Dr. Multicast: *Rx* for Data Center Communication Scalability

Ymir Vigfusson
IBM Haifa Research Lab
ymirv@il.ibm.com

Hussam Abu-Libdeh
Cornell University
hussam@cs.cornell.edu

Mahesh Balakrishnan
Microsoft Research
maheshba@microsoft.com

Ken Birman
Cornell University
ken@cs.cornell.edu

Robert Burgess
Cornell University
burgess@cs.cornell.edu

Gregory Chockler
IBM Haifa Research Lab
chockler@il.ibm.com

Haoyuan Li
Cornell University
haoyuan@cs.cornell.edu

Yoav Tock
IBM Haifa Research Lab
tock@il.ibm.com

Abstract

IP Multicast (IPMC) in data centers becomes disruptive when the technology is used by a large number of groups, a capability desired by event notification systems. We trace the problem to root causes, and introduce *Dr. Multicast* (MCMD), a system that eliminates the issue by mapping IPMC operations to a combination of point-to-point unicast and traditional IPMC transmissions guaranteed to be safe. MCMD optimizes the use of IPMC addresses within a data center by merging similar multicast groups in a principled fashion, while simultaneously respecting hardware limits expressed through administrator-controlled policies. The system is fully transparent, making it backward-compatible with commodity hardware and software found in modern data centers. Experimental evaluation shows that MCMD allows a large number of IPMC groups to be used without disruption, restoring a powerful group communication primitive to its traditional role.

1. Introduction

As data center networks scale out, the software stack running on them is increasingly oriented towards one-to-many (multicast) communication patterns. Services such as Facebook and Twitter are supported by multicast-centric architectures. Publish-subscribe and other enterprise service bus layers [24, 26] use multicast to push data to large numbers of receivers simultaneously. This capability allows clustered application servers to replicate state updates and heartbeats between server instances [6, 16, 17], and to maintain coherent caches by invalidating or updating cached information on large numbers of nodes [15, 23]. IP Multicast (IPMC) [12] permits each message to be sent using a single I/O operation, reducing latency and load at end-hosts and in the network. It is included by many of these products as a communication option.

But modern data centers rarely enable IPMC communication because of problems with the technology. IPMC lacks reliable packet dissemination [5, 14], security [19], flow control [31] and scalability in the number of groups [10, 18]. We focus on the last point — preventing disruptions that may arise when a large number of IPMC groups are in use. Our goal is to mend IPMC group scalability in a manner that complements solutions to the other problems seamlessly. Additional goals are efficiency, transparency and robustness of our solution under stress.

IPMC adoption on the wide-area Internet has been limited for a variety of reasons, including economic concerns (how ISPs should charge for IPMC traffic) and security issues (IPMC can be exploited for distributed denial-of-service attacks) [13, 20]. Accordingly, although we believe that MCMD can be extended for WAN settings, this paper focuses only on data centers.

Our key insight is that IPMC addresses are scarce and sensitive resources. When too many are used, network routers and network interface cards (NICs) malfunction in ways that trigger heavy packet loss. As a data center scales up, the aggregated number of IPMC addresses used by the varied applications can easily exceed these limits. One solution is to just require that everything run over TCP. For example, one can modify enterprise service bus and publish-subscribe infrastructure components to create a TCP connection between every source and each of its receivers, sending each packet once per receiver. For situations with large fanouts, some form of application layer overlay could be deployed. Clearly, such an approach will be safe, but it will be more complex and slower than IPMC, which sends just a single packet.

MCMD solves this problem using a novel clustering algorithm to efficiently allocate a limited number of IPMC addresses to selected groups (or sets of groups), with the

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

EuroSys '10 April 13-16, Paris.

Copyright © 2010 ACM [to be supplied]. . . $10.00

number selected to reflect hardware capacity and local administrative policy. Groups that do not receive an IPMC address use unicast communication.

MCMD is implemented as a layer that resides between the application and the operating system network stack. The system efficiently and transparently intercepts standard IPMC system calls, translating each IPMC group address used by the application into a combination of IPMC and unicast addresses. The translation for a group spans two extremes:

- A true IPMC address is allocated to the group.
- Communication to the group is performed using point-to-point unicast messages to individual receivers.

We also examined other options, such as mapping a single application group to multiple IPMC addresses, but concluded that the two cases listed above suffice.

MCMD makes it safe to use a standard, widely deployed communication option that fell into disuse. Our hope is that IPMC might now be revisited for a wide range of possible uses.

The contributions of this paper are thus as follows:

- An approach to mitigate IPMC scalability problems within data centers, which optimizes the allocation of multicast addresses to application layer groups.
- A scalable and robust implementation that resides transparently between the application and the network stack.
- An evaluation using real-world subscription patterns based on a trace collected from a widely deployed commercial application server.

Assumptions. We focus on an administratively homogeneous data center that runs trusted, non-malicious IPMC applications. Our solution will complement any mechanism for IPMC reliability, total ordering or security by virtue of residing in a layer below the IPMC interface. As such, minor packet loss is acceptable. We further assume that the data center network is primarily switched, with multiple levels of switching hierarchy and a top-level gateway router. Finally, we assume that the data center is strongly biased towards commodity hardware and software, and hence would not accept non-transparent interventions that might require modifying applications, or non-standard hardware solutions that might endow NICs or routers with unusually high capacities for IPMC addresses.

Road map. We start by looking closely at the limitations of IPMC in data centers. The policy primitives and architecture of MCMD are discussed in section 3. We formalize the central MCMD optimization problem and provide a greedy algorithm for solving it in section 4. Our evaluation is in two parts, first we evaluate the algorithm on various data sets in section 5, and then we evaluate a prototype of MCMD experimentally in section 6. The last two sections discuss related work and then offer some concluding thoughts.

Alcatel-Lucent OmniSwitch OS6850-48X	260
Cisco Catalyst 3750E-48PD-EF	1,000
D-Link DGS-3650	864
Dell PowerConnect 6248P	69
Extreme Summit X450a-48t	792
Foundry FastIron Edge X 448+2XG	511
HP ProCurve 3500yl	1,499

Table 1. Group capacity on switches. Maximum number of multicast groups supported by 10Gbps switches, according to a NetworkWorld study [22].

2. IPMC Scalability Problems

In this section we touch upon the factors that combine to limit IPMC group scalability in data centers.

2.1 Tragedy of the Commons

A *tragedy of the commons* is said to occur when an individually effective tactic (grazing one's sheep on the commons, in the original formulation) is widely adopted within a community. The individual use is sustainable but not the collective behavior: overgrazed, the commons are denuded.

In a data center, the communications network is a commons: a shared space on which every application relies. Our focus is on the limited IPMC state space on NICs and switches on commodity hardware: filtering becomes ineffective when a large number of groups are used, and this can burden end-host kernels with high rates of unwanted traffic, overwhelming receivers who in turn begin dropping packets. IPMC will only work properly if the number of IPMC groups, both in aggregate and for individual NICs, can be controlled so that the hardware limits are not exceeded. This creates a tension: to scale services up, one wants to massively replicate data, for which IPMC has obvious appeal. Yet if no measures are taken to protect the network, an unbounded demand for IPMC resources could easily arise.

Data centers that permit applications to use IPMC quickly encounter this issue. To scale applications up, modern data centers clone them, running many side-by-side instances. If such a service uses k IPMC addresses, n clones will use nk of them. Thus even given individually "safe" services, by running a collection of them or cloning some to handle more clients, one can generate a collective demand that exceeds the finite capacity.

Membership churn. When a node joins or leaves an IPMC group, the router receives an IGMP packet and must update its forwarding tables. Normally, applications join or leave groups infrequently and this cost will be negligible. However, in poorly designed or malfunctioning applications, high rates of join/leave events could arise, overloading the router and degrading the entire data center network.

2.2 Group Capacity on Switches

We noted that the most fundamental problem is the limited capacity available on devices for storing membership information. Network Ethernet switches vary in sophistication, ranging from layer 2 switches that broadcast all multicast traffic to switches that operate at higher layers or perform IGMP snooping and track multicast group membership in memory. The memory to store group membership is bounded, so what happens when the capacity is reached? Some IGMP-aware switches silently ignore membership information beyond a threshold number of groups [23]. Others begin forwarding IPMC messages on all network segments; it will fall to the NICs to filter the unwanted traffic. This behavior is also seen when a data center router is overloaded with too many IPMC addresses: routers employ filtering mechanisms that can become inaccurate, causing IPMC to behave like a broadcast.

A recent review of the performance of modern 10Gbps network switches found that their IGMPv3 multicast group capacity ranged between 70 and 1,500 [22], as shown in Table 1. Less than half of the switches tested were able to support 500 multicast groups under stress.

2.3 Filters on Network Interface Cards

Unfortunately, end-host NICs also have limited space to store group membership. To filter incoming multicast packets, a typical end-host NIC uses a combination of a perfect check against a small set of addresses, as well as an imperfect check against a hashed location within a table. The latter check is effectively a single-hash Bloom filter.

Stevens *et al.* [25] cites one commercial NIC as having a perfect matching set of 16 addresses and an imperfect matching table of 512 bits, another NIC as having a perfect matching set of 80 addresses with no imperfect matching table, and older NICs as supporting only imperfect matching with a 64-bit table. Even the best of these would accept messages to random IPMC addresses with probability $\frac{1}{2}$ once a node has joined 360 groups.

2.4 Repercussions

These limitations add up to trouble. If these limits are exceeded, every IPMC packet sent to groups above the limit will become a broadcast, forwarded to every node in the data center, received by every NIC and in effect dumped onto the operating system stack. An operating system can silently filter and discard unwanted traffic at low data rates, but high rates of aggregated IPMC traffic of multiple groups is a different matter. The operating system will be overwhelmed and drop incoming packets of all kinds: not just IPMC packets, but also unicast UDP and TCP. TCP will interpret the loss as a signal to throttle back.

Packet loss. We conducted an experiment to try to provoke packet loss. A multicast sender transmits on $2k$ multicast groups, whereas the receiver listens to only k of those multicast groups. The sender transmits 8,000 byte packets at a constant rate of 15,000 packets/sec, and divides the traffic among the $2k$ multicast groups in the system, so the receiver expects to receive half of the traffic, or 7,500 packets/sec. The extra groups simulate background multicast traffic that is normally filtered out by the switch and NIC. The sender and receiver both have 1Gbps NICs and are connected by an IGMP-aware switch.

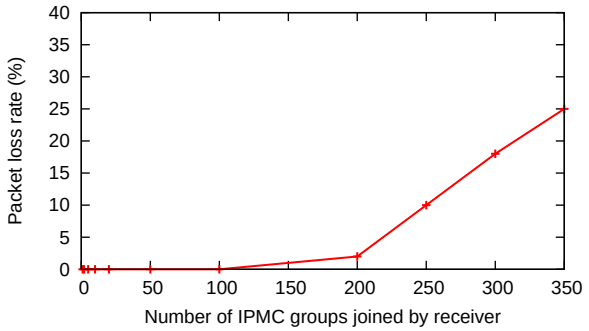

Figure 1. IPMC scalability issues. Packet loss rate at a receiver as multicast traffic is divided among more IPMC groups.

We varied the number of multicast groups k and measured the packet loss at the receiver. The results show that the hardware can handle roughly 200 IPMC groups before high CPU load and packet loss ensues. The resulting packet loss rate as a function of k can be seen in Figure 1. Our findings confirm the intuition given above.

Multicast storms. Even modest levels of IPMC packet loss due to overload can dramatically impact higher layers in the software stack, and will stress any IPMC reliability layer. For example, with SRM [14], a slow receiver who has lost packets will continuously multicast retransmission requests to the group, potentially provoking a multicast storm of retransmissions by other receivers that slows down the entire group and causes further packet loss—further cascading to disrupt the entire data center [6, 7].

3. Design and Implementation

The basic operation of MCMD is simple. It translates an application-level multicast address used by an application to a set of unicast addresses or a network-level multicast address, as shown in Figure 2. The translation is governed by an administrator specified policy for the data center, as described in the following subsection.

MCMD consists of two primary components:

- A *library* module overloads the standard socket interface and allows MCMD to be transparently loaded into applications.
- An *agent* daemon is responsible for implementing the user-defined policy and the application-level multicast mapping.

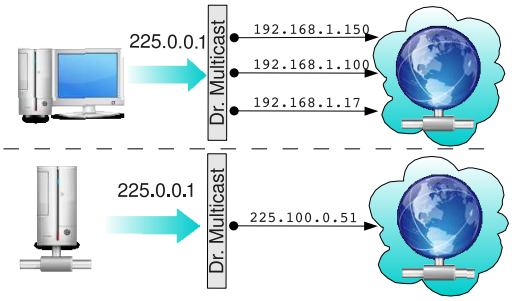

Figure 2. Translation. Two under-the-hood mappings in MCMD, point-to-point unicast mapping (top) and a direct IPMC mapping (bottom).

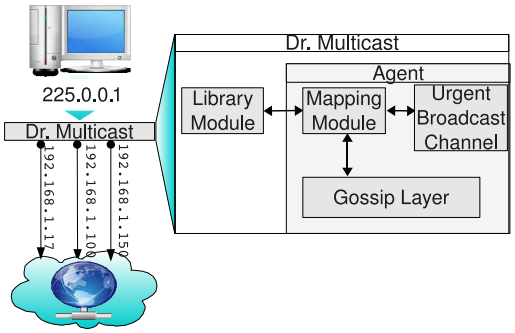

Figure 3. The MCMD architecture.

Each node in the system has a running agent, and one of these agents is designated as a *leader* that periodically issues multicast group mappings. The mapping information is replicated across all the agents via a gossip layer, and an additional urgent broadcast channel is used to quickly disseminate urgent updates. Figure 3 highlights the different components of MCMD. We will detail the design and implementation of both components in this section.

Notice that when using MCMD, there is an important difference between application-level IPMC groups and network-level IPMC groups. With MCMD, the former becomes a purely logical abstraction seen by applications. The latter are the physical addresses used by the hardware, but multiple logical groups can share the same physical address, and these addresses are under MCMD control.

3.1 Policy

MCMD allows administrators to mitigate IPMC group scalability concerns using the following knobs:

- *limit-IPMC-node(i, m_i):* Node i is allowed to join at most m_i network-level IPMC groups.
- *limit-IPMC(m):* A maximum of m IPMC groups can be used within the data center.

The use of network-level IPMC can be disabled system-wide or at an individual node i by respectively setting $m = 0$ or $m_i = 0$.

Setting the policy. The policy can be dynamically changed by updating a configuration file at any agent, and the changes will propagate to other agents via gossip and the urgent broadcast channel. In practice, we imagine a mixture of hard policy limits calibrated to match router and NIC characteristics, with soft policies: MCMD can be extended to support fine-grained access-control policy primitives and rate-limiting, enabling administrators to allow or deny specific applications from joining particular application-level groups, or to allow operators to specify triggers for events such as high rates of messages or packet loss. Changes take effect quickly, permitting a level of security and offering the administrator a means of blocking IPMC use by buggy applications.

3.2 Library Module

The library module exports a standard IP Multicast interface to applications [12]. By overloading the relevant socket operations, MCMD intercepts join, leave and send operations. For example:

- In the overloaded version of `setsockopt()`, invocations with *e.g.* the `IP_ADD_MEMBERSHIP` parameter will be intercepted by the library module. An IGMP join message will only be sent if the application-level IPMC address is mapped to a new network-level IPMC address.
- `sendto()` is overloaded so that a send to an application-level IPMC group address is intercepted and converted to multiple sends to a list of addresses.

Interaction with agent. The library module interacts with the agent daemon via a UNIX socket, and periodically pulls and caches the list of IPMC groups it is supposed to join as well as translations for the application-level groups it wants to send data to. The library module may receive invalidation messages from the agent, causing the library module to refresh its cached entries. Simultaneously, the library module pushes information and statistics about grouping and traffic patterns used by the application to the agent. A traffic pattern is an exponential-average of the message rate λ_g received in application-level group g.

3.3 The MCMD Agent

The *agent* is a background daemon process that runs on every node in the system. Each agent instance acts as a *mapping module*, maintaining the following pieces of information that are replicated on every agent in the system — collectively referred to as the *agent state*:

- *Membership sets:* a map from each node to the application-level groups within which it receives messages.
- *Sender sets:* a map from each node to the application-level groups in which it sends messages.

- *Group translations:* a map from application-level groups to a set of unicast addresses, a single network-level IPMC address, or both.

Each agent in the system has read-access to a locally replicated copy of the agent state. Write-access to the agent state, however, is strictly controlled. Each node manages its own membership set and its sender set. The group translations may only be modified by the leader agent. When any agent, leader or not, writes to its local copy of the agent state, the change is propagated to other agents in the system via a gossip layer, which guarantees eventual consistency of agent state replicas. Since each item in the agent state has exactly one writer, there are no conflicts over multiple concurrent updates to the agent-state.

In large deployments of MCMD, replicating the entire agent state on every node may be infeasible. To reduce the network overhead needed for replication, the administrator can opt to deploy agents on a subset of the nodes in the data centers. However, this can increase the load on these agents since more library modules will rely on each agent for receiving and sending updates. We discuss ways to accommodate very large scales in Section 4.2.

Group translations. The leader agent uses the group membership and sender information to determine the best set of translations from application-level groups to network-level IPMC addresses. Once these translations are written to the leader's local state, the gossip layer disseminates the updates to other agents in the system, which read the translations off their local replicated copy of the agent state and direct their corresponding library modules to join and leave the appropriate IGMP groups. If present, groups with no receivers are mapped to empty lists, and groups with exactly one receiver are mapped to unicast, while non-trivial groups are subjected to our translation algorithm. Section 4 describes how the leader allocates network-level IPMC resources to application-level multicast groups.

State replication. We use a gossip-based failure detector [28] to replicate the agent state across all the agents. Each node maintains its own version of a global table, mapping every node in the system to a time-stamp or heartbeat value. Every T milliseconds, a node updates its own heartbeat in the map to its current local time, randomly selects another node and reconciles maps with it. The reconciliation function is simple – for each entry, the new map contains the highest time-stamp from the entries in the two old maps. As a result, the heartbeat timestamps inserted by nodes into their own local maps propagate through the system via gossip exchanges between pairs of nodes.

The comparison of maps between two gossiping nodes is highly optimized. The initiating node sends its peer a set of hash values for different portions of the map, where portions are themselves determined by hashing entries into different buckets. If the receiving node notices that the hash for a portion differs, it sends back its own version of that portion. This interchange is sufficient to ensure that all maps across the system are kept loosely consistent with each other. An optional step to the exchange involves the initiating node transmitting its own version back to the receiving node if it has entries in its map that are more recent than the latter's.

Urgent broadcast channel. Gossip is a robust way to replicate agent state data across multiple nodes, but can be slow. We use an urgent notifications broadcast channel to quickly disseminate important updates, and to ensure that nodes are responsive to sudden changes in the state of the system, in particular to membership and mapping information. The channel is used for three types of events.

New receiver: When a new receiver joins a group, its agent updates the local version of agent state via gossip and simultaneously sends unicast notifications to every node listed in the agent state as a sender to that group, as well as the leader. As a result, senders can immediately include the new receiver in their transmissions. In addition, the new receiver's agent contacts the leader agent for updates to the sender set of that group; if the leader reports back with new senders not yet reflected in the receiver's local copy of the agent state, the receiver's agent sends them notifications as well.

New sender: When a new sender starts transmitting to a group, the agent running on it updates the sender set of the group on its own local version of the global agent state, and simultaneously sends a notification to the leader agent. The leader agent responds with the latest version of the group membership information for that particular group.

Translation map change: When the leader agent creates or modifies a translation, it sends notification messages to all the affected nodes — receivers who should join or leave IPMC groups to conform to the new translation, and senders who need to know the new translation to transmit data to the group. These messages cause their recipients to "dial home" and obtain the new translation from the leader.

Leadership. Our gossip protocol is also used to track nodes in the system that are eligible to take on the leader role, and also to detect leader failure [28]. The leader is defined to be the eligible node with the lowest node identifier. If by some fluke two leaders run concurrently in a non-partitioned system, the situation will resolve itself: nodes that see proposed mappings from both simply ignore the IPMC mapping proposed by the one with the larger node identifier. Moreover, two concurrent leaders running in the same portion of the network would select nearly identical mappings: our heuristic is stable and with similar data produces identical or nearly identical mappings. In the event of a partitioning failure, our solution will result in multiple leaders, one per partition. As soon as partitioning ends, one of the two leaders will dominate the other.

Because we use an urgent broadcast channel when mappings change, backed up by gossip repair to disseminate

mappings, no node will be confused about who the leader is for more than a few milliseconds. A leader election mechanism with stronger guarantees could be implemented if needed, but the current scheme is simple and appears to be adequate for our target setting.

Rate limits and churn control. In the large, well-managed data centers of interest to us, node failures are not common enough to represent a problematic source of overhead, as we will see in section 6. The more likely sources of membership changes are associated with startup or shutdown of services that span groups of nodes and use IPMC. For example, suppose a data center hosts services on behalf of many corporate customers and handles flash loads by launching extra copies. A "service" might well run on many nodes, using IPMC internally. Back-end applications send updates to these cloned front-end services, again using IPMC. Thus dynamic expansion or reduction in the number of cloned copies of such a service is likely to be an important source of dynamicism. We evaluate such a scenario in the experimental part of this paper.

MCMD limits the rate of membership change events at any single node as a defense against buggy applications that frantically join and leave groups at high speed. Under normal (non-buggy) conditions, joins and leaves involve a single unicast exchange with the leader, imposing load on it that increases linearly with the rate of such events in the data center as a whole. As mentioned above, the node that joined or departed from the group then sends a multicast to update the membership lists of other group members. Thus MCMD handles services composed of nodes that tend to join groups and then remain in them, but may face performance issues with applications that create IPMC groups very dynamically.

The purpose of the rate limits is to keep multicast communication within a customizable "safe zone", preventing buffers on network cards from filling up and potentially spiraling into a multicast storm. Alternatively, the rate checking mechanism could notify data center operators if multicast traffic rates exceed specified thresholds.

Robustness. When a node notices that the time-stamp value for some other node in its map is older than T_{fail} seconds, it flags that node as "dead", where T_{fail} is picked using the analysis in [28]. The entry is not immediately deleted, but maintained in dead state for T_{fail} more seconds. This prevents a dead node from being resurrected by some node that has not yet sensed the failure. After $2T_{\text{fail}}$ seconds have elapsed, the entry is deleted. In [28] this scheme is shown to be quite robust. Our experiments suggest that T_{fail} should be in the order of $R \log n$, where R is the gossip rate and n the system size.

The system is able to tolerate leader failure because all nodes replicate the agent state. Once agents realize that the leader is no longer responsive, the leader is marked as dead and the failure detector disseminates that information to all the nodes. A leader election protocol is started to appoint a new leader agent, selecting the operational node with the largest uptime value.

Memory requirements. The size of the replicated global view is not prohibitive, because only 24 bytes are required per membership. For example, we can store the agent state for a 1,000-node cluster with a membership pattern based on the WVE trace from section 5.2 within 1MB of memory.

4. Optimization Problem

At the heart of MCMD is the optimization problem of making the best use of scarce IPMC resources. The MCMD leader can assign a limited number of IPMC addresses to application-level groups in the system to reduce redundant network traffic. We observe that application-level groups with similar membership could be assigned the same IPMC address at the cost of forcing some receivers to filter out unwanted traffic. Traffic to those groups which are not assigned a network-level IPMC address is sent using an alternative multicast mechanism, currently point-to-point unicast. We trade off the following objectives.

- *Minimize the number of network-level IPMC addresses.* NICs, routers and switches scale poorly in the number of IPMC addresses, as discussed earlier.
- *Minimize redundant sender transmissions*. When a sender maps IPMC to unicast, that sender will send identical packets to multiple destinations, incurring an associated cost.
- *Minimize receiver filtering*. If a receiver must filter unwanted traffic it will incur significant costs [9]. If the unwanted traffic load becomes too high, packet loss will ensue: precisely the condition MCMD was created to address.

The above goals spur a family of optimization questions, some which have been previously addressed in the literature. The *channelization* problem [1, 27, 32] is the following formulation:

> *Allocate a fixed number of IPMC addresses to collections of groups to minimize both sender transmission costs and receiver filtering costs such that subscribers receive all messages they are interested in at least once.*

The channelization problem is NP-complete [1], and several heuristics to solve it have been proposed and experimentally evaluated in the past [1, 27].

In this section, we extend the channelization problem to take into account alternative multicast mechanisms, specifically point-to-point unicast, as well as the administrative policy. We present a greedy algorithm to tackle the generalized translation problem, which remains NP-complete. Finally, we show the feasibility of the algorithm by evalu-

ating it on a wide range of inputs from real-world data sets and synthetic models.

4.1 Model

Let U denote the set of n *users* (or nodes) in a system, and G denote the set of k application-level *groups*. Define $U_g \subseteq U$ for $g \in G$ as the set of users who subscribe to group g, and $G_u \subseteq G$ for $u \in U$ as set of groups to which user u subscribes, that is $G_u = \{g \in G : u \in U_g\}$. In the example in figure 4, $U_{g_5} = \{u_4, u_6\}$ and $G_{u_1} = \{g_1, g_2, g_3\}$.

Recall from section 3.1 that the administrator policy permits m network-level IPMC groups to be used in the data center and that the limit of network-level IPMC groups for node $u \in U$ is m_u. The goal is to find a set P of m pairwise disjoint *meta-groups* $P = \{P_1, P_2, \ldots, P_m\}$, where $P_i \subseteq G$ for each i. The idea is that meta-groups should contain "similar" groups in terms of membership. For example, the partition $P = \{P_1, P_2, P_3\}$ in figure 4 covers all five groups. The meta-group $P_1 = \{g_1, g_2, g_3\}$ merges together three application-level that have similar sets of receivers. We can then assign each meta-group P_i in P to a network-level IPMC addresses that is shared by all the groups in P_i and use point-to-point unicast for groups that are not contained in any meta-group.

Let $\Pi = \bigcup_{i=1}^{m} P_i$ be the set of groups covered by meta-groups. In the example, $\Pi = \{g_1, \ldots, g_5\}$. Let $R_u = \{P_i : u \in U_g \text{ for some } g \in P_i\}$ denote the set of meta-groups that cover every application-level group user u has joined, and $\Pi_u = \bigcup_{P_i \in R_u} P_i$ be the set of groups that belong to the same meta-groups as u. To illustrate, $R_{u_4} = \{P_1, P_2, P_3\}$ and $\Pi_{u_4} = \{g_1, \ldots, g_5\}$ in figure 4, whereas $R_{u_2} = \{P_1\}$ and $\Pi_{u_2} = \{g_1, g_2, g_3\}$ even though u_3 does not belong to g_3.

Consider an arbitrary group $g \in G$. Assigning network-level IPMC addresses to meta-groups effects senders and receivers as follows:

- *Senders:* If g is contained in some meta-group P_i, then a single message is sent to the IPMC address associated with P_i. Otherwise, the message is sent individually to each receiver in U_g.
- *Receivers:* If g belongs to some meta-group P_i, then a receiver may need to filter out messages to other groups in P_i that it did not join. Otherwise, no filtering is necessary.

Let us define these overheads formally as *duplication* and *filtering* costs. Recall that λ_g denotes the average rate of traffic received in group g per time unit.

Definition: *Let $\beta \geq 0$. Define the total* cost *of translation P on a set of groups G as*

$$cost(P, G) = cost_F(P, G) + \beta cost_D(P, G),$$

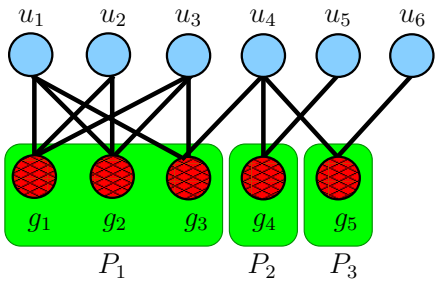

Figure 4. Notation. The users on top receive data from the application-level groups on the bottom. The rectangles correspond to the meta-groups in partition P.

where the filtering cost $cost_F(P, G)$ *is*

$$cost_F(P, G) = \sum_{u \in U} \sum_{g \in \Pi_u - G_u} \lambda_g$$

and the duplication cost $cost_D(P, G)$ *is*

$$cost_D(P, G) = \sum_{g \in G - \Pi} \lambda_g \left(|U_g| - 1\right).$$

Example. Looking at figure 4 again, $\text{cost}_F(P, G) = \lambda_{g_1} + \lambda_{g_2} + \lambda_{g_3}$ since user u_4 must filter traffic from groups g_1 and g_2, and u_2 must filter traffic from group g_3. For the duplication cost, not that it counts only redundant copies of messages beyond the first one sent. All groups in the example are covered by the partition P so the duplication cost in the example is zero. If the partition was instead $P' = \{P_1\}$, then the duplication cost would be $\text{cost}_D(P', G) = \lambda_{g_4} + \lambda_{g_5}$ since messages sent to the uncovered groups g_4 and g_5 each need to be sent to two receivers.

Optimization problem. The generalized channelization problem for the translation mechanism is the following:

> *Given a set of groups G, find the set P of m meta-groups such that $|R_u| \leq m_u$ for all $u \in U$ with the lowest cost(P, G).*

By minimizing only filtering costs ($\beta = 0$) and making the node-limits m_u infinite, we obtain the original channelization problem from Adler *et al.* [1] as a special case. Finding an optimum solution to our optimization question is thus an NP-complete problem. In the next subsection, we present an algorithm to find an approximate solution.

For simplicity, we will assume throughout that $\beta = 1$. In other words, we assume that the cost of producing redundant packets for the sender and the networking hardware roughly equals the CPU cost for filtering out an unwanted packet.

4.2 Translation Algorithm

We give a simple heuristic method that constructs meta-groups by traversing large groups with high traffic, then

repeatedly moving groups to these meta-groups in a greedy fashion if doing so decreases the total cost.

Algorithm 1 TRANSLATION(G), where G is the set of groups.

```
m' ← 0
for all g ∈ G in decreasing order by λ_g|U_g| do
    i ← arg max_{i=1,...,m'} (C(i, ∅) − C(i, {g}))
    if (i = 0 or C(i, ∅) − C(i, {g}) < 0) and m' < m then
        m' ← m' + 1 {Create a new meta-group}
        P_{m'} ← {g}
    else
        P_i ← P_i ∪ {g}
    end if
    G ← G − {g}
end for
```

We assume that $\arg\max$ over the empty set returns 0. The function $C(i, H)$ computes the solution cost after the groups in H have been migrated to meta-group P_i. More specifically, $C(i, H) = \text{cost}(\hat{P}, G - H)$, where $\hat{P}$ equals P except P_i is replaced by $P_i \cup H$.

For clarity, the algorithm does not address the node-specific limits m_u on the number of multicast a receiver can join. This is amended by a provision to the loop to only consider those groups g whose members are all below the m_u limit.

We also adapted other algorithms from the literature, such as a variant of k-means, but found that the greedy heuristic consistently outperformed those approaches. We are currently exploring how well the algorithm approximates the optimal one in worst-case scenarios.

Incremental version. Because the mapping module will periodically update the translations, a desirable property is that if group memberships do not change much, neither should the translation computed by the algorithm. Once the translation algorithm is rerun, we initialize the meta-groups with the output of the previous run. If a new group g has been created with value $\lambda_g|U_g|$ higher than the value $\sum_{g' \in P_i} \lambda_{g'}|U_{g'}|$ for some meta-group P_i, or if there are fewer than m meta-groups in the system, then P_i is broken up and g gets a meta-group of its own. Toggling the transport mechanism from using network-level IPMC groups to point-to-point unicast in this case only affects group g and the groups in P_i. The for-loop is then executed as before.

It follows that the translation algorithm can be run incrementally for each new event (*e.g.*, a membership change) without imposing high load on the leader agent.

Running time. The running time of the translation algorithm is $O(kmQ)$, where k is the number of groups, m is the number of meta-groups, and Q is the size of the largest group. Because the m and Q factors both depend on physical hardware, they can be assumed to be constant with respect to the number of groups; hence the algorithm scales linearly in the number of groups. The running time of the algorithm to compute each data point in Figure 5 (described later) was 1.13 seconds on average using a Python implementation. Note that running the translation algorithm from scratch represents a worst-case use, more typically the algorithm only needs to incrementally adjust a previously computed solution.

Decentralization. In ongoing work, we are exploiting properties of the cost function to create a decentralized translation algorithm to accommodate very large networks. Nodes would maintain information only about membership of nodes in groups they subscribe to, together with statistics about sizes and traffic rates of application-level groups. Each node would run a portion of the computation based on a portion of the overall group overlap graph, and probabilistically report observed group traffic rates to a global group via gossip.

5. Evaluation of the Translation Algorithm

Before we can discuss how to evaluate a group optimization method, such as the translation algorithm, we must first ask what kinds of groups and group structure should be expected within data centers. Here the term *group structure* refers to properties of individual groups as well as overlapping membership between multiple groups.

The answer to the question is non-trivial, because the definition of a "group" is fuzzy and depends on context. Some groups abstract social human interactions, such as chat rooms, newsgroups and blogs. Others arise in the course of design of real systems, for instance the groups used to replicate data within components of a distributed system to allow load to be spread over multiple computing nodes [29], or the groups that serve as communication channels for multiple inventory systems while they process a web-query from a customer.

We obtained data sets and models for both abstractions (social groups and systems groups) in the form of bipartite graphs between a set of users and groups. We will first describe the social data sets, then discuss an interesting systems data set and last evaluate the translation algorithm on those inputs graphs.

5.1 Social Data Sets

We obtained a number of data sets for socially influenced group structure, including one generated by a model. Each data set consists of edges between groups and the users belonging to those groups.

- LIVEJOURNAL: LiveJournal communities and users who belong to them [4].
- AMAZON: Products reviewed by customers at Amazon.com [21]. Each product corresponds to the group of customers who reviewed the product.
- YAHOO-GROUPS: The users and topics of Yahoo! Groups, an on-line community driven forum [33].

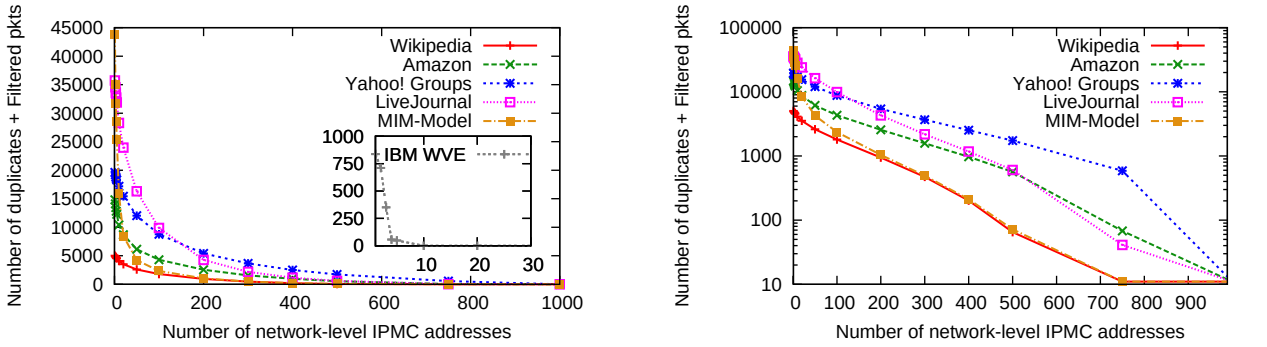

Figure 5. Translation algorithm. Number of duplicate packets sent plus packets that must be filtered out by receivers as we vary the number of available IPMC groups after running the translation algorithm on 1,000 group samples from the social data sets and the full WVE trace (embedded). Most of the cost is due to duplicate packets. Note the logarithmic scale on the y-axis on the right.

- WIKIPEDIA: Wikipedia articles ever edited by registered authors [11]. Each article represents a group of those who are interested in it.
- MIM-MODEL: A bipartite generalization of the preferential attachment model to produce power-law degree distributions for users and groups [29].

These data sets may not directly correspond to a realistic use of multicast; for instance, it is unlikely a multicast group is assigned to every LiveJournal community or Wikipedia article. Instead, they illuminate the similarity between human interests, which can indirectly benefit socially influenced data-center applications, such as a publish-subscribe layer for trading in the stock exchange [27] or updates sent to newsfeed followers in real-time on Facebook or Twitter.

5.2 Systems Data Set

We also obtained a trace of multicast patterns from a real-world system. IBM WebSphere Virtual Enterprise (WVE) is a widely deployed commercial distributed system for running and managing web applications [16]. Each WVE cell consists of some (possibly large) number of servers, on top of which application clusters are deployed. Larger data center deployments clone these cells, partitioning clients among them to balance load. Internal management of each cell, such as workload balancing, dynamic configuration, inter-cluster messaging and performance measurements, uses a built-in bulletin board component. The bulletin board (BB) exports an interface resembling publish/subscribe which is implemented as an overlay [8]. Note that when more than one cell is active, each cell uses its own private BB service.

IBM created the trace by deploying 127 WVE nodes constituting 30 application clusters for a period of 52 minutes, and recording the messages sent to each group along with the sender and receivers. An average node posted to 280 groups and received from 474 groups. There were 1,364 application-level groups with both senders and receivers that were used to disseminate messages during the trace.

The group patterns in the trace are highly structured. There are four prevalent communication patterns for publishers and subscribers: few-to-few, few-to-many, many-to-few, and many-to-many. Here, *few* means no more than 10 nodes, and *many* implies all 127 nodes except at most 10. Interestingly, every group in the trace fits one of the four categories. Some communication patterns directly result from the design of particular WVE components — a subset of the many-to-few groups, for instance, were used for gathering statistical reports. Other behavior is harder to characterize, supporting our case for automatically compressing subscription patterns instead of changing existing code to manually optimize group membership.

5.3 Translation Algorithm on Data Sets

We evaluated the translation algorithm on each of the data sets, using a sample of 1,000 groups from each, assigning a network-level IPMC address to each meta-group produced by the algorithm. Figure 5 shows how the total cost decreases as the number of available IPMC addresses increases. Note that if 1,000 IPMC addresses are available, each group in the data set can use IPMC as transport and the cost due to filtering and duplicates becomes zero.

The cost decrease is close to exponential, as seen in Figure 5, implying that major cost savings arise even when a modest number of IPMC addresses are enabled in the network. For every data set and model we tried, the translation algorithm endowed with only 100 IPMC addresses — 10% of the total number of groups — saves more than 50% of the cost that is incurred when IPMC is disabled. Using 4 IPMC addresses in the WVE data set, the cost of filtering and duplicates using our algorithm is almost negligible, and with 10 IPMC addresses it goes down to zero as the embedded plot in Figure 5 shows. In other words, the translation algorithm was able to assign a meta-group to every distinct

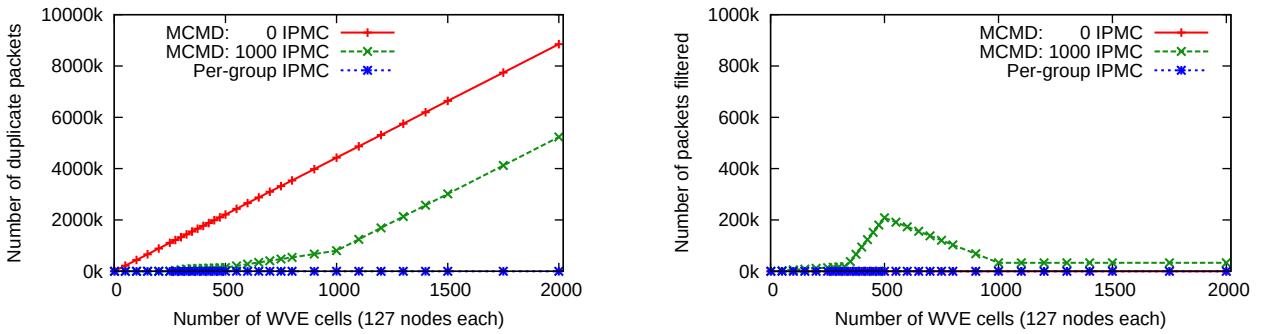

Figure 6. Traffic simulation. Total number of duplicate sends (left) and packets filtered (right) assuming an application that sends 1 packet/group per unit time as we vary the number of concurrent WVE cells, each consisting of 127 nodes and 1,364 groups. The MCMD translation algorithm is endowed with 1,000 network-level IPMC addresses. MCMD has optimal duplication cost until about 250 WVE cells consisting of 30,000 nodes and 340,000 groups, while filtering costs are modest. Note the different scales on the y-axes: filtering costs are substantially lower than duplication costs.

group pattern encountered in the data set. We conclude that our algorithm makes effective use of a scarce number of IPMC addresses.

Traffic rates. In the experiment above, we assumed a uniform rate of traffic on all the groups even though this assumption may be unreasonable. For instance, 80% of the traffic in the WVE trace was carried in just 18% of the groups. By adopting a more realistic model of traffic rates, specifically by letting λ_g follow a power-law distribution uncorrelated with the group size, we observed even more dramatic cost savings than with uniform traffic rates for every single data set. The need for brevity makes it impossible to include the associated figures in this paper.

5.4 Simulation

Thus far, we have demonstrated that our greedy translation algorithm is able to make efficient use of a limited number of network-level IPMC groups without incurring high costs of filtering or duplication. We now vary the number of application-level groups while keeping the number of network-level IPMC addresses constant to understand how our translation algorithm works at scale.

In the previous subsection, we treated the WVE trace as a form of ground truth giving fine-grained information about group membership and communication patterns in a real, widely used platform. Here we use the trace as a tool for generating substantially larger data center scenarios by cloning parallel instances of the membership patterns. We simulate k side-by-side cells of WVE running on distinct sets of 127 nodes each, while running a single instance of MCMD that spans the entire data center.

We compare the number of send operations for a sender who transmits one packet per group in the cloned WVE trace scenario for three different transports:

- *MCMD: 0 IPMC.* Unicast transmissions to each receiver.
- *MCMD: 1000 IPMC.* Using the MCMD translation algorithm with $1,000$ network-level IPMC addresses.
- *Per-group IPMC.* Each group has a network-level IPMC address.

We assume a traffic rate of one message per group, per time unit. In the multicast case, this results in a single network message. When a group is mapped to unicast, the number of network messages will be determined by the number of group members.

The simulation shows that the cost for the sender using MCMD with the translation algorithm is between the two extremes. In Figure 6(a), we see that until 250 WVE cells — a scenario constituting 31,750 nodes and 341,000 groups — MCMD uses the optimal number of sends with zero duplicates. The filtering costs in Figure 6(b) are also modest during that period. This confirms our earlier observation that 4 IPMC addresses suffice with negligible cost for a single WVE instance. With more than 250 concurrently active WVE instances, trade-offs between duplication and filtering costs arise. Up to 500 instances, the algorithm saves duplication cost by merging less similar groups, increasing filtering costs. As we add even more WVE cells, MCMD prioritizes large groups for mapping to IPMC and still saves cost compared to individual unicast even though it has fewer than 4 IPMC addresses available per WVE instance. Beyond 1,000 concurrent WVE instances — 127,000 nodes and 1,364,000 groups — MCMD has fully exhausted the 1,000 network-level IPMC groups it was provided and must resort to using individual unicast to further groups.

Although we lack WVE traces for cells containing larger numbers of nodes, we believe that MCMD would do just as well when confronted with scaled scenarios in this

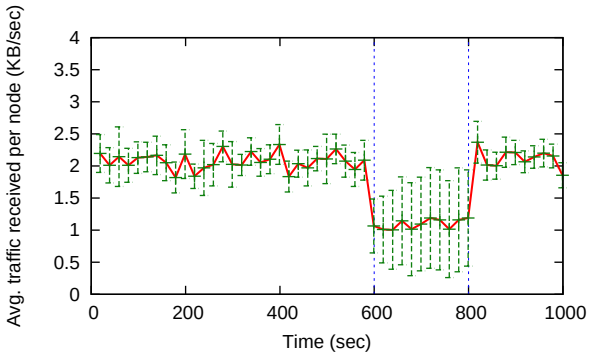

Figure 7. Robustness. Average traffic received per MCMD agent in a 91-node deployment over time. At time 600, half of the nodes are killed and come back to life at time 800. Error bars represent sample standard deviation over 24 trials.

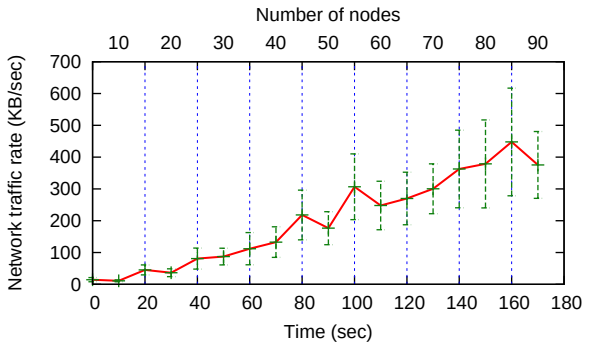

Figure 8. Network overhead. The total network traffic overhead in a 90-node deployment where 10 MCMD agent nodes enter the system every 20 seconds. Each agent has an application whose group membership is selected randomly from the WVE trace. The network load scales linearly in the number of agents. Error bars represent sample standard deviation over 20 trials.

dimension. We say this because MCMD exploits correlation in group interests; to defeat the translation algorithm a system would need to exhibit highly uncorrelated group membership patterns. In real-world uses of multicast, we believe such unstructured membership to be more of an exception, and that correlated structure like in the IBM WVE system is closer to the rule. Group membership correlations are discussed further in [29].

6. Evaluation of Prototype

Thus far, we have shown that the algorithm used by MCMD can effectively map application-level groups to a small set of IPMC addresses. We now evaluate a prototype implementation of MCMD to answer the following questions:

- *Robustness.* How do node failures affect MCMD?
- *Overhead.* How much overhead does the system impose on applications and on the network?
- *Scalability.* Does MCMD scale in the number of IPMC groups without experiencing disruption?

Our results suggest that MCMD provides group scalability to IP Multicast applications with negligible overhead, while remaining robust to failures.

Experimental set-up. We have implemented MCMD in C/C++ and deployed it on 91 nodes in the DETERlab testbed and 17 nodes in the CUNET Emulab test bed. The nodes in DETERlab are equipped with Intel Xeon 64-bit 3.0GHz processors, 2GB of RAM and Intel Pro1000 1Gbps NICs. They connect to a Cisco Catalyst 6500 series high-end switch. The CUNET Emulab nodes are connected by 1Gbps Broadcom Tigon3 NICs to a single Nortel Baystack BS5510-48T IGMP-aware switch.

Every node runs a single MCMD agent and one of the following two simple IPMC applications:

- A *sender* application continuously transmits packets at a fixed rate to k IPMC groups in a round-robin fashion.
- A *receiver* application joins the same k IPMC groups, and retrieves incoming packets in a loop.

Agents gossip once per second.

6.1 Robustness and Network Overhead

MCMD must be robust if it is to be deployed in data centers; with this in mind, we chose to implement a gossip-layer and to replicate agent state on all nodes. A secondary benefit for using gossip is to maintain a balanced load on the network.

We subjected the 91-node MCMD deployment to a major correlated failure: half of the nodes in the data center died simultaneously at time 600 in Figure 7. Nevertheless, the MCMD system continued running. The dead nodes were resurrected at time 800 without any problems.

To evaluate network overhead, we measured the rate of gossip and urgent broadcasts in MCMD on 90 nodes in the DETERlab test bed. We gradually introduce nodes into the system with 10 nodes entering every 20 seconds. Every node runs an MCMD agent and a receiver application that picks a random node from the WVE trace and joins application-level groups accordingly. The random roles are fixed over 20 measurement trials. The total traffic overhead from running MCMD in this setting is shown in Figure 8. When 90 nodes are in the system the total traffic imposed is less than 500KB/sec, or 5.6 KB/sec on average per-node. The increase in network overhead is roughly linear in the number of nodes.

One can extrapolate the overhead from Figures 7 and 8 to predict larger scale behavior. The per-node overheads of MCMD seen in Figure 7 are quite low and the current implementation could scale to large configurations without obvious problems. The network-wide load imposed by the

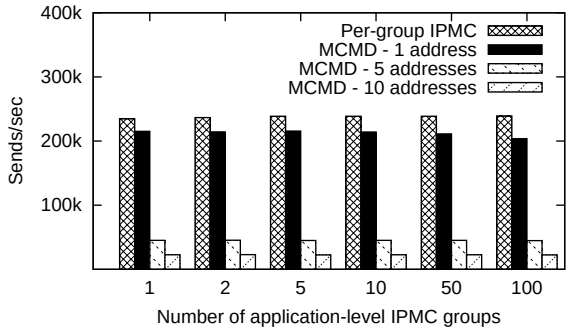

Figure 9. Application overhead. Experiment on a single IPMC sender measuring the number of send operations sustained with and without MCMD library module loaded for direct IPMC mapping and using MCMD with 5 or 10 addresses per group. Error bars over 10 trials were too small to be visible.

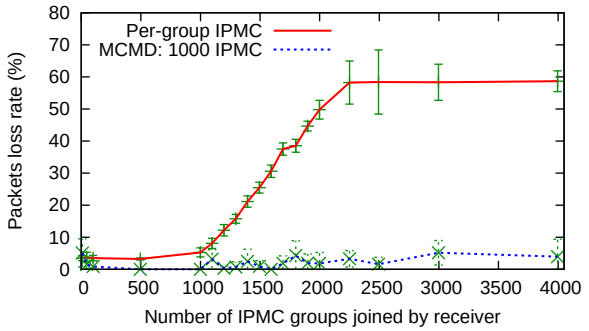

Figure 10. Scalability in the number of IPMC groups. Experiment showing that per-group IPMC can sustain heavy packet loss while MCMD with 1,000 IPMC addresses prevents ill-effects. Error bars represent sample standard deviation over 10 trials.

system, however, might be prohibitive: with 100,000 nodes, the network would bear approximately 1GB of overhead per second.

As discussed in Section 4.2 (decentralization), a large scale system could subdivide execution of the optimization heuristic, rather than having a single leader run it for the whole system. For example, a scaled-up WVE system with k cells might result in k connected components, plus perhaps an additional component consisting of management groups that span most or all nodes. We then divide the budget of IPMC addresses up, allowing each component to run its own version of MCMD.

6.2 Application Overhead

We next measure the overhead of using the MCMD library layer on a simple IPMC application. The application is a copy of the sender application but with rate-limiting disabled so that it sends IPMC packets to k groups as rapidly as possible. We measure the maximum sending rate possible with and without MCMD. We also vary k to see the effect of the data structures used by MCMD.

First, assume MCMD maps each application-level group to a single network-level IPMC address. We saw an average increase of 10% CPU utilization for the application, irrespective of the number of groups. We observe that the number of operations per second falls by 10-15% in the application by running MCMD, as the tall bars in Figure 9 show. Collisions in hash-maps account for the slight drop in performance as k increases.

Next, consider the case where each group resolves to both an IPMC address and a list of unicast addresses. The shorter bars in Figure 9 show the effect when each `sendto()` operation resolves to one IPMC address along with either 4 or 9 unicast addresses, resulting in a total of 5 and 10 send operations, respectively. The performance of point-to-point unicast met our expectations, realizing a little less than $1/r$ of the maximum number of operations per second when each application-level group is mapped to r physical addresses.

6.3 Scalability in the Number of IPMC Groups

The primary goal of MCMD is to prevent disruption when the number of multicast groups scales up. We conducted an experiment on the CUNET Emulab test bed akin to the one in section 2.4 to evaluate the amount of packet loss incurred by MCMD with a large number of groups. Nine senders in the Emulab test bed transmit 8 KB packets in a round-robin fashion to $2k$ IPMC groups. A receiver joins k of these groups and measures the number of packets received. The message rate per sender is 10,000 messages/sec, divided equally between the groups the receiver joined and the remaining ones.

In this set-up, the MCMD translation algorithm would simply merge the $2k$ groups to a single meta-group and use unicast as transport. To produce a more non-trivial group structure for MCMD, the senders also join a random subset of the k groups the receiver joined in a way that creates a mix of small and large groups. The limit of network-level IPMC groups used by the translation algorithm is set to $1,000$, including the per-node limit.

Our experiment revealed that the capacity for the hardware to handle IPMC groups appears to be around $k = 1,000$ when 2,000 groups are in the system, as seen in Figure 10. The receiver application incurred at most 5.2% packet loss by MCMD, well within the bounds of what IPMC reliability layers can handle [5]. Without MCMD, the application consistently experienced over 55% packet loss when over $4,500$ multicast groups were in the system.

7. Related Work

Jokela *et al.* recently proposed LIPSIN [18], a protocol that achieves multicast scalability by encoding forwarding state in packet headers with Bloom filters, minimizing forwarding state at intermediate routers and switches. In contrast,

MCMD is a "dirty-slate" approach to the same problem, requiring no modification to routers and switches. Earlier, Wong and Katz [32] explored the problem of multicast state minimization in inter-domain settings.

Akella *et al.* have pointed out the prevalence of packet-level redundancy in network traffic [2, 3]. Measurements show that in a trace of a data center link, 45% of packet contents were redundant [3]. We believe that a significant fraction of this observed redundancy results from point-to-point unicast transmissions used by applications to carry out multicast operations. While the de-duplication techniques used by Akella *et al.* eliminate this redundancy in the network, they still require end-hosts to perform multiple send operations where single IPMC sends might have sufficed. In addition, the proposed techniques require additional processing and storage on routers.

In an earlier version of our work [30], the optimization goal was the NP-complete problem of finding multicast groups with *zero* filtering costs while also minimizing the number of multicast groups and duplicate packets. Although solutions with zero filtering costs are desirable, later work shows that group overlaps needed for zero filtering solutions to save costs rarely arise [29].

8. Conclusion

IP Multicast can be a powerful communication option in data centers; however, scaling barriers imposed by hardware limits in routers, switches and end-host NICs have prevented its usage. MCMD multiplexes limited numbers of network-level IPMC addresses across large numbers of application-level multicast groups. The effect is to enable safe and scalable use of multicast. MCMD is completely transparent to applications, which continue to use standard IP Multicast interfaces, and does not require any modification to the network. Our evaluation shows that MCMD is scalable and robust to node failures.

Acknowledgments

We thank Tudor Marian for valuable help with producing the paper, Jure Leskovec for data and advice, Mike Spreitzer for producing the WVE trace, and Alexey Roytman, Robbert van Renesse, Hakim Weatherspoon and Hitesh Ballani for useful discussion and assistance. We are also grateful to the anonymous reviewers and our shepherd Fernando Pedone for constructive comments and suggestions to improve the paper. This work was supported in part by grants from AFOSR, AFRL, NSF, Intel Corporation and Yahoo!.

References

[1] M. Adler, Z. Ge, J. F. Kurose, D. F. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. In *ICNP*, 2001.

[2] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *SIGCOMM '08*. ACM, 2008.

[3] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: findings and implications. In *SIGMETRICS '09*, New York, NY, USA, 2009. ACM.

[4] L. Backstrom, D. P. Huttenlocher, J. M. Kleinberg, and X. Lan. Group formation in large social networks: membership, growth, and evolution. In *KDD*, pages 44–54. ACM, 2006.

[5] M. Balakrishnan, K. P. Birman, A. Phanishayee, and S. Pleisch. Ricochet: Lateral error correction for time-critical multicast. In *NSDI*. USENIX, 2007.

[6] BEA. WebLogic Server 10.3 Documentation. `http://e-docs.bea.com/wls/docs103/pdf/cluster.pdf`.

[7] K. Birman, G. Chockler, and R. van Renesse. Toward a cloud computing research agenda. *SIGACT News*, 40(2):68–80, 2009.

[8] V. Bortnikov, G. Chockler, A. Roytman, and M. Spreitzer. Bulletin board: A scalable and robust eventually consistent shared memory over a peer-to-peer overlay. In *ACM SIGOPS LADIS '09*, October 2009.

[9] B. Carmeli, G. Gershinsky, A. Harpaz, N. Naaman, H. Nelken, J. Satran, and P. Vortman. High throughput reliable message dissemination. In *SAC*. ACM, 2004.

[10] Cisco. IP Multicast Best Practices for Enterprise Customers. `http://www.cisco.com/en/US/prod/collateral/iosswrel/ps6537/ps6552/ps6592/whitepaper_c11-474791.pdf`, September 2008.

[11] D. J. Crandall, D. Cosley, D. P. Huttenlocher, J. M. Kleinberg, and S. Suri. Feedback effects between similarity and social influence in online communities. In *KDD*, pages 160–168. ACM, 2008.

[12] S. E. Deering and D. R. Cheriton. Multicast routing in datagram internetworks and extended LANs. *ACM Trans. Comput. Syst.*, 8(2):85–110, 1990.

[13] C. Diot, B. N. Levine, B. Lyles, H. Kassem, and D. Balensiefen. Deployment issues for the IP multicast service and architecture. *Network, IEEE*, 14(1):78–88, 2000.

[14] S. Floyd, V. Jacobson, C.-G. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, 1997.

[15] GEMSTONE. GemFire. `http://www.gemstone.com/products/gemfire/enterprise.php`.

[16] IBM. WebSphere. `http://www-01.ibm.com/software/webservers/appserv/was/`.

[17] JBoss. Application Server. `http://www.jboss.org/`.

[18] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander. LIPSIN: line speed publish/subscribe internetworking. In *SIGCOMM '09*, pages 195–206, New York, NY, USA, 2009. ACM.

[19] P. Judge and M. Ammar. Security issues and solutions in multicast content distribution: A survey. *IEEE Network*, 17:30–36, 2003.

[20] I. Lazar. The challenge of implementing IP multicast. *Computer Technology Review*, August 1999.

[21] J. Leskovec, L. A. Adamic, and B. A. Huberman. The dynamics of viral marketing. In *ACM EC '06*, pages 228–237, New York, NY, USA, 2006. ACM.

[22] D. Newman. 10 Gig access switches: Not just packet-pushers anymore. *Network World*, 25(12), March 2008.

[23] Oracle. Coherence 3.4 User Guide. `http://coherence.oracle.com/display/COH34UG/Coherence+User+Guide+(Full)` (accessed in July 2009).

[24] RTI. Real Time Innovations Data Distribution Service. `http://www.rti.com/products/data_distribution/`.

[25] W. Stevens, B. Fenner, and A. Rudoff. *UNIX Network Programming: The Sockets Networking API*. Addison-Wesley, 2004.

[26] TIBCO. Rendezvous. `http://www.tibco.com/software/messaging/rendezvous/default.jsp`.

[27] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In *IASTED PDCS*, 2005.

[28] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-based failure detection service. In *Middleware*, September 1998.

[29] Y. Vigfusson. *Affinity in Distributed Systems*. PhD thesis, Cornell University, 2009.

[30] Y. Vigfusson, H. Abu-Libdeh, M. Balakrishnan, K. Birman, and Y. Tock. Dr. Multicast: Rx for Datacenter Communication Scalability. In *7th ACM Symposion on Hot Topic in Networks (HotNets VII)*, October 2008.

[31] J. Widmer and M. Handley. Extending equation-based congestion control to multicast applications. In *SIGCOMM*, pages 275–285, 2001.

[32] T. Wong and R. Katz. An analysis of multicast forwarding state scalability. In *ICNP*, 2000.

[33] Yahoo! Research. Webscope groups dataset v1.0. *http://research.yahoo.com/*, 2008.