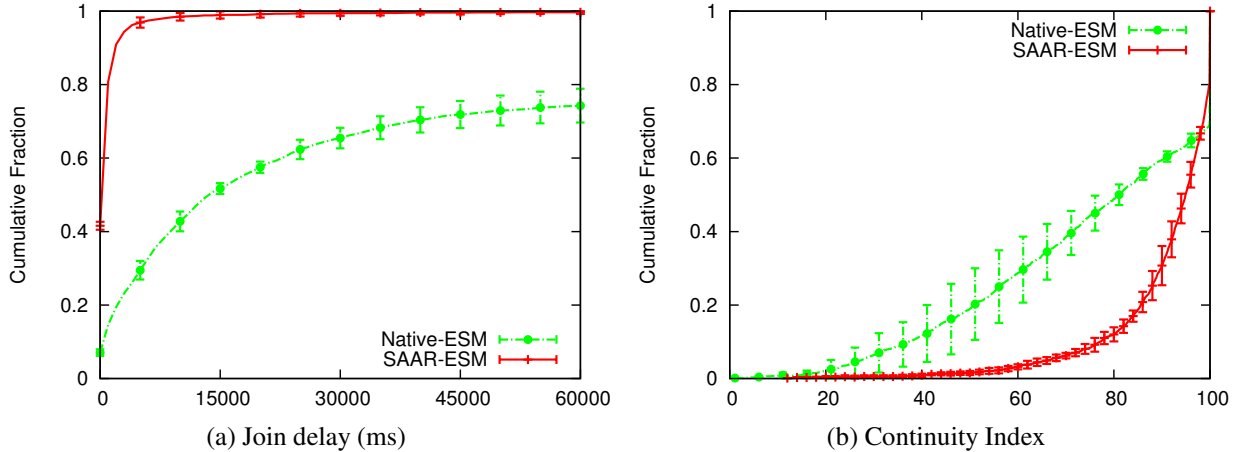


Figure 11: Planetlab single-tree CEM performance

neighbor nodes at all times. SAAR-ESM could implement an optimization that would have a similar effect: cache the results of previous anycasts and attempt to use nodes on this list while starting a new anycast in parallel. We have not yet implemented this optimization, since we are not convinced it is necessary in most practical deployments.

6 Conclusions

We have presented SAAR, a shared control overlay for CEM systems. SAAR separates the control mechanism from the policy of peer selection in CEM systems. By factoring out the control plane into a separate overlay network, SAAR enables powerful and efficient peer selection, while avoiding constraints on the structure of the data dissemination overlay. Moreover, once decoupled, the control overlay can be shared among many data overlay instances. This sharing increases efficiency and dramatically reduces the delay for joining a channel or switching between channels, which is critical for IPTV.

SAAR's anycast primitive locates appropriate data overlay neighbors based on a constraint and an objective function. The primitive can be used to build and maintain a variety of data overlay organizations. We evaluate a prototype implementation of SAAR experimentally. The results show that SAAR can effectively support single-tree, multi-tree and block-based data plane organizations. Its control efficiency allows it to achieve rapid channel join/switching and high content dissemination quality at low overhead, even under high churn and at large scale.

7 Acknowledgments

This work was supported in part by the Max Planck Society, by National Science Foundation grants (ANI-

0225660, CNS-0520187, CNS-0085920, CNS-0435382, CNS-0448546, ANI-0092806) and by Bhattacharjee's Slowly Fellowship. We would like to thank Jeff Hoyer, Andreas Haeberlen and Alan Mislove for their help and advice. We would also like to thank the anonymous reviewers and our NSDI'07 shepherd Albert Greenberg for helpful feedback on earlier versions of the paper.

References

- [1] Akamai FreeFlow. <http://www.akamai.com>.
- [2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proc. of ACM SIGCOMM*, Aug. 2002.
- [3] A. Bhambe, S. Rao, V. Padmanabhan, S. Seshan, and H. Zhang. The impact of heterogeneous bandwidth constraints on DHT-based multicast protocols. In *Proc. of IPTPS '05*, Feb. 2005.
- [4] S. Bhattacharjee, M. H. Ammar, E. W. Zegura, V. Shah, and Z. Fei. Application-layer anycasting. In *Proc. of INFOCOM'97*, pages 1388–1396, 1997.
- [5] R. Braynard, D. Kostic, A. Rodriguez, J. Chase, and A. Vahdat. Opus: An overlay peer utility service. In *Proc. of 5th International Conference on Open Architectures and Network Programming (OPENARCH '02)*, June 2002.
- [6] M. Castro, P. Druschel, Y. Hu, and A. Rowstron. Proximity neighbor selection in tree-based structured peer-to-peer overlays. Technical Report MSR-TR-2003-52, Microsoft Research, 2003.
- [7] M. Castro, P. Druschel, A. Kermarrec, A. Nandi, A. Rowstron, and A. Singh. SplitStream: High-bandwidth multicast in a cooperative environment. In *Proc. of SOSP 2003*, Oct. 2003.
- [8] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8), Oct. 2002.
- [9] M. Castro, P. Druschel, A.-M. Kermarrec, and A. Rowstron. Scalable application level anycast for highly dynamic groups. In *Proc. NGC'2003*, Sept. 2003.
- [10] Y. Chu, A. Ganjam, T. Ng, S. Rao, K. Sripanidkulchai, J. Zhan, and H. Zhang. Early experience with an Internet broadcast system based on overlay multicast. In *Proc. of USENIX Annual Technical Conference*, 2004.
- [11] Y. Chu, S. Rao, and H. Zhang. A case for end system multicast. In *ACM Sigmetrics*, pages 1–12, June 2000.
- [12] B. Chun, P. Wu, H. Weatherspoon, and J. Kubiatowicz. ChunkCast: An anycast service for large content distribution. In *Proc. of IPTPS '06*, Feb. 2006.

- [13] F. Dabek, B. Zhao, P. Druschel, J. Kubiawicz, and I. Stoica. Towards a common API for structured peer-to-peer overlays. In *Proc. IPTPS '03*, Feb. 2003.
- [14] S. Deering and D. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Transactions on Computer Systems*, 8(2), May 1990.
- [15] J. Dunagan, N. Harvey, M. Jones, M. Theimer, and A. Wolman. Subscriber/volunteer trees: Polite, efficient overlay multicast trees. Technical Report MSR-TR-2004-131, Microsoft Research, 2004.
- [16] M. Freedman, E. Freudenthal, and D. Mazieres. Democratizing content publication with Coral. In *Proc. NSDI '04*, Mar. 2004.
- [17] M. Freedman, K. Lakshminarayan, and D. Mazieres. Oasis: Anycast for any service. In *Proc. NSDI '06*, May 2006.
- [18] Freepastry. <http://freepastry.rice.edu/>.
- [19] A. Ganesh, A. Kermarrec, and L. Massoulié. Scamp: Peer-to-peer lightweight membership service for large-scale group communication. In *Proc. NGC'2001*, Nov. 2001.
- [20] V. Goyal. Multiple description coding: Compression meets the network. *IEEE Signal Processing Magazine*, 18(5):74–93, Sept. 2001.
- [21] K. Gummadi, R. Gummadi, S. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The impact of DHT routing geometry on resilience and proximity. In *Proc. ACM SIGCOMM 2003*, Aug. 2003.
- [22] K. Gummadi, S. Saroiu, and S. Gribble. King: Estimating latency between arbitrary internet end hosts. In *Proc. ACM SIGCOMM IMW*, Nov. 2002.
- [23] X. Hei, C. Liang, J. Liang, Y. Liu, and K. Ross. Insights into PPLive: A measurement study of a large-scale P2P IPTV system. In *Proc. of Workshop on Internet Protocol TV(IPTV) services over World Wide Web(WWW'06)*, May 2006.
- [24] K. Hildrum, J. Kubiawicz, S. Rao, and B. Zhao. Distributed object location in dynamic network. In *Theory of Computing Systems*, Springer verlag, Mar. 2004.
- [25] J. Jannotti, D. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O. Jr. Overcast: Reliable multicasting with an overlay network. In *Proc. of OSDI 2000*, Oct. 2000.
- [26] D. Katabi and J. Wroclawski. A framework for scalable global IP-Anycast (GIA). In *Proc. ACM SIGCOMM'00*, 2000.
- [27] D. Kostic, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High bandwidth data dissemination using an overlay mesh. In *Proc. of SOSIP*, 2003.
- [28] T. Ng and H. Zhang. A network positioning system for the internet. In *Proc. of USENIX Annual Technical Conference*, 2004.
- [29] V. Padmanabhan, H. Wang, P. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proc. of NOSSDAV*, May 2002.
- [30] V. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. Mohr. Chainsaw: Eliminating trees from overlay multicast. In *Proc. of IPTPS 2005*, Feb 2005.
- [31] C. Partridge, T. Mendez, and W. Milliken. RFC 1546: Host anycasting service, Nov. 1993.
- [32] P. Pietzuch, J. Shneidman, J. Ledlie, M. Welsh, M. Seltzer, and M. Roussopoulos. Evaluating DHT-based service placement for stream-based overlays. In *Proc. IPTPS '05*, Feb. 2005.
- [33] Planetlab. <http://www.planet-lab.org/>.
- [34] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content-addressable network. In *Proc. ACM SIGCOMM'01*, Aug. 2001.
- [35] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proc. of NGC*, 2001.
- [36] R. Renesse, K. Birman, and W. Vogels. Astrolabe: A robust and scalable technology for distributed system monitoring, management and data mining. *ACM TOCS*, 21(2):164–206, May 2003.
- [37] S. Rhea, D. Geels, T. Roscoe, and J. Kubiawicz. Handling churn in a DHT. In *Proc. of USENIX Annual Technical Conference*, 2004.
- [38] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. In *IFIP/ACM Middleware 2001*, Nov. 2001.
- [39] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application end-points. In *Proc. of SIGCOMM 2004*, 2004.
- [40] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *SIGCOMM'2002*, Aug. 2002.
- [41] I. Stoica, R. Morris, D. Karger, M. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. ACM SIGCOMM'01*, Aug. 2001.
- [42] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostic, J. Chase, and D. Becker. Scalability and accuracy in a large-scale network emulator. In *Proc. of OSDI 2002*, Dec. 2002.
- [43] V. Venkataraman, P. Francis, and J. Calandrino. Chunkspread: Multi-tree unstructured peer-to-peer multicast. In *Proc. IPTPS '06*, Feb. 2006.
- [44] V. Vishnumurthy and P. Francis. On heterogeneous overlay construction and random node selection in unstructured p2p networks. In *Proc. INFOCOM 2006*, April 2006.
- [45] L. Wang, V. Pai, and L. Peterson. The effectiveness of request redirection on CDN robustness. In *Proc. of OSDI 2002*, Dec. 2002.
- [46] W. Wang, D. Helder, S. Jamin, and L. Zhang. Overlay optimizations for end-host multicast. In *Proc. of 4th Intl. Workshop on Networked Group Communication (NGC)*, Oct. 2002.
- [47] P. Yalagandula and M. Dahlin. A scalable distributed information management system. In *Proc. ACM SIGCOMM 2004*, Aug. 2004.
- [48] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proc. of INFOCOM 2002*, 2002.
- [49] X. Zhang, J. Liu, B. Li, and T. Yum. Donet: A data-driven overlay network for efficient live media streaming. In *Proc. INFOCOM 2005*, March 2005.