2010

Revisiting Overlay Multicasting for the Cloud

Karthik Nagaraj
Purdue University, Knagara@cs.purdue.edu

Hitesh Khandelwal
Purdue University, hkande@cs.purdue.edu

Charles Killian
Purdue University, ckillian@cs.purdue.edu

Ramana Rao Kompella
Purdue University, kompella@cs.purdue.edu

Report Number:
10-011

Revisiting Overlay Multicasting for the Cloud

Karthik Nagaraj, Hitesh Khandelwal, Charles Killian, Ramana Rao Kompella
Purdue University
{*knagara, hkhande, ckillian, kompella*}*@cs.purdue.edu*

Abstract

Group communication systems (e.g., multicast, DHTs) have emerged as basic primitives for several large-scale distributed systems. Most existing systems that implement these primitives often assume a flat topology of overlay nodes. For instance, many DHTs assume that all overlay links are often homogeneous in their capacities, costs and other such characteristics. Similarly, multicast protocols create overlay trees without taking into account the physical location of nodes. While a few systems do consider latency between different overlay nodes, the fact that metrics such as latency change continuously often translates into additional complexity in constantly measuring and reorganizing nodes. Modern trends in hierarchical data center designs and global services running across several geographically disparate data centers pose unique challenges and opportunity to revisit the design of group communication systems with location awareness in-built into these systems from the ground up. In this paper, we present the design and architecture of one such system called DC2, in which nodes are aware of their location within the data center (e.g., rack, aggregation switch) and organize group communications in order to minimize the expensive cross-data center links or cross-hierarchy links as much as possible. In our experiments using a real prototype deployed over 700 virtual nodes running over 15 physical machines, we found that DC2 minimizes message latencies by several orders of magnitude, and reduces node and link stress by a factor of 2 to 3 $\times$.

1 Introduction

Group communication systems (e.g., multicast, DHTs) are essential building blocks of many large-scale distributed systems. For instance, publish-subscribe systems (e.g., Open Publish Subscribe [2]) use multicast mechanisms as effective tools to deliver content to a group of end users. Similarly, distributed hash tables (DHTs) (e.g., Chord [18], Pastry [16]) that have gained significant prominence in the recent times facilitate easy object sharing (e.g., file sharing) across many different users. Given their importance, a lot of research (e.g. [4, 14–16, 18, 21]) has gone into building efficient group communication systems that scale to a large number of nodes.

Most proposed group communication systems (e.g., Scribe [4], Pastry [16]) assume a flat topology of overlay nodes. In other words, these systems do not make any assumptions about where in the physical network these overlay nodes are present. Thus, these systems are designed to minimize the number of overlay links used, assuming that all links are homogeneous in their capacity, bandwidth and even costs. In some cases, they support a modest amount of latency-aware opportunistic neighbor selection, but this is a secondary consideration only. While this assumption is fine in a general context, there are several emerging deployment scenarios where this assumption does not hold. We describe two such instances that differ in their context, but are similar in spirit.

In our first example, consider a set of nodes within a modern data center that implement a group communication system. Most existing data centers use a 'scale-up' architecture. In this architecture, racks of machines are connected to top-of-rack (ToR) switches that are subsequently connected to aggregation and core switches. Providing full bisection bandwidth between any pair of nodes requires very costly redundancy and oversubscription. As a result, most existing data centers run with limited oversubscription ratios of about 240:1 or 80:1. While researchers have begun proposing new 'scale-out' architectures such as Portland [12], VL2 [6], and BCube [7], existing data centers still are predominantly in the 'scale-up' category. In a 'scale-up' architecture, overlay links between various data center nodes are not homogeneous; the links that cross many hierarchies (e.g.,

aggregation, core) typically tend to be more expensive than the ones that are local within say a rack.

Our other example concerns services deployed on a global basis spanning many different data centers. As cloud services continue to grow quickly with users in many different geographic domains, such group communication systems that span many different data centers will gain more prominence. In these systems, the links within a data center typically cost much less to access than the links that cross data centers, Even when the service is being hosted by a single entity (e.g., Amazon), it is quite expensive to move data across data centers in different availability zones. Amazon EC2, for instance, charges about 19c per GB transferred across cross-continental data centers, while within US or EU regions, it only charges 1c per GB [1]. Thus, overlay link costs are not homogeneous and depend a lot on whether the overlay link lies entirely within a data center or spans multiple of them.

Given the heterogeneity in the link costs and bandwidths, it is important to revisit the design of group communication systems to incorporate some notion of location- and/or cost- awareness. This awareness helps keep communication costs low and reduce the stress on expensive links as much as possible. In this paper, we present one such location-aware group communication system called DC2 (short for data-center aware distributed communication). DC2 can avoid wasteful extra link traffic within a data center, or expensive cross-data-center links, by specifically seeking out topologically nearby peers with which to join the group overlay used for data dissemination and location.

The basic design idea of keeping communication local is by itself not a novel idea. Indeed, systems such as AMMO [15] optimize multicast trees using metrics such as latency and so on. DC2 differs from such systems in two ways. First, DC2 relies on making nodes aware of their position within the hierarchy. Hierarchy could be as simple as knowing whether two nodes lie within the same data center or across data centers. It can also be as complex as two nodes knowing they exist within same rack, same aggregation switch, same core switch and many more such levels. Second, DC2 facilitates incorporating different costs of operating different overlay links which are not easily discovered in a system such as AMMO. Besides, AMMO is an overkill for the task at hand since it tries to compute least latency paths; since latency keeps changing over time, it is continuously estimated and shortest paths are recomputed repeatedly making it quite complicated.

In this paper, we demonstrate the effectiveness of DC2 using two instances of location-aware group communication systems. The first is based on DHT protocol design, while the second is a ground-up design of a replacement mechanism. In doing so, we also provide an evaluation of competing design principles and their strengths and weaknesses. In our experiments using a real prototype deployed over 700 virtual nodes running over 15 physical machines, we found that DC2 minimizes message latencies by several orders of magnitude, and reduces node and link stress by a factor of 2 to 3 $\times$.

The remainder of the paper is organized as follows. Section 2 describes background in overlay trees and DHT routing necessary to understanding the DC2 design presented in Section 3. Our implementation is described in Section 4, and our experimental setup and evaluation in Section 5. We then detail closely related work in Section 6 before concluding in Section 7.

2 Background

Before diving into the DC2 design, we first give background on the Scribe publish-subscribe design we build on top of, focusing on the relevant aspects necessary to understand our design decisions. A treatment of the complete related work can be found in Section 6.

The core functionality of DC2 is to build location aware overlay trees for group members, that minimizes the number of links that cross from one hierarchical grouping to another. As such, it provides a mechanism for group members to share and spread information with the least amount of group cost.

2.1 Overlay trees

To connect together group members, we will be building overlay trees spanning the group members. An overlay tree is a logical entity, connecting tree member endhosts by having each member track its parent and children, independently of the underlying network structure. With an overlay tree, we can support a flexible range of data communication options. At one end of the spectrum, messaging can be pushed along tree edges, providing multicast to participants. At the other end, data can be stored at the root of the tree, and can be pulled toward querying nodes, potentially being cached along the way. In the middle, trees could also support pushing data to a subset of tree members, effectively pre-loading some caches for performance of later lookups.

2.2 DHT routing

One common mechanism for supporting group and data location in a peer-to-peer distributed system is to use the consistent hashing provided by typical distributed hash tables (DHTs). DHTs such as Chord [18], Pastry [16], and Bamboo [14] assemble overlay participants

into a ring, assigning each a portion of key-space to manage. Then, to support data location, group identifiers are mapped to the key-space using consistent hashing. The consistent hashes ensure that all nodes find a common group "owner," namely the node to which the group identifier maps to. Additionally, consistent hashing provides load-balancing, and ensures that node churn affects only a small degree of existing keys.

Pastry, Chord, and Bamboo are all common in providing location of a key owner in a number of hops logarithmic in their distance. These services also tolerate node churn, and scale to a large number of nodes. At each node, a node can respond to a query for the next hop toward a destination, returning a node an order of magnitude closer in the key-space to the destination, while storing only $O(\lg n)$ state for n system nodes. The differences in the protocols are based on specific decisions such as defining nearness, proximity neighbor selection, and how to handle node churn. Proximity neighbor selection (PNS) is the idea that at each hop, among all nodes which are an order of magnitude closer to the destination in the key-space, we should select the node with the shortest physical distance or latency.

2.3 Scribe

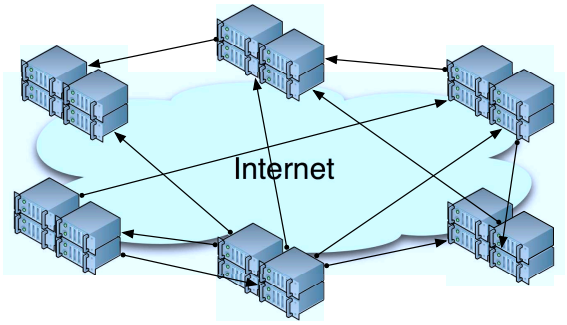
Scribe [4] is a group service designed to support group-based multicast over overlay trees. Nodes join a group by routing a Join message to the owner of the key a group identifier hashes to. Scribe builds the reverse-path forwarding tree, by having each intermediate node along the path join the tree itself, and accepting as a child any node it processes a join request for. Effectively, this is the *same path* a query in the underlying DHT would take.

While Scribe was originally designed to run over Pastry, in our evaluation, we use Scribe over an implementation of Bamboo. This decision reflects the fact that Bamboo is an extension and modification of Pastry, and in our tests, Scribe performed better over Bamboo than over Pastry due to better handling of churn, and lower initialization costs.

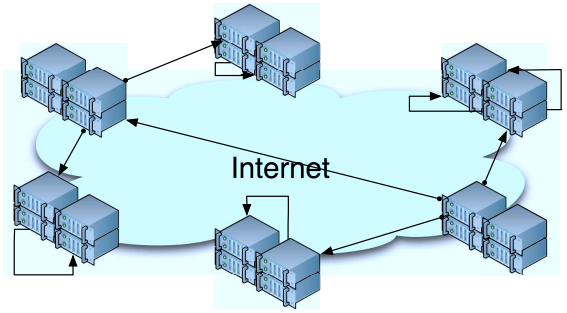
Scribe is designed to be location-aware in that by using the DHT routing with PNS, you expect each hop to be to a node which is physically close to the previous hop. However, since there are a limited number of possible options, our results demonstrate that overall, PNS is not sufficient to build efficient group trees in the data center environment.

3 Design

Insufficient location awareness of the existing tree construction schemes motivated our work for building Datacenter location-aware trees. Figure 3 shows the nature of trees that are constructed by Scribe against those where the location information is specifically induced. The number of tree edge links for a particular tree are shown, which can potentially be large for Scribe. However a location-aware scheme such as DC2 would merge trees in a hierarchical domain first, in this case racks and then proceed to combine the trees between all racks. We first describe the location information that must be gathered for each participant in order to design a good tree construction protocol. We suggest some modifications to be performed on Scribe which can leverage location efficiently in constructing trees. We then design a new and improved protocol, DC2, which overcomes limitations of Scribe and its modified version. We detail the design of DC2 along with the decisions made for improving its performance.



(a) Scribe trees across data centers



(b) Location-aware trees

Figure 1: Comparison of multicast trees constructed across nodes in multiple data centers

3.1 Location inference

Location plays an important part in our performance optimizations and improvements, as it allows us to make network-specific decisions. Location is an indirect attribute of the network topology that is available underneath. In a data center network, machines within a rack are separated by a single rack switch and machines on

different racks need to communicate through higher level switches. Over-subscription of higher layer switches is typical to current data center designs, that makes the network performance dependent on traffic and congestion at core routers. Similarly, localizing traffic inside a data center is essential in improving network throughput avoiding cost incurred on leased WAN links.

Ideally, we would like to maximize traffic within a rack and minimize traffic between racks. Similarly, at a higher hierarchy, we would like to maximize traffic within a data center and minimize traffic across data centers. We would like to identify location parameters for each participant of our system which identify its location at each hierarchical level. In the data center environment, this is identifying the data center hosting the participant, and the identification of the rack within.

In a managed data center, we can assume the presence of a single authority such as system administrators who are aware of the placement of various machines inside the facility. This enables the construction of a simple numbering and lookup mechanism to tag and lookup every machine to the location parameters. The Internet IP address allocated to a node can be used to identify the various subnets that it belongs. If it is known that a single data center is given a single subnet block and similarly each rack given a smaller subnet block, we can directly use the subnet values to infer the location parameters.

Inside cloud hosting platforms such as Amazon EC2, Microsoft Azure, etc., users are exposed to resources through virtual machines. Some of the coarse location details such as region of the country may be given out, but the actual location of data centers and VM placement in a rack inside a data center is kept transparent. This makes it hard for us to make assumptions on the availability of the location parameters. However, it might be possible to use some latency and bandwidth estimations against known thresholds to make predictions and suitably define the location parameters. We leave this as future work.

We assume that each participant host is aware of its location parameters when joining the system. This is not an unreasonable assumption for several simple hierarchies, nodes can easily identify which data center they belong to by say looking at the IP address allocated to that node (different data centers often have different IP subnet prefixes). By a similar token, racks or other forms of hierarchy may be easily deciphered using a similar mechanism. From the location parameters provided (or inferred), we assume that all clusters in a hierarchy are uniquely numbered within that hierarchy, i.e., all data centers throughout the world are given unique identifiers, and all racks within a data center are numbered uniquely within that data center. Such a unique numbering scheme also enables us to fix the number of bits required to represent

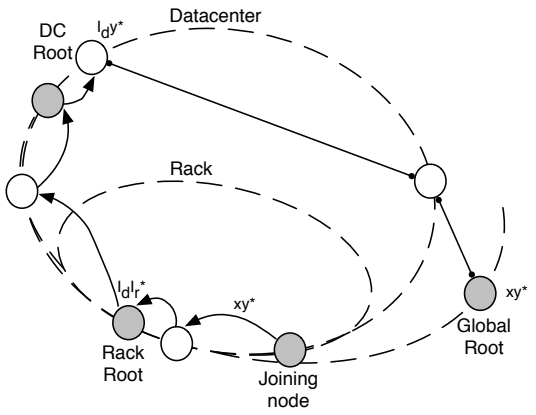


Figure 2: Modified Scribe tree construction

these identifiers. Let us represent the number of bits required for data center identifiers as B_d, and the number of bits required for rack identifiers as B_r.

Each host is assigned a unique 160-bit identifier, which is computed usually as a SHA-1 hash of the local IP address and port, similar to previous overlay rings [14]. The hosts are present in a logical ring with their position determined by their hash. Each host extracts its unique data center identifier I_d and its unique rack identifier I_r from its location parameters using a common well known mechanism. These identifiers are induced into the node hash to build protocols that can seamlessly take advantage of the location knowledge. This is followed by setting the first B_d bits of the node hash to I_d, and the next B_r bits to I_r.

Bamboo overlay construction is run on all the hosts in the system across multiple data centers with this hash assignment scheme. The hosts in the same rack will be neighbors on the ring forming a contiguous sequence. The hosts from multiple racks in the same data center will group together to form a larger contiguous sequence. The key distance between two hashes is defined by Pastry and Bamboo as the absolute minimum difference of the hashes in the logical ring. It is significant to note that key distance is also a direct implication of closeness of the hosts. Any two hosts within the same data center would share the same prefix of B_d bits and their positions on the logical ring would be close.

3.2 Modified Scribe

We describe our modifications to Scribe to improve the performance in the availability of location information. This modification benefits mainly from the changes to node hash, which implicitly allows Scribe to cluster tree edges to within location boundaries. We describe the

modifications using an example shown in Fig. 2. Consider a node joining a group which has the topic key as $xy*$ where x,y represent the first B_d, B_r bits of the key followed by the rest of the bits ($*$ is a wildcard). The host substitutes its own values of data center and rack identifier for the respective positions in the topic hash, to get the key I_dI_r*. We expect that touting this join on the Bamboo overlay will terminate at a *Rack Root* within the same rack as the initiating host (shown by the arrows tracing the path). All hosts within the same rack attempting to join the same group would contact this same rendezvous node. The same Scribe protocol is used in constructing a tree under the *Rack Root*.

Since the *Rack Root* is not already a part of the tree, the join process continues by substituting the value of data center identifier for the respective position in the group hash: I_dy*, and routing to this key. Similarly this join request would collect for all hosts in this data center at a single host denoted by *DC Root*. At this point, we route the join request to the plain group key $xy*$ which is the global root for this Multicast tree. In this example, since the *DC Root* was already joined under the *Global Root* for this group, this particular Join request terminates at the *DC Root* (this is shown as solid constructed tree edges).

The trees constructed by forcing the overlay routing to go through the selected hierarchical rendezvous hosts gives these trees the desirable properties. Within a rack, which is the lowest hierarchical level we have all the subscribed hosts have their path to root pass through a single host which is the *Rack Root*. The intuition behind this key is to randomly pick a common node within the same rack to form a local tree. This property constructs a local tree rooted at this node that serves all the subscribers within the rack. Thus there would be a single tree edge entering/leaving the rack, and all nodes in the rack would distribute content among themselves by limiting traffic to within the rack. Similarly, all hosts in the data center would construct a tree rooted at the *DC Root* (which manages the key I_dy*). This phase attempts to serve all the nodes within this data center using a tree that is completely local to the data center. This attempts to reduce the traffic that is ingress/egress out of the data center.

Discussion. We evaluated this modified implementation of Scribe to study its properties. We see that this inherits the scalability of Scribe and has the capability to scale to large number of groups and large group sizes as well. The relative simplicity of Scribe by benefiting from the efficiency and robustness of Pastry allows for this scaling. But, one of the most important problems of Scribe is that it involves nodes that are not subscribers as interior forwarders of the tree. Figure 2 shows the nodes shaded as the actual subscribers to the tree and the unshaded nodes being just forwarders. These nodes do not benefit themselves in any way from this forwarding, but are equally sharing work in the peer-to-peer network. This affects the performance of the tree in multiple ways: *(i)* The tree consists of more members than subscribers, and hence, the height of the tree increases; *(ii)* Members of the tree which do not subscribe to the tree have to contribute toward processing traffic and network bandwidth. It might also be the case that when we pick the local root hash for our locality, the node managing the hash might actually be outside, in the next group. This increases the number of tree edges that cross the locality boundary, thus leading to increased costs.

3.3 DC2: Data center aware distributed communication

From the modified Scribe implementation, we realized the incoherence between the requirements of tree construction and the overlay routing properties of Pastry/Bamboo. Also the tight dependence of the tree edges on the overlay route provided leads to unsatisfactory performance. We propose an efficient tree construction mechanism that mitigates each of the issues described above, while still scaling to large groups.

By hierarchically organizing nodes into clusters in a data center environment, we infer that the number of unique entities within a single hierarchical level is limited to a couple of hundred. More specifically, the number of machines that can be present in a single rack is about 20-40. Even if they are hosting multiple Virtual Machines (each of which would be a unique entity), the total number of nodes is bounded. At the next hierarchical level, the number of racks within a data center is also bounded by physical constraints. At the topmost order, we know that the total number of data centers around the world is limited, again bounding the number of entities (data centers). We can thus construct a hierarchical tree of trees by using some simpler tree construction techniques that scale to the limited number of entities at each level. It is also easier to handle and optimize smaller trees without violating any minimum cost constraints.

We explain the tree construction in DC2 using Fig. 3.3. Each cluster of nodes in a hierarchy initially identify a cluster coordinator, which is another node identified in the overlay ring. Consider a host in a data center with its overlay hash key being $xy*$ as before. The *Rack Coordinator* is identified as the node which manages the key I_dI_r* in the overlay ring. The coordinator provides a lookup mechanism for nodes in the cluster to the local rack root. Thus each group is randomly, yet consistently coordinated by one of the nodes in the cluster. A node joining a group routes the lookup in the overlay ring to the coordinator, and hears back with the identity of the

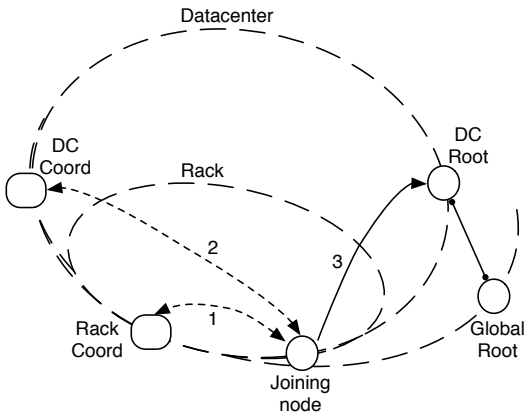


Figure 3: DC2 tree construction

rack root (as shown by arrow 1).

If the coordinator identifies a request for a new group from a node, it assigns the contacting node to be the local root of the group and notifies that node. In this example, the joining node is the first node in this rack joining this group and is hence elected to be the Rack root. This approach aims at reducing the members of the tree by forcing the root of the tree to be an actual subscriber. This decision indeed puts the burden of the root node to the first subscriber of the group within this cluster. We discuss ways of alleviating node stress and techniques for load balancing.

As soon as a node is assigned to be the local root of a cluster, it becomes responsible for bridging the local tree to the global tree for the group. For joining the upper level, this node contacts the *DC coordinator* in a similar mechanism as described above (shown in line 2). The coordinator for the upper hierarchy (data center) is identified as the node owning the key I_dy*. By the structure of key assignment to nodes, this coordinator will be common to all nodes in the same data center. Each Rack root for this group would communicate with this coordinator for forming the tree of racks. In this example, the *DC Coord* notifies the joining node of the existence of the *DC Root* in message 2. The joining node contacts the *DC Root* in message 3 for joining the tree. The final part of this process proceeds by the data center roots routing to the global coordinator for the group (node managing the key $xy*$). However, in our case the *DC Root* has already maintained connection with a *Global Root* (shown by the solid line) and this step is not needed.

RandTree. We have already provisioned for nodes in a cluster to identify and contact a single root for a group. We describe the process of tree creation using the RandTree protocol. Once, a node is picked as the root of the tree, it stays the root until told otherwise by the coordinator. Any node wishing to participate in the tree contacts the root. The root checks to see if it has space to take one more child and becomes a parent of the new node if it does. Each node is allowed to maximum number of children defined to be a constant for the system (can this be group/processing capacity dependent?). When the root notices that it is unable to take more children, it forwards the join to a random child. This process continues until the node is accepted as a child at come level. This process is guaranteed to terminate if every node can parent at least one child (for each group).

One of the key improvements in our technique is participation of only the subscribers in the global tree for a group. Although we involve coordinator nodes picked randomly, they are passively involved in the tree construction and do not contribute resources toward data transfer. This globally reduces the processing needed at all nodes to multicast a message, and also the network bandwidth consumed. The global reduction directly translates to overall local gains for all participants.

We described the tree construction at individual levels of the hierarchy to use a random tree construction protocol because the performance (latency and bandwidth) guarantees is usually similar between any pair of nodes within the same hierarchy. Thus our performance is not expected to be deviating too much from the ideal. Also, once we have *some* tree constructed between the nodes, it is always possible to optimize such trees for any number of desirable properties. Adapting overlay trees based on network measurement has already been studied well in literature. AMMO [15] describes a mechanism to optimize Multicast trees using multiple metrics. An important property of our technique is that each level of the hierarchy is logically separated at the nodes allowing for independent optimizations to levels. The hierarchical organization also reduces the members of the tree in each hierarchy (as discussed earlier) and thus allow for optimizations on smaller trees. This means that we could individually optimize for the topmost level of the hierarchy; between data centers, bandwidth is scarce and latencies are high, exposing opportunities for improvement.

Leaving a Group. When a node wishes to unsubscribe from a group, it gracefully leaves the tree for the group. This is quite simple for a non-root member of the tree by sending Leave messages to the parent and all children. When the child detects that its parent has left, it would proceed to contact its local cluster root to join the tree again for the group. When an intermediate root node needs to leave a tree, it sends out a notification to the coordinator to stop pointing to this node. It also asks its children to leave as above, who would now contact the coordinator again for a root. This mapping at the coordinator is refreshed for the group to point to the first node that contacts it. As the group membership changes, the

trees are continually adapted to contain only subscriber nodes. The tree updates messages are always contained within the lowest hierarchical cluster except in the case of intermediate root nodes. When an intermediate root node leaves the group, all trees heights which contain this node must be updated, i.e.,. a number of successive levels from the bottom level to the highest level to which this node belongs. The following section discusses instances where a node leaves the system on crash stop failure.

3.3.1 Handling Failures

Each node exchanges Heartbeat messages continuously with every node who is either a parent or child in any of the participating groups. A continued silence from a peer on the tree is considered to be a failure. The frequency of heartbeats can be tuned for the deployment setting and importance of speedy tree recovery. We outline the values used for our experiments in the Evaluation section.

Group Member failures. If the parent of a node died, it is orphaned from the global tree and seeks to join under a new parent. As described above, the node sends a Join request again to the identified root. This node is already aware of the root of the tree which was identified during the initial join procedure. If the failure of the root is detected, the coordinator for the group at this level is again contacted for a lookup to the current root of the tree. The random tree building protocol is repeated with this new root. A change in the root of the tree signals an update message indicating the new root to be sent down the tree. This update is forwarded by all nodes to their children until the leaves.

Coordinator failures. The root keeps contacting the coordinator to keep its state alive. When it detects the coordinator failure, it informs a new coordinator of the existence of this group and its own identity as the root. A new coordinator will be picked by Bamboo if the earlier coordinator fails.

4 Implementation

We built prototype implementations of modified Scribe and DC2 in Mace [9]. Mace is a toolkit for building systems by writing C++ code in an event-driven programming, and specifying system as layered services. We reused the previously implemented Bamboo and Scribe services. The layering of these services on top of each other and eventually over TCP transport is shown in Fig 4.

Our implementation of Bamboo is used in all services requiring overlay routing, providing the same functionality as Pastry with better handling of churn. Bamboo directly interacts with other peers using TCP. The default implementation of Scribe as described in [4] is run over

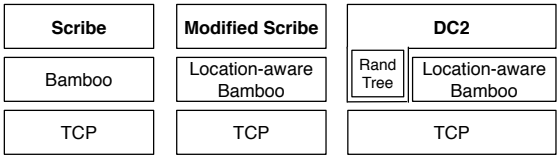


Figure 4: Layering of services

Pastry, as against our experiments which use Bamboo. The modified Scribe implementation is location aware in terms of both its underlying routing mechanism, and hierarchical building of the tree. Hence, we use our adjusted location-aware version of Bamboo to achieve this. This version primarily differs in the way a node constructs its own key hash, by forcing the hierarchy bits into the hash after computing a random hash as described in Section 3.1. Modified Scribe is layered similar to Scribe and constructs the hierarchical trees by routing over the location-aware nodes.

The DC2 service utilizes the overlay routing functionality of location-aware Bamboo to identify and communicate with coordinators at each level of the hierarchy. Once the coordinator points a node to the root of the Multicast tree, it is unnecessary to depend on the overlay routing (that Pastry or Bamboo provides) for tree construction. Unlike Scribe trees which rely on Bamboo routing to identify tree edges, DC2 constructs trees using the RandTree protocol which runs directly over TCP. DC2 however needs location-aware Bamboo to find the coordinator. Figure 4 illustrates this layering of DC2 over TCP making use of both location-aware Bamboo as well as RandTree that is implemented as part of DC2.

Our implementation of modified Scribe requires changes to the previous implementation of Scribe by about 250 lines of C++ code in Mace. The implementation of the DC2 protocol including the integrated Random Tree construction protocol constitutes about 800 lines of C++ code in Mace.

All of these various implementations, namely, Scribe, modified Scribe and DC2, provide the same basic abstraction of *group trees*. Group tree abstraction allows the upper layer to specify group subscriptions based on a group key. The upper layers can query for the parent and children in the tree for the respective group by providing a group key. The multicast application that we use to test the multicast functionality and performance of the protocols is layered over the group tree abstraction. It takes in a *subscriptions* file to identify the groups it should join during a run of the experiment and issues 'subscribe' requests over all these groups. The application also takes in a *schedule* file that identifies the communication involved in the groups. This schedule identifies a time along with

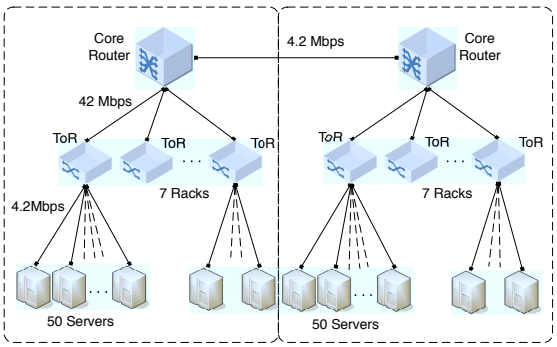
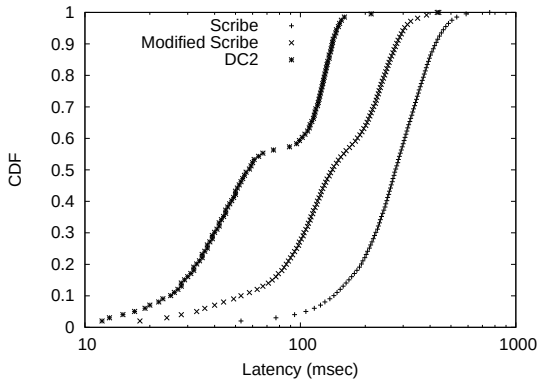
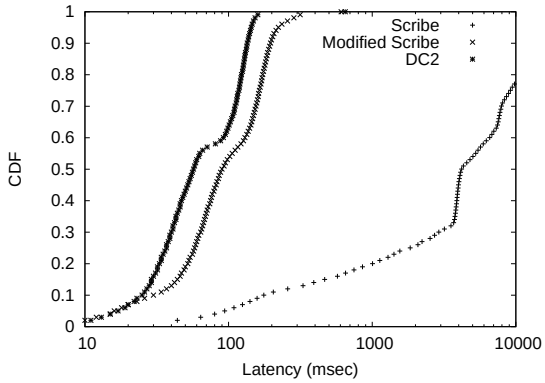


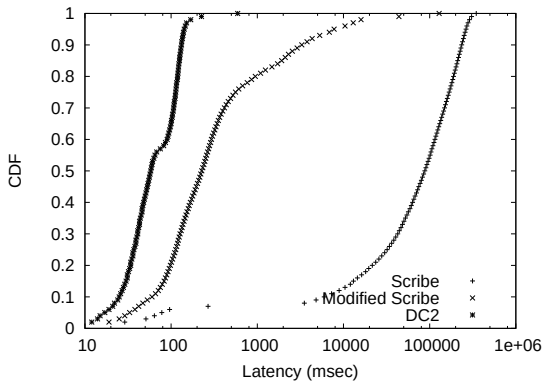
Figure 5: Hierarchical Datacenter Topology with 7 racks each in 2 datacenters



(a) 2 pkts/sec



(b) 10 pkts/sec



(c) 50 pkts/sec

Figure 6: CDF of latency for different packet injection rates

the node that should multicast a message in a specified group (it is assumed that the node already subscribes to this group). The use of the subscription and schedule files enables us to easily abstract and reproduce our experiments.

5 Evaluation

In this section, we compare Scribe, modified Scribe and DC2 across various performance metrics such as link and node stress, end-to-end latency and so on. Before we describe our results, we first explain the topology and other details of our experimental setup.

5.1 Experimental setup

Our experimental testbed consists of a cluster of 15 machines, each equipped with 8-core 2.4 GHz processors with 8GB of RAM running Gentoo Linux 2.6.32.3. All the 15 machines are connected through a switch using 3 Gbps links (3 x 1Gbps interfaces bundled together). We emulate a large number of nodes for our experiments by running multiple processes on a single machine, with each machine hosting the same number of nodes for load balancing purposes.

We compare the 3 tree building protocols Scribe, Modified Scribe and DC2 on different testbed scenarios. We used Modelnet [19] to emulate a hierarchical network topology (similar to a typical data center, except with fewer number of levels in the hierarchy) as shown in Fig 5. This emulated topology consists of two datacenters with seven racks each, with each rack consisting of 50 machines, giving a total of 350 virtual nodes per data center, and a total of 700 nodes overall.

Modelnet [19] emulates any virtual topology inside a single node in the testbed. A topology file listing the end hosts, switches and link properties is given to the emulator. All traffic for the distributed system is redirected to the emulator which emulates packet queueing, link latencies, etc. The traffic is then sent to the destination host where it is delivered at the appropriate virtual node. Diverting traffic to the Modelnet emulator is achieved by intercepting traffic at the kernel in the host nodes. Modelnet assigns the 10.0.0.0/24 subnet IP addresses to the virtual nodes for network emulation. In our testbed, we

have the first node to serve as the emulator while the other 14 nodes host the virtual nodes. Given the physical bandwidth limitation of 3 Gbps for the Modelnet core, each node cannot be assigned more than 4.2 Mbps to the virtual switch.

All the racks have a Top of Rack (ToR) switch connected to all hosts in the rack using 4.2 Mbps links (corresponding to the maximum physical bandwidth that can be assigned) with 1 ms latency. The ToR switches are all connected to the core router for the data center on 42 Mbps links with 2 ms latency emulating the next level of hierarchy found in a typical data center. The core routers are connected by a 4.2 Mbps, 50 ms link in order to emulate a high latency cross-data-center link. The link bandwidths are assigned to follow the trend which is similar to typical 'scale-up' data center topologies, i.e., the link bandwidth of the ToR-Core router link is about 10 $\times$ more than the end host link. We assume that the inter-data-center link carries the same bandwidth as the end host link, although with much higher latency (of 50 ms) typical of WAN links.

5.2 Multicast performance

For studying the performance benefits of DC2, we run experiments on these 700 nodes using Modelnet assuming a stable topology without node failures. At the beginning of each experiment, all the virtual nodes are brought up on the 14 machines and given sufficient time to stabilize the overlay ring. Each of the nodes proceeds to subscribe to all of its groups and waits till the trees have finished building and stabilized. At this point the schedule file is used to deterministically generate traffic in the system.

The group sizes follow a Zipf-ian distribution with minimum, maximum, average group sizes to be 25, 150 and 75 respectively with a random assignment of nodes to groups. The number of groups is also 700 with a single identified node being the source per group. For these experiments, the group memberships remain constant throughout the experiment. The offered Multicast traffic is uniform throughout the experiment with a fixed packet size of 1KB. The rate of packet generation is fixed for the whole system, and random nodes are picked to initiate multicast on their assigned groups. We have chosen these numbers somewhat arbitrarily for illustration purposes; real benefits of our system may to some extent depend on the particular deployment scenario.

5.2.1 Latency

Fig 6 shows the CDF of the latencies of all messages delivered at nodes (x-axis in logscale). We show the CDFs for different packet injection rates ranging from 2 pkts/sec to 50 pkts/sec. Note that the CDF is across all multicast groups, i.e., across messages delivered at all nodes within all the multicast groups for which messages are injected. We observe that when the system is lightly loaded, at 2 pkts/sec, the difference between DC2 and Scribe is the least compared to when the system is heavily loaded at 50 pkts/second. As the system observes higher packet injection rates, the latency experienced by nodes in Scribe is much higher compared to either modified Scribe or DC2. For instance, at 50 pkts/second, the median packet latency in Scribe is as high as 100 seconds compared to less than 100 ms for DC2 and 200 ms for modified Scribe, translating to several orders of magnitude improvement over the naive Scribe implementation.

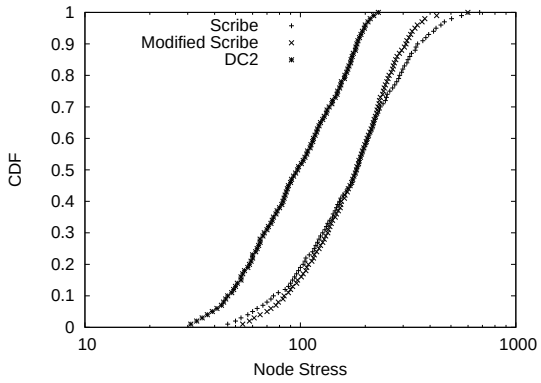
The improvement exhibited by both DC2 and modified Scribe can mainly be attributed to the reduced use of the inter-data-center link. This link alone contributes 50 ms of latency; by avoiding this link as much as possible, DC2 achieves better performance. Since the trees constructed by Scribe are not localized to within data centers, there could be multiple tree edges along a single root-to-leaf path which make use of the inter-data-center link. Our modified Scribe performs much better than Scribe because of its location awareness. DC2 performs better than even modified Scribe because of the tree construction over overlay paths in modified Scribe involves non-subscriber intermediate nodes in the tree which in turn leads to skinnier trees.

As the traffic rate increases, we can observe a significant shift in the latencies when we use Scribe. Both the median and maximum latencies of Scribe grow as traffic increases because, many of the Scribe tree edges are actually mapped onto the inter-data-center link. Since this link is a bottleneck for communication, this link is easily flooded with traffic from Scribe trees. However, DC2 only constructs minimal overlay tree links that overlap with the inter-datacenter link (and also links between racks). Therefore, DC2 maintains the median latency of 50 ms which is close to the actual latency between the datacenters indicating least communication (at most once in many cases) between the datacenters.

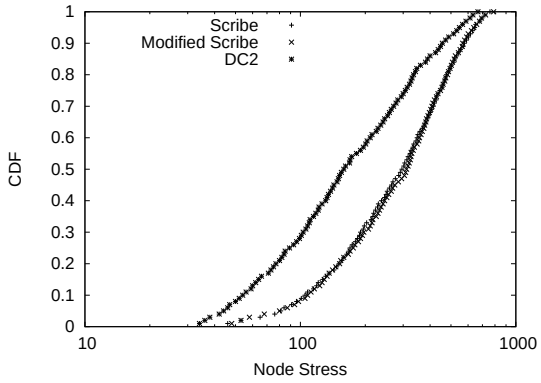
5.2.2 Node and Link stress

Low latency is nice, but it is also important to understand how DC2 affects *node-* and *link-stress* as they directly relate to how congested nodes are, and usage on individual links. We define node stress as the total number of messages that a node receives (from a parent or child in any of the trees in which it belongs). Link stress is defined similarly as the total number of messages sent on a link.

The experimental results for node stress on two different graphs is shown in Fig 7, where we show a CDF



(a) Min, Max, Avg degree = 25, 150, 75



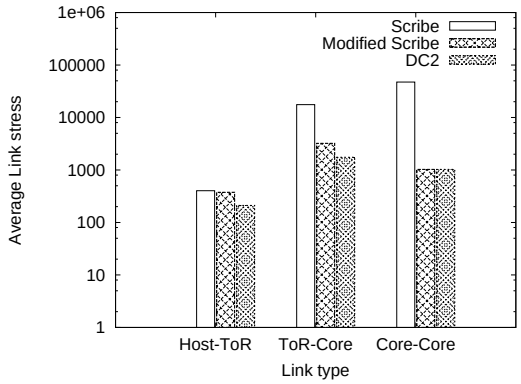
(b) Min, Max, Avg degree = 25, 500, 150

Figure 7: CDF of Node Stress

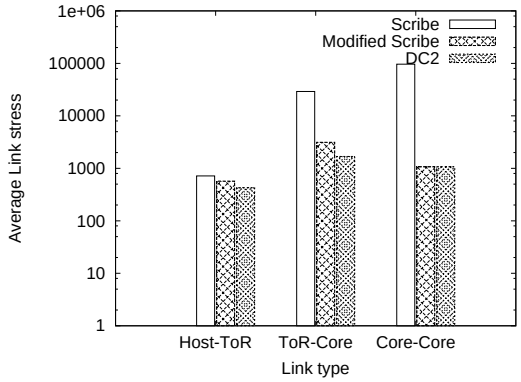
of node stresses on all 700 nodes (note that the x-axis in logscale). Fig 7(a) was evaluated for the standard subscription group sizes defined before. Given node stress directly translates to the number of packets that should be handled by a node, lesser node stress implies lesser network traffic and lesser load on the node. DC2 shows a node stress that is half of the Scribe variants at the median, while the maximum is almost $3\times$ lesser for DC2. The reason behind lesser node stress is the property of DC2 to involve just the subscribers of the group in the tree. Thus any node would only receive messages (and hence forward) only on the trees it is subscribed. It should be noted that the number of nodes in a DC2 tree is equal to the number of subscribers to the tree.

Modified Scribe performs similar to Scribe because of the similar mechanism to construct tree edges and involve intermediate nodes in the path to root. On a closer observation, Scribe has a higher percentage of nodes exhibiting lower amounts of stress than modified Scribe, but these curves cross each other at around the median and Modified Scribe. At larger values, the trend reverses and stress on Scribe nodes is larger. We noticed, however, that on an average, nodes in Scribe exhibit larger stress.

Fig 7(b) shows the node stress in an experiment with larger subscriptions to nodes (following a power law with different parameters, 25, 500 and 150 as the min, max and average degree). The observations are similar to the previous experiment with DC2 maintaining a smaller node stress. The maximum values for all the systems, however, are quite similar, which can be explained by the difference between the number of nodes in the system (700) and the maximum size of a group (500). As the group size nears the total number of nodes, Scribe has more subscribers in its trees and its stress is closer to the ideal (similar to DC2).



(a) Min, Max, Avg degree = 25, 150, 75



(b) Min, Max, Avg degree = 25, 500, 150

Figure 8: Average Link Stress on the different Datacenter network links

Link stress. We measure link stress by logging the count of messages received from each peer. This data is used to identify the route in the Modelnet topology that should have been used for the packet. Since there exists a single unique path between any pair of end hosts in this topology, this measurement is precise. We have 3 types of links in this topology: host-ToR (end host and ToR

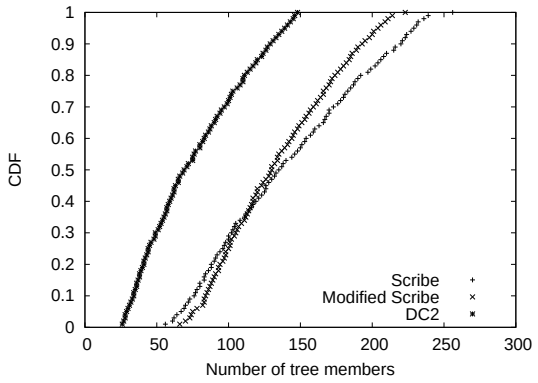


Figure 9: Number of Subscribers

switch), ToR-core (ToR switch and core router), core-core (inter-data-center link between core routers). The link stresses are accumulated through the experiment for the multicast messages and are plot as averages based on the link type in Fig 8 (note that the y-axis is in logscale).

The links are shown in increasing order of cost, traffic and hence importance. Thus lesser link stress on the core-core link implies lesser cost incurred on the WAN. DC2 and modified Scribe both equally reduce the link stress to the minimum. Intuitively, for this experiment generating 10 packets/sec for 100 sec is 1,000 packets which translates to an ideal link stress of 1,000. This is the same value that we observed leading to least cost and traffic. Scribe's notion of location is weak and limited, and hence, many of its tree edges coincide with the high latency links. DC2 performs better than both the Scribe variants for other links because of forcefully limiting the tree edges to within location bounds. Modified Scribe has the notion of locality, but its trees include many edges when a local group root tries to join the global tree for the group.

Fig 8(b) plots the link stress for the larger subscription groups (with 500 as the maximum group size). We see that the link stress worsens for Scribe because of an increase in the total tree edges, and hence, more chances of overlapping with a costlier link. These two plots show the capability of DC2 to scale with the node count and group sizes. The link stress remains dependent only on the number of location boundaries, i.e., the number of racks and number of datacenters that a group spans.

Number of tree members. In order to expose the actual number of nodes involved in the construction of trees (which was an important factor in the previous results), we plot the CDF of the tree size for each group in Fig 9. By construction, the curve for DC2 is ideal, with equal number of subscribers and tree members. Both Scribe and its modified version have almost 2 $\times$ more tree members than subscribers. Similar to the node stress figures earlier, Scribe and modified Scribe cross over each other. One curious observation is that Scribe trees have lesser tree members when the number of members of a group is small. This phenomenon is because modified Scribe is forced to route to multiple destinations before reaching the root, and hence, does not take the shortest path length. As group sizes are larger, the height of either trees (for Scribe and modified Scribe) increases, and also the overall node stress. Increasing the height of the tree would mean larger latencies for leaf nodes. In keeping the number of nodes in the tree to a minimum, DC2 aims at solving these problems and construct trees of lesser height, decrease node stress and decrease end-to-end latencies.

6 Related work

Many application level multicast systems have been proposed in the past literature [3–5, 8, 13]. Many of these systems attempt at constructing optimized platforms for multicast. Multicast in general is includes different underlying topologies such as Tree, Mesh, Hybrid, etc. based on the application requirements. However, we target application multicast using a tree that is optimized for end host performance.

Bayeux [21] is similar to Scribe [4] in constructing multicast trees over the Tapestry DHT. Our Modified Scribe implementation could also be tailored to suit such similar overlay tree construction mechanisms.

NICE [3] proposes a hierarchical overlay construction protocol by clustering nodes with lower end to end latency together. A source specific tree is constructed on top of the overlay for multicast. However, they do not discuss a scalable mechanism of constructing multiple independent trees. Also, these trees are constructed by latency measurements on the network which is susceptible to unoptimal clustering inside data centers.

AMMO [15] describes a mechanism to optimize constructed trees simultaneously for multiple parameters in a scalable manner. However, specifying optimization metrics that optimize for the data centers is hard and also do not guarantee link stress minimization. Also, there would be unnecessary cost of continually trying to optimize the tree safely. Our mechanism attempts to scale to large number of trees by minimizing the overhead in maintaining trees.

Kademlia [10] uses the XOR metric to construct routing tables and route to a given key. In our current implementation with Bamboo, it is possible that the search for a local coordinator ends a node outside our cluster (because the key is closer to a node in the next cluster). The use of a XOR metric for key distance would solve this problem and avoid outside traffic even in corner cases.

Internet indirection infrastructure [17] aims at decoupling senders and receivers on the Internet by having an intermediate mapping. It uses Chord to consistently identify a rendezvous node which maintains the mapping of all receivers interested in some sender's messages. This concept is similar to Scribe and our own mechanisms, but it does not explicitly provide location-aware services.

SAAR [11] separates the control plan from the data plane of building a multicast overlay. This is to utilize shared overheads of control maintenance and probing across multiple groups and overlays. This is somewhat orthogonal to the DC2 partitioning of the graph into hierarchical units—SAAR could be used if needed to construct overlays within a unit.

Dr. Multicast [20] focuses on overcoming scalability limitations of IP multicast, by merging similar groups and devising a hybrid IP multicast and IP unicast solution to provide multicast service within data centers. However, this will only be effective when IP multicast is enabled in the platform. DC2 cannot provide the efficiency of IP multicast; however, by focusing on minimizing the link stress of expensive links, we do achieve many of the benefits of IP multicast, without inheriting its limitations.

7 Conclusion

We argue in this paper that traditional group communication systems (e.g., multicast, DHTs) that form a key building block for several large-scale distributed systems need to be revisited to accommodate location awareness. Our motivation stems from the emergence of hierarchical data centers and cloud environments, where links tend to be quite heterogeneous in nature. Further, the operators of these services in these environments have incentive to avoid costly aggregation or core links in a typical scale-up data center architecture which currently run with large oversubscription ratios. A similar argument applies in the context of avoiding inter-data-center links in cloud environments as much as possible to reduce the cost incurred in running such large-scale services.

Unfortunately, most current group communication systems are either built assuming a flat topology or are optimized for reducing latency and other metrics using complicated mechanisms. In contrast, we present a simple group communication system called DC2 that allows nodes to be aware of their location within the hierarchy, and minimize cross-hierarchy communication as much as possible. DC2 achieves low link and node stress by keeping communication within the hierarchy as much as possible thus reducing the cost of operating large-scale distributed services that rely on group communications. In our evaluation using a prototype implementation, we observed that DC2 results in 2-3 $\times$ reduction in the node and link stresses, and reduces median latency by several orders of magnitude compared to Scribe. While the results are based on a relatively small-scale system and exact benefits may vary, we believe that the core ideas embodied in DC2 will prove to be useful in redesigning many different group communication systems, that can now be revisited in the context of hierarchical data centers and global cloud environments.

References

[1] Amazon web services. `http://aws.amazon.com/`.

[2] Open publish subscribe. `http://ops.googlecode.com/`.

[3] BANERJEE, S., BHATTACHARJEE, B., AND KOMMAREDDY, C. Scalable application layer multicast. In *SIGCOMM '02: Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2002), ACM, pp. 205–217.

[4] CASTRO, M., DRUSCHEL, P., KERMARREC, A.-M., AND ROWSTRON, A. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications (JSAC) 20* (2002), 2002.

[5] CHU, Y.-H., RAO, S. G., AND ZHANG, H. A case for end system multicast (keynote address). In *SIGMETRICS '00: Proceedings of the 2000 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (New York, NY, USA, 2000), ACM, pp. 1–12.

[6] GREENBERG, A., HAMILTON, J. R., JAIN, N., KANDULA, S., KIM, C., LAHIRI, P., MALTZ, D. A., PATEL, P., AND SENGUPTA, S. Vl2: a scalable and flexible data center network. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (New York, NY, USA, 2009), ACM, pp. 51–62.

[7] GUO, C., LU, G., LI, D., WU, H., ZHANG, X., SHI, Y., TIAN, C., ZHANG, Y., AND LU, S. Bcube: a high performance, server-centric network architecture for modular data centers. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (New York, NY, USA, 2009), ACM, pp. 63–74.

[8] JANNOTTI, J., GIFFORD, D. K., JOHNSON, K. L., KAASHOEK, M. F., AND O'TOOLE, JR., J. W. Overcast: reliable multicasting with on overlay network. In *OSDI'00: Proceedings of the 4th conference on Symposium on Operating System Design & Implementation* (Berkeley, CA, USA, 2000), USENIX Association, pp. 14–14.

[9] KILLIAN, C. E., ANDERSON, J. W., BRAUD, R., JHALA, R., AND VAHDAT, A. M. Mace: language support for building distributed systems. In *PLDI '07: Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation* (New York, NY, USA, 2007), ACM, pp. 179–188.

[10] MAYMOUNKOV, P., AND MAZIÈRES, D. Kademlia: A peer-to-peer information system based on the xor metric. In *IPTPS '01: Revised Papers from the First International Workshop on Peer-to-Peer Systems* (London, UK, 2002), Springer-Verlag, pp. 53–65.

[11] NANDI, A., GANJAM, A., DRUSCHEL, P., NG, T. S. E., STOICA, I., ZHANG, H., AND BHATTACHARJEE, B. Saar: A shared control plane for overlay multicast. In *NSDI* (2007).

[12] NIRANJAN MYSORE, R., PAMBORIS, A., FARRINGTON, N., HUANG, N., MIRI, P., RADHAKRISHNAN, S., SUBRAMANYA, V., AND VAHDAT, A. Portland: a scalable fault-tolerant layer 2

data center network fabric. In *SIGCOMM '09: Proceedings of the ACM SIGCOMM 2009 conference on Data communication* (New York, NY, USA, 2009), ACM, pp. 39–50.

[13] RATNASAMY, S., HANDLEY, M., KARP, R. M., AND SHENKER, S. Application-level multicast using content-addressable networks. In *NGC '01: Proceedings of the Third International COST264 Workshop on Networked Group Communication* (London, UK, 2001), Springer-Verlag, pp. 14–29.

[14] RHEA, S., GEELS, D., ROSCOE, T., AND KUBIATOWICZ, J. Handling churn in a dht. In *ATEC '04: Proceedings of the annual conference on USENIX Annual Technical Conference* (Berkeley, CA, USA, 2004), USENIX Association, pp. 10–10.

[15] RODRIGUEZ, A., KOSTIC, D., AND VAHDAT, A. Scalability in adaptive multi-metric overlays. In *ICDCS '04: Proceedings of the 24th International Conference on Distributed Computing Systems (ICDCS'04)* (Washington, DC, USA, 2004), IEEE Computer Society, pp. 112–121.

[16] ROWSTRON, A. I. T., AND DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Middleware '01: Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg* (London, UK, 2001), Springer-Verlag, pp. 329–350.

[17] STOICA, I., ADKINS, D., ZHUANG, S., SHENKER, S., AND SURANA, S. Internet indirection infrastructure. *SIGCOMM Comput. Commun. Rev. 32*, 4 (2002), 73–86.

[18] STOICA, I., MORRIS, R., KARGER, D., KAASHOEK, M. F., AND BALAKRISHNAN, H. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01: Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications* (New York, NY, USA, 2001), ACM, pp. 149–160.

[19] VAHDAT, A., YOCUM, K., WALSH, K., MAHADEVAN, P., KOSTIĆ, D., CHASE, J., AND BECKER, D. Scalability and accuracy in a large-scale network emulator. In *OSDI '02: Proceedings of the 5th symposium on Operating systems design and implementation* (New York, NY, USA, 2002), ACM, pp. 271–284.

[20] VIGFUSSON, Y., ABU-LIBDEH, H., BALAKRISHNAN, M., BIRMAN, K., BURGESS, R., CHOCKLER, G., LI, H., AND TOCK, Y. Dr. multicast: Rx for data center communication scalability. In *EuroSys '10: Proceedings of the 5th European conference on Computer systems* (New York, NY, USA, 2010), ACM, pp. 349–362.

[21] ZHUANG, S. Q., ZHAO, B. Y., JOSEPH, A. D., KATZ, R. H., AND KUBIATOWICZ, J. D. Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination. In *NOSSDAV '01: Proceedings of the 11th international workshop on Network and operating systems support for digital audio and video* (New York, NY, USA, 2001), ACM, pp. 11–20.