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

Ymir Vigfusson
Cornell University

Hussam Abu-Libdeh
Cornell University

Mahesh Balakrishnan
Cornell University

Ken Birman
Cornell University

Yoav Tock
IBM Haifa Research Lab

Abstract

Data centers avoid IP Multicast because of a series of problems with the technology. We propose *Dr. Multicast* (MCMD), a system that maps IPMC operations to a combination of point-to-point unicast and traditional IPMC transmissions. MCMD optimizes the use of IPMC addresses within a data center, while simultaneously respecting an administrator-specified acceptable-use policy. We argue that with the resulting range of options, IPMC no longer represents a threat and can therefore be used much more widely.

1 Introduction

As data centers scale up, IP multicast (IPMC) [15] has an obvious appeal. Publish-subscribe and data distribution layers [7, 8] generate multicast distribution patterns; IPMC permits each message to be sent using a single I/O operation, reducing latency both for senders and receivers. Clustered application servers [1, 5, 4] need to replicate state updates and heartbeats between server instances. Distributed caching infrastructures [2, 6] need to update cached information. For these and other uses, IPMC seems like a natural match.

Unfortunately, IPMC has earned a reputation as a poor citizen. Part of the problem relates to scalability; routers are stressed by the need to maintain routing state and perform costly per-group translations, and end-host NICs fail to filter messages effectively beyond a few dozen multicast addresses [23, 16]. Multicast is also perceived as an unstable technology. The very property that makes IPMC so attractive — its intrinsic asymmetry between sending and receiving rates — also makes it dangerous in the absence of regulatory mechanisms. Multicast reliability and flow control protocols are prone to 'storms' that can disrupt the entire data center. With management of multicast use practically unsupported, administrators choose to banish IPMC from their data centers.

Our paper introduces *Dr. Multicast* (MCMD), a technology that permits data center operators to selectively enable IPMC while maintaining tight control on its use. Applications are coded against the standard IPMC socket interface, but IPMC system calls are intercepted and each group is translated to a set of unicast and IPMC addresses. This translation 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.

The choice of translation is determined by acceptable-use policies designed to prevent multicast instability as well as to optimize IPMC usage. MCMD allows administrators to define policies that dictate access control and IPMC usage rules for groups and nodes in the data center. In accordance with these policies, MCMD computes the best allocation of IPMC addresses to groups (or to overlapping sets of groups), adapting as usage patterns change over time. MCMD tracks membership and distributes translations to senders using a gossip-based control plane that's robust, timely, and scalable in the number of groups in the system.

Users benefit from MCMD in several ways:

- **Policy:** Administrators can centrally impose traffic policies within the data center, such as limiting the use of IPMC to certain machines, placing a cap on the number of IPMC groups in the system, or eliminating IPMC entirely.
- **Performance:** MCMD approximates the performance of IPMC, using it directly where possible.
- **Transparency and Ease-of-Use:** Applications express their intended communication pattern using standard IPMC interfaces, rather than using hand-coded implementations of what is really an administrative policy.

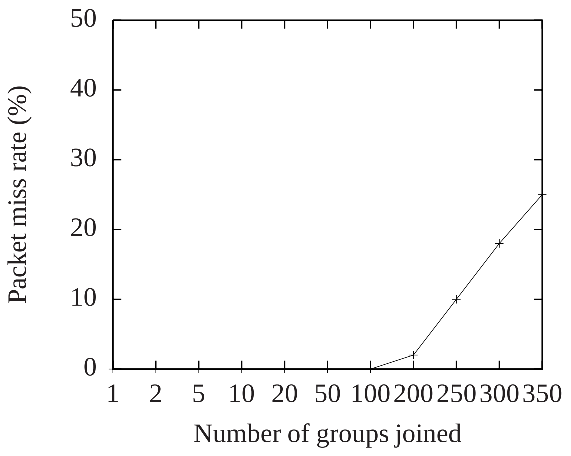

Figure 1: Receiver packet miss rate vs. number of IPMC groups joined

We will consider the problems of IPMC in data centers in the following section. The acceptable-use policy primitives and architecture of MCMD are discussed respectively in sections 3 and 4. We formalize the optimization problem of allocating the limited number of IPMC addresses, and provide and evaluate an effective heuristic for solving it in section 5. We experimentally evaluate components of MCMD in section 6, and discuss related work in section 7. Section 9 concludes.

2 IPMC in the Data Center

Modern data centers often have policies legislating against the use of IPMC, despite multicast being the natural expression of a common data communication pattern seen in a wide range of applications. This reflects a number of pragmatic considerations. First, IPMC is perceived as a potentially costly technology in terms of performance impact on the routing and NIC hardware. Second, applications using IPMC are famously unstable, running smoothly in some settings and yet, as scale is increased, potentially collapsing into chaotic multicast storms that disrupt even non-IPMC users. We know of several large Internet sites and vendors where IPMC was disabled after experiencing major disruptions due to storms.

Routers, switches and end-host NICs do not scale to large numbers of groups. For example, a typical NIC maintains a set of Ethernet mappings for joined groups and forwards packets to the kernel only if the destination group maps to one of these Ethernet addresses. Multicast IP addresses are mapped to Ethernet addresses by placing the low-order 23 bits of the IP address into the low-order 23 bits of the Ethernet address; as a result, more than one IP address can map to an Ethernet address [15]. With large numbers of groups, the NIC may accept undesired packets that the kernel must then discard.

Figure 1 illustrates the issue. In this experiment, a multicast transmitter transmits on $2k$ multicast groups, whereas a single receiver listens to k multicast groups. We varied the number of multicast groups k and measured the packet loss at the receiver. The transmitter transmits 8,000 byte packets at a constant rate of 15,000 packets/sec, spread across all the groups. The receiver thus expects to receive half of that, i.e. 7,500 packets/sec. The receiver and transmitter have 1Gbps NICs and are connected by a switch with IP routing capabilities. The experiments were conducted on a pair of single core Intel® Xeon™ 2.6GHz machines. The figure shows that the critical threshold that the particular NIC can handle is roughly 100 IPMC groups, after which throughput begins to fall off.

The performance of modern 10Gbps switches was evaluated in a recent review [22] which found that their IGMPv3 group capacity ranged between as few as 70 and 1,500. Fewer than half of the switches tested were able to support 500 multicast groups under stress without flooding receivers with all multicast traffic. While future hardware may allow networks and end-hosts to support large numbers of multicast groups, existing data centers running on commodity components are constrained in this respect.

The perception that IPMC is an unstable technology is harder to demonstrate in simple experiments. Below are some common scenarios encountered in modern data center deployments:

- *Multicast Storms* — An application uses IPMC to send to large number of receivers at a substantial data rate. Some phenomenon now triggers loss. The receivers detect the loss and solicit retransmissions, but this provokes a further surge, exacerbating the original problem. A “multicast storm” ensues.

- *Multicast DoS* — An incorrectly parameterized loop results in a sender transmitting data to an IPMC group at high speeds, overloading all the receivers in the group.

- *Traffic Magnets* — A receiver in a particular cluster within the data center inadvertently subscribes to one or more high data-rate groups used by a different cluster within the data center; the resulting flood of incoming traffic saturates the bandwidth connecting this cluster to the main data center topology.

- *Scattershot Senders* — An application running on a node is supposed to transmit data to the IPMC group 239.255.0.1; however, an off-by-one programming

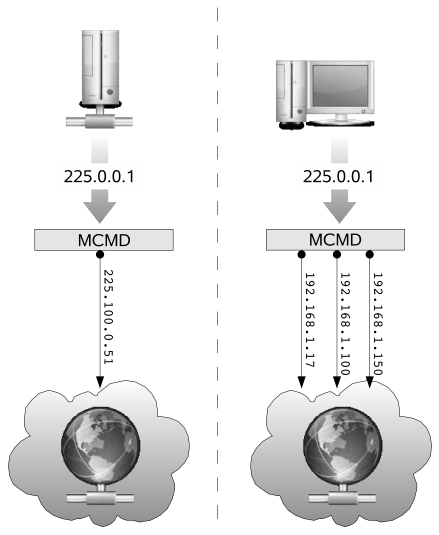

Figure 2: Two under-the-hood mappings in MCMD, a direct IPMC mapping on the left and point-to-point mapping on the right.

bug makes it send data to group 239.255.0.2 instead, spamming machines subscribed to that group with packets that need to be discarded.

The root cause of these problems is the free-for-all nature of IPMC — any machine can join or send data at any speed to any group in the system. IPMC provides absolutely no regulatory mechanisms for multicast usage.

3 Acceptable-Use Policy

The basic operation of MCMD is simple. It translates an application-level multicast address used by an application to a set of unicast addresses and network-level multicast addresses, as shown in figure 2. The translation is governed by an acceptable-use policy for the data center as defined by the system administrator.

In this section we describe the policy primitives supported by MCMD, and demonstrate how scalability and stability concerns can be mitigated by constructing a high-level acceptable-use policy made from those building blocks.

3.1 Policy Primitives

We use the following notation while describing the primitive operations:

- Logical or Application-level groups by upper-case letters: A, B, C ...
- Physical or Network-level groups by lower-case letters: a, b, c ...

An arbitrary node is denoted by the letter n. If the physical group a is included in the set of unicast and multicast addresses that a logical group A is translated into by MCMD, we say that the physical group a is a *transport* for the logical group A.

In reality, identifiers for both logical and physical groups are independently drawn from the set of class D IP addresses. For convenience, we assume that the physical and logical groups represented by the same letter are mapped to the same IP address; for example, logical group A and physical group a are both identified by the IP address 239.255.0.1. In addition, while discussing unmodified IP Multicast, we ignore the existence of logical groups and deal only with nodes and physical groups.

By default, no node in the data center is allowed to send to or join any logical groups. The primitives serve the purpose of selectively allowing nodes to join and send to logical groups, as well as mandating when physical IP Multicast groups can be used as transports for logical groups.

MCMD understands a small set of primitives to specify policies:

- allow-join(n, A) — Node n is allowed to join the logical group A.
- allow-send(n, A) — Node n is allowed to send data to the logical group A.
- allow-IPMC(n, A) — Node n is allowed to use physical IP Multicast groups as transports for the logical group A.
- max-rate(n, A, X) — n is allowed to send data at a maximum rate of X KB/s to any of the physical addresses that are mapped to the logical group A.
- max-IPMC(n, M) — n is allowed to join at most M physical IP Multicast groups.
- total-IPMC(M) — A maximum of M IPMC groups can be used within the data center.

Our system enforces these policy primitives efficiently; by intercepting socket system calls and controlling the mapping from logical groups to physical addresses, it can prevent nodes from joining or sending to logical groups, as well as limit the sending rate to these groups. Further, the use of IPMC can be enabled selectively on a per-group and per-node basis. We believe that this compact set of primitives is sufficient to mitigate most if not all vulnerabilities of multicast communication within data centers.

How does the administrator determine the right set of access control permissions for nodes? One simple scheme involves mapping applications within the data center to the specific nodes they run on, and then giving those nodes allow-join or allow-send permissions to the groups used by that application. Similarly, the administrator selectively enables IPMC usage for applications by applying allow-IPMC permissions to the corresponding nodes and groups.

While MCMD accepts a list of the primitive operations as input, we expect data center administrators to use higher level tools — such as IBM's Tivoli product [3] — that allow them to define acceptable-use policies in large systems. These policies 'compile' into the lower level primitives that MCMD understands.

3.2 Policy Examples

The policies defined by the administrator resolve the stability problems of IP Multicast by implementing a form of access control for groups. In addition, they mitigate the scalability concerns of IPMC in two ways — by placing a limit on the total number of IPMC addresses in use within the data center and by each node individually, and by using these IPMC groups intelligently for large, high rate application-level groups that benefit the most.

Are the simple primitives sufficient to prevent the stability problems of IPMC? We consider the instability scenarios outlined earlier:

- Cure for the *Multicast Storm* scenario: While it is difficult to prevent unstable reliability protocols running within a group from impacting the receivers in that group, MCMD can isolate the slowdown to just that group by either disabling IPMC transports for it or placing a rate cap on their usage.

- Cure for the *Multicast DoS* scenario: By limiting the maximum rate at which any sender is allowed to transmit data to a particular group, we can prevent the scenario where a single machine accidentally launches a DoS attack on a group by sending data to it as fast as possible.

- Cure for the *Traffic Magnet* and *Scattershot Sender* scenarios: Strict access control is sufficient to prevent both these cases. Nodes can longer join or send data to arbitrary multicast groups.

4 Design and Implementation

The MCMD architecture has two components, as seen on figure 3:

- A *library* module responsible for the *mechanism* of translation. It intercepts outgoing multicast messages and instead sends them to a set of unicast and multicast destinations. This module is *stateless*.

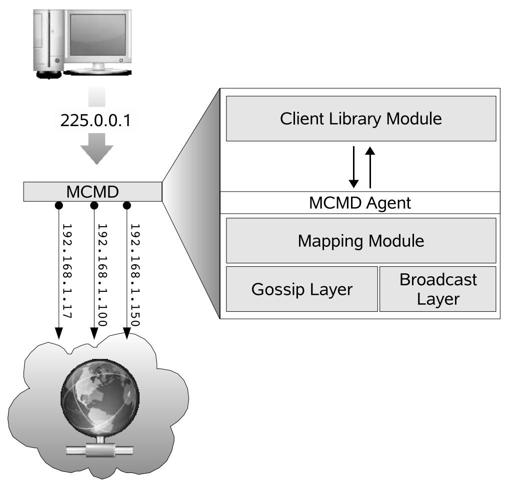

Figure 3: Overview of the MCMD architecture

- An *MCMD agent* runs as a daemon process on every node — with a single designated instance acting as a *leader* — and has two parts. **(i)** A *mapping* module responsible for the *policy* of translation. This module is *stateful*; it maintains the translation from each application-level group to a set of unicast and network-level multicast addresses. It also stores access control lists and membership/sender sets for application-level groups. **(ii)** A *control plane* that uses a combination of gossip and urgent notifications to replicate the state of the mapping module on each agent in the system.

We spend the remainder of this section discussing the design and implementation of each of these components in detail.

4.1 Library Module

Making MCMD easy to use has been the primary goal in our design. A simple interface is necessary for a seamless transition from existing multicast systems to MCMD. The library module exports a `netinet/in.h` library to applications, with interfaces identical to the standard POSIX version. By overloading the relevant socket operations, MCMD can intercept join, leave and send operations. For example:

- `setsockopt()` is overloaded so that an invocation with the IP_ADD_MEMBERSHIP or IP_DROP_MEMBERSHIP option as a parameter results in a 'join' message being sent to the mapping module. In this case, the standard behavior of `setsockopt()` – generating an IGMP message – is suppressed.

- `sendto()` is overloaded so that a send to a class D group address is intercepted and converted to multiple sends to a set of addresses. The acceptable-use policy can limit the rate of sends.

As mentioned, the library module is essentially stateless; it interacts with the mapping module via a UNIX socket to periodically pull — and cache — two pieces of state: the list of IP Multicast groups it is supposed to join, and the translations for application-level groups it wants to send data to. The library module can receive invalidation messages from the mapping module, causing it to refresh its cached entries. Simultaneously, it pushes information and statistics about grouping and traffic patterns used by the application to the mapping module. This includes an exponential-average of the message rate for the application-level group.

Additionally, the library module uses a custom multisend system call implemented in the Linux 2.6.24 kernel — a variant of the `sendto()` call that accepts a list of destinations for the message. As a result, when the application sends a message to an application-level group and the library module translates the operation into a multisend to a set of physical addresses, it can send the message to these addresses in a single efficient system call.

4.2 The MCMD Agent

The *agent* is a background daemon process running on every node in the system. Each agent instance acts as a *mapping module*, maintaining four pieces of state that are globally replicated on every agent in the system — we refer to these collectively as the *agent state*:

- Membership sets for all the nodes in the system — essentially, a map from nodes to the application-level groups they are receivers in.
- Sender sets for all the nodes in the system — a map from nodes to the application-level groups they are senders to.
- Group translations — a map from application-level groups to sets of unicast and multicast network addresses.
- Access control lists — two separate maps determining which application-level groups each node in the system is allowed to receive data in and send data to, respectively.

Each agent in the system has read-access to a locally replicated copy of the agent state. However, write-access to the agent state is strictly controlled. The first two items of the agent state can be written to only by the nodes concerned — a node can change only its own membership set or its sender set. The last two items of the agent state can be modified only by the leader agent. When an agent — leader or otherwise — writes to its local copy of the agent state, the change is propagated to other agents in the system via the control plane. Since each item in the agent state has exactly one writer, there are no conflicts over multiple concurrent updates to the agent-state.

The leader agent allocates IPMC addresses to different sets of machines in the data center, using the group membership information, sender information and access control lists in its local state to determine the best set of translations for the system. Once it writes these translations to its local state, the control plane 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. The process followed by the leader while allocating network-level IPMC resources to application-level multicast groups is the subject of section 5.

We have described a setup where each node essentially has a global view of the system; this approach has several benefits, including robustness to failure, high availability and extremely fast normal-case operation. The size of this global view is not prohibitive; for example, we can store the agent state for a 1,000-node data center within a few MB of memory.

4.3 The MCMD Control Plane

The MCMD control plane is based on a simple and powerful gossip-based failure detector identical to the one described by Van Renesse [25]. Each node maintains its own version of a global table, mapping every node in the data center 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 extremely 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.

When a node notices that the time-stamp value for some other node in its map is older than T_1 seconds, it flags that node as 'dead'. It does not immediately delete the entry, but instead maintains it in a dead state for T_2 more seconds. This is to prevent the case where a deleted entry is reintroduced into its map by some other node. After T_2 seconds have elapsed, the entry is truly deleted.

The comparison of maps between two gossiping nodes is highly optimized. The initiating node sends the other node 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 simple 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.

Crucially, *the failure detector can be used as a general purpose gossip communication layer.* Nodes can insert arbitrary state into their entries to gossip about, not just heartbeat timestamps. For example, a node could insert the average CPU load or the amount of disk space available — or, more relevantly, its agent state — and eventually this information propagates to all other nodes in the system. The reconciliation of entries during gossip exchanges is still done based on which entry has the highest heartbeat, since that determines the staleness of all the other information included in that entry.

Using a gossip-based failure detector as a replication layer for agent state has many benefits. It provides resilience and robustness for agent state, eliminating any single points of failure. It provides clean semantics for data consistency – a node can write only to its own entry, eliminating any chance of concurrent conflicting writes. In addition, a node's entry is deleted throughout the system if the node fails, allowing for fate sharing between a node and the information it inserts into the system. For example, when a node fails, its membership and sender sets in the agent state are automatically garbage-collected.

Urgent Notifications

The Achilles heel of gossip at large system sizes is *latency* — the time it takes for an update to propagate to every node in the system. To mitigate these propagation delays, MCMD uses urgent notifications in three cases: when a new receiver joins an application-level group, when a new sender starts transmitting to an application-level group, and when the leader agent generates a new translation for an application-level group.

- When a new receiver joins a group, its agent updates the local version of agent state and simultaneously sends unicast notifications to every node that is listed in the agent state as a sender to that group. As a result, senders that are using multi-send unicast to transmit data to the group 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.

- 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.

- 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.

Importantly, the first two cases involve a single unicast exchange with the leader, imposing load on it that increases linearly with the level of churn in the system. The task of updating other interested nodes in the system is delegated to the node that caused the churn event in the first place; this ensures that nodes can only disrupt themselves by changing membership and sender sets at a high rate.

5 Theoretical Considerations

We now explain how MCMD maps network-level IP multicast addresses to application-level groups. Informally, given an acceptable-use policy and complete information about membership of receivers in application-level groups, the problem is the following.

- Translate each application-level group into a minimal number of physical addresses (multicast and unicast) while respecting the policy constraints. Moreover, multiple application-level groups with identical membership can be assigned to use the same physical IPMC group, which we refer to as *collapsing* subscription overlaps.

- Minimize the bandwidth overhead on the network. Network overhead can be reduced by allocating physical IPMC addresses to larger logical groups or those with high message rates instead of smaller or less active ones.

Furthermore, we wish to find a solution that optimizes the use of IPMC resources.

We formalize this as a multi-objective optimization question that has two parts: **(i)** collapse subscription overlaps into as few groups as possible, and **(ii)** assign physical IPMC to the collapsed groups as to minimize bandwidth overhead on the network. Feasible solutions must obey the limits and constraints specified by the policy. We show that the optimization problem is NP-complete in the general case, even if we only consider the collapsing of subscriptions. A simple heuristic for the first part of the problem is to find duplicate groups, and merge them. To evaluate this approach, we contemplate what kinds of subscription patterns might arise in a data center. We don't expect this simple method to perform well if subscriptions are uniformly random, or driven entirely by human interests. However, a trace from a product used in many data centers suggests that real-world use of multicast produces numerous duplicate groups, a case in which this basic heuristic performs well. The second part of our optimization algorithm assigns IPMC resources to the collapsed groups to minimize bandwidth overhead in an optimal fashion.

5.1 Model

We begin by discussing the policy constraints and multicast scalability concerns, and then formalize the observations into an optimization problem. In section 2 we highlighted some IPMC scalability issues which are addressed by the policy primitives from section 3. Specifically:

- total-IPMC(M_{IPMC}): the total number of network-level IPMC groups used. A large number of groups adversely affects the performance of routers and switches.
- max-IPMC(n, M_n): the number of IPMC groups a receiver n can join effectively, that is, without causing the NIC to go into promiscuous mode.

For convenience, we will use M_{IPMC} and M_n to refer to these limits.

The following factors also affect multicast scalability.

- The amount of filtering the receivers are required to process, that is, the amount of redundant traffic they receive.
- The number of duplicates transmissions required to deliver messages to all intended recipients. This also corresponds to the extra bandwidth incurred on the network.

The relative importance of these factors depends on whether CPU overhead due to filtering outweighs the cost of extra bandwidth caused by duplicate transmissions.

As an example, consider a pair of partially overlapping groups A and B, where A has a high rate of traffic but B has low rate. If we merge A and B, we would use only one physical multicast group and alleviate duplicate transmissions from transmitters altogether, but at the cost of members of $B - A$ having to filter out high rates of traffic destined for A.

In what follows we assume that receivers do not perform *any* filtering for redundant or irrelevant packets. Our reasoning is twofold.

- Imposing CPU loads on receivers can have unanticipated consequences and potentially cause more trouble than our system solves.
- As we will see, the problem of collapsing subscription overlaps to minimize the number of groups is already hard. The problem of merging groups with similar subscription patterns to minimize duplicate transmissions is even more general [9].

Furthermore, we will argue that a crucial opportunity for reducing duplicate transmissions arises in replicated components of multicast applications, an opportunity that our heuristic exploits. We document the literature that assumes filtering in section 7.

5.2 Problem Statement

Let L denote the set of application-level (logical) multicast groups, and set $K = |L|$. Let us assume that the message transmission rate on logical group $k \in L$ is λ_k messages per second, $k \in L$, and $\lambda = [\lambda_1, \cdots, \lambda_K]$.

Let P denote the set of processes, and $N = |P|$. Each process n contributes a binary "subscription vector" of length K, where a 1 in the k^{th} position denotes the process receives traffic from logical group k.

Let us define the "subscription matrix" $W = (w_{nk})$, $k \in L$, $n \in P$, the rows of which are the processes' subscription vectors:

$$w_{nk} = \begin{cases} 1 & \text{process } n \text{ subscribes to logical group } k \\ 0 & \text{otherwise} \end{cases}$$

We would like to assign each logical group k to one or more multicast groups, some of which could then be assigned to use physical IPMC and others made to use point-to-point unicast. Let us denote the set of multicast groups by G, with $|G| = M$. We will determine the transport vector $T = (t_m)$, $m \in G$, defined as:

$$t_m = \begin{cases} 1 & \text{if group } m \text{ should use physical IPMC} \\ 0 & \text{if } m \text{ should use point-to-point unicast} \end{cases}$$

The logical group to multicast group mapping matrix, $X = (x_{km})$, $k \in L$, $m \in G$, is defined as:

$$x_{km} = \begin{cases} 1 & \text{logical group } k \text{ is mapped to } m \\ 0 & \text{otherwise} \end{cases}$$

The group listening matrix, $Z = (z_{nm})$, $n \in P, m \in G$, specifies to which multicast groups each process must join

$$z_{nm} = \begin{cases} 1 & \text{process } n \text{ should join multicast group } m \\ 0 & \text{otherwise} \end{cases}$$

As argued previously, we require that each process receives *exactly* what it needs, that is no filtering cost is incurred. Moreover, we require that each process joins no more than M_n multicast groups (i.e., $\sum_{m \in G} z_{nm} \leq M_n$, $\forall n \in P$), as specified by the max-IPMC policy primitive. We minimize the number of multicast groups M necessary to reach these goals, and end up with the following multi-objective optimization problem.

Definition (Exact Tiling)**.** *Find a set of mappings X, Z and a transport vector $T = (t_m)_{m \in G}$ such that:*

$$\min_{X,Z} M \tag{1}$$

$$\min_{X,Z,T} \sum_{m \in G} \sum_{k \in L} x_{km} \lambda_k \left(t_m + (1 - t_m) \sum_{n \in P} z_{nm} \right) \tag{2}$$

subject to the constraints:

$$\sum_{m \in G} z_{nm} \cdot x_{km} - w_{nk} = 0, \quad \forall n \in P, \forall k \in L \tag{3}$$

$$\sum_{m \in G} t_m \leq M_{IPMC} \tag{4}$$

$$\sum_{m \in G} z_{nm} \leq M_n, \quad \forall n \in P \tag{5}$$

Objectives 1 and 2 reflect our primary and secondary goals of respectively minimizing the total number of multicast groups used, and the number of duplicate transmissions (equivalent to minimizing bandwidth), both in collapsed groups mapped to use physical IPMC and for point-to-point unicast as defined by T. The order of the objectives reflects the priority assigned to them. Inequalities 3 and 5 reflect the zero filtering and NIC capacity constraints, respectively. Inequality 4 makes sure that no more than M_{IPMC} physical IPMC groups are used in the network.

We discuss the two objectives of the Exact Tiling optimization problem separately.

- The first objective minimizes the number of groups, which involves collapsing overlaps in the subscriptions patterns. We show that this problem is hard in the general case, devise a simple algorithm for finding exact overlaps and discuss how well this heuristic would do in data centers by looking at models and a real-world trace.
- The second objective deals with determining which collapsed groups should be allotted the potentially sparse IPMC addresses so that network overhead is minimized.

5.3 Collapsing Subscription Overlaps

To minimize the total number of multicast groups, the primary objective of the optimization problem, we must determine and collapse subscription overlaps between groups or users into the subscription patterns. If a number of users all subscribe to the same set of topics, they could all be assigned to use the same multicast group.

Theorem. *The Exact Tiling problem is NP-complete, even without the second objective.*

Proof. The MINIMUM NORMAL SET BASIS (MNSB) problem is stated as follows. Given a finite set A, and $S = \{S_1, \ldots, S_l\}$ with $S_i \subseteq A$ for $1 \leq i \leq l$, find a minimal collection B of subsets in A such that for each $1 \leq i \leq l$, S_i equals the union of a pairwise disjoint sub-collection in B.

We will show that MNSB $\leq_P$ Exact Tiling. The feasibility of an input can be checked in polynomial time easily by verifying the constraints, so Exact Tiling $\in NP$. Take a finite set A and $S = \{S_1, \ldots, S_l\}$ with $S_i \subseteq A$ for $1 \leq i \leq l$. We will imagine we have l processes, and A represents the set of logical groups. Let $P = \{1, 2, \ldots, l\}$ be l processes such that process n is a member of the logical groups in S_n, formally

$$w_{ik} = \begin{cases} 1 & \text{if } k \in S_i \\ 0 & \text{otherwise} \end{cases}$$

Set $\lambda_i = 1$ for all $i \in P$, and $M_i = \infty$ for all $i \in P$.

Consider the output of Exact Tiling on this instance, namely a set of groups G and mappings x_{km} and z_{nm} such that $\sum_{m \in G} z_{nm} x_{km} = w_{nk}$ for all $n \in P$ and $k \in A$, with $M = |G|$ minimized.

Let $B_m = \{k \in A \mid x_{km} = 1\}$ for each $m \in G$. We claim that $B = \{B_m \mid m \in G\}$ is a minimal collection of subsets of A such that for each $n \in P$, S_n is the union of a pairwise disjoint sub-collection of B.

First, B is indeed such a collection. For any $n \in P$, we claim that $\bigcup_{m \in G \mid z_{nm}=1} B_m = S_n$. The expression on the left hand side equals the set of $k \in A$ such that $x_{km} = z_{nm} = 1$ or equivalently $x_{km} z_{nm} = 1$ for some $m \in G$. By constraint 3 this is equivalent to the set of $k \in A$ such that $w_{nk} = 1$, which is exactly S_n.

Second, suppose B is not minimal and B' is a smaller collection satisfying the above conditions. Then the mappings x_{km} and z_{nm} to the appropriate subsets in B' are a feasible solution to the Exact Tiling problem that uses fewer than $|B| = M$ groups, contradicting our primary objective 1.

Since MNSB is NP-complete [18], and the reduction takes polynomial time, we have established that Exact Tiling is an NP-complete problem. $\square$

This result is not too surprising. Instead of searching for an optimal algorithm that is likely to have exponential running time, we consider the following simple heuristic we call MERGE-DUPLICATES: For each logical group m, create a corresponding multicast group m' with the same members. For each pair of multicast groups m_1 and m'_2, merge m'_1 and m'_2 into m'_1 if they have identical member sets. The mapping sets X, Z follow trivially.

We would like to determine the *expected* performance of MERGE-DUPLICATES on multicast use in real-world data centers. Due to the lack of publicly available trace data-sets we must contemplate what these inputs might look like. We will enumerate three types of models here.

Uniform Random Subscriptions

A basic idea for generating subscription patterns is to randomly connect processes to logical groups. Consider the following model. Assume there are n processes in a system. For each of n logical groups, we pick a number k uniformly at random between 1 and n, and then let random subset of k processes subscribe to this logical group. Then MERGE-DUPLICATES will have almost no opportunity for collapsing subscription overlaps, as the following theorem shows:

Theorem. *The expected number of exact overlaps is a constant less than 8 independent of n.*

Proof. We start by proving that $k^2 \leq \binom{n}{k}$ for $k \leq \frac{n}{2}$ and $n \geq 7$ by induction. The base case when $n = 7$ follows from inspection. Assume $k^2 \leq \binom{n}{k}$ for some $n \geq 7$ and $k \leq \frac{n}{2}$.

If $k < \frac{n+1}{2}$ then $k^2 \leq \binom{n}{k} = \binom{n+1}{k} - \binom{n}{k-1} \leq \binom{n+1}{k}$, with the convention that $\binom{n}{-1} = 0$. If $k = \frac{n+1}{2}$ then $k^2 \leq k^2 + (k^2 - 4k + 2) = 2(k-1)^2 \leq 2\binom{n}{k-1} = \binom{n}{k} + \binom{n}{k-1} = \binom{n+1}{k}$. Thus the statement holds true by induction.

Let N_j denote the size of logical group j. Let A_{ij} indicate event that logical groups i and j overlap exactly, in which case $A_{ij} = 1$ and 0 otherwise. We wish to bound the expected number of exactly overlapping logical groups, $A = \sum_{i<j} A_{ij}$.

Clearly, $\mathbb{E}[A_{ij} \mid N_i \neq N_j] = 0$. We have

$$\mathbb{E}[A_{ij} \mid N_i = N_j = k] = \binom{n}{k}^{-1}.$$

Using the argument above, if $n \geq 7$ then $\mathbb{E}[A_{ij} \mid N_i = N_j = k] \leq \frac{1}{k^2}$ for $k \leq \frac{n}{2}$, and $\mathbb{E}[A_{ij} \mid N_i = N_j = k] \leq \frac{1}{(n-k)^2}$ for $k \geq \frac{n}{2}$.

When the number of logical groups is $m \leq \alpha n$ for some constant α, we get

$$\begin{aligned}
\mathbb{E}[A] &= \sum_{i<j} \mathbb{E}[A_{ij}] \\
&= \sum_{i<j} \sum_{k=1}^{n} \mathbb{E}[A_{ij} \mid N_i = N_j = k] \mathbb{P}[N_i = N_j = k] \\
&\leq \sum_{i<j} 2 \sum_{k=1}^{\lceil n/2 \rceil} \mathbb{E}[A_{ij} \mid N_i = N_j = k] \frac{1}{n^2} \\
&\leq \sum_{i<j} 2 \left(6 + \sum_{k=7}^{\lceil n/2 \rceil} \frac{1}{k^2} \right) \frac{1}{n^2} \\
&< \frac{2}{n^2} \binom{m}{2} \left(6 + \frac{\pi^2}{6} \right) < 8\alpha^2.
\end{aligned}$$

In particular, we have $\mathbb{E}[A] < 8$ when $m \leq n$. $\square$

Human Preferences

The subscriptions to logical groups of a multicast system could be driven by human interest, for instance in a dissemination system for stock trades in a financial data center [24] or by disseminating RSS feeds [20]. If multicast subscriptions were entirely influenced by human preferences, the body of the distribution showing the number of subscribers to each logical group would follow a power-law [14, 20], as we have seen in multiple traces of distributions of human interest preferences.

It is still an open question whether one can craft a generative model for interest similarity between humans. In attempt to answer this question, we have constructed several models of varying complexity that we are currently working on validating. Our preliminary results suggest that exact overlaps of topics of interests between humans are rare.

Replication of Components

Here we describe an important case in which the MERGE-DUPLICATES heuristic performs well.

If we consider the applications running in the data center, we find that they are often componentized: a single application is actually constructed from multiple interacting components. For example, to build a web page, a front-end component may issue requests to a set of back-end components. Such structures have important implications for subscription overlaps. In particular, the componentized application development model will tend to confer structure on the underlying communication subsystem, because the components will often have identical patterns of replication.

Furthermore, operators in large-scale data centers may deploy services automatically. The automation mechanisms create regularities. For example, if services X, Y and Z often interact, they might be deployed as a unit onto the same nodes: where one finds a replica of X (and the associated logical groups), one would also find replicas of Y and Z (and their associated groups). Collocation of services can also be useful for performance reasons, for instance to place a cache service near its clients.

We have an example of a major application in which such mechanisms produce heavy overlaps. In a trace showing the subscription patterns within an internal publish-subscribe component of IBM's Websphere [4], there are 79 processes subscribed to over 6,600 logical multicast groups. On average, 10 processes subscribe to each logical group. Importantly, there are only *314* distinct logical groups with respect to membership, or less than 5% of the total number of groups. From our experience with the use of multicast within data centers, this result is typical. The MERGE-DUPLICATES heuristic successfully collapses all these duplicates, showing a trivial yet very important opportunity to make efficient use of the IPMC resources of the data center.

5.4 Allocating IPMC Resources

Assume subscriptions overlaps have been collapsed by MERGE-DUPLICATES or some other algorithm. The second objective of the optimization question asks *which* of the collapsed groups should be allotted the limited number physical IPMC groups addresses in the system, and which should use point-to-point unicast such that network overhead is minimized. Given the set of collapsed groups, this can be done optimally.

Definition. *Let*

$$\gamma_m = \sum_{k\in L} x_{km}\lambda_k \left(\sum_{n\in P} z_{nm} - 1\right)$$

for $m \in G$ *represent the* broadcast overhead *of group* m.

In essence, γ_m denotes the extra network bandwidth required for doing transmissions to m via unicast instead of as a network-level multicast.

Let X, Z be a feasible solution to the Exact Tiling program without the secondary objective (2) or constraint (4). Let $m_1, \ldots, m_M$ denote the collapsed groups in G sorted by γ_m in reverse order. Define the HEAVIEST-FIRST heuristic as follows: Assign $t_{m_i} = 1$ if and only if $i \leq \min\{M, M_{\text{IPMC}}\}$. This heuristic captures the intuition that the 'expensive' groups in terms of network bandwidth should use network-level multicast.

Theorem. *The* HEAVIEST-FIRST *transport vector minimizes the second objective of the Exact Tiling optimization problem.*

Proof. (Sketch) Consider objective (2) as a potential function Φ with all t_m set to 0. Changing t_m from 0 to 1 for a group $m \in G$ reduces the value of the potential function by γ_m. By construction, T assigns $t_m = 1$ to the most expensive groups with respect to γ_m, thus minimizing Φ without violating inequality (4). □

The value of γ_m for $m \in G$ can be calculated by the agent using the group membership information along with the exponential average used to estimate λ_k for $k \in L$ that is reported by the library module.

5.5 Optimization Algorithm

The MCMD combines the MERGE-DUPLICATES and HEAVIEST-FIRST heuristics to find an approximate solution to the Exact Tiling problem. The algorithm can be summarized as follows.

- Run MERGE-DUPLICATES, such that the traffic reports for collapsed groups are aggregated.
- Run HEAVIEST-FIRST on the collapsed groups using the aggregated traffic reports.

We further optimize the resource use by noting that collapsed groups with two or fewer members that are allowed to use physical IPMC (due to the allow-IPMC or max-IPMC primitives) to any of the groups should always use point-to-point unicast.

Between runs of the algorithm, data structures can easily be updated to reflect incremental changes.

The algorithm uses hash-tables to test for overlaps in linear time, and thus has $O(K^2Q)$ complexity where Q denotes the maximum number of processes subscribed to any logical group.

6 Experimental Results

The fundamental goal of the MCMD is to allow administrators to make controlled use of IPMC to improve the performance of the multicast technology in data centers. Since MCMD is a new critical component in the data center, in this section we demonstrate that it does not create new worries for administrators. Specifically, we experimentally evaluate a prototype of MCMD on the Emulab test-bed to answer the following questions.

1. What is the overhead of running MCMD?
2. How quickly are membership or policy changes reflected in the system?
3. Does the system scale in the number of groups?

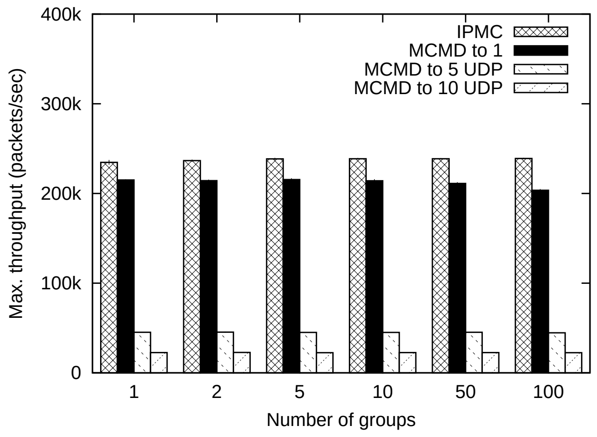

Figure 4: Maximum throughput for a sender using regular IPMC and MCMD with direct IPMC mapping, and MCMD unicast to 5 or 10 receivers per group.

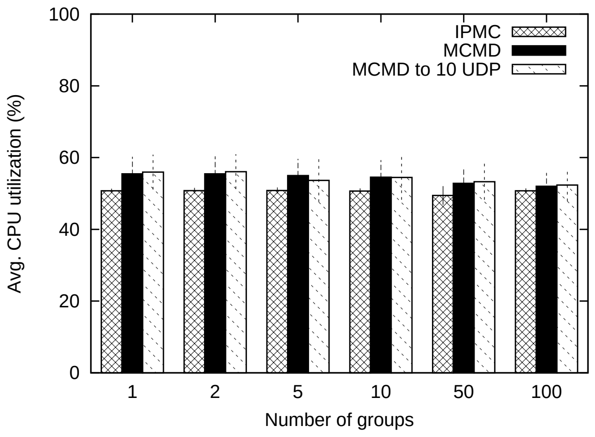

Figure 5: Average CPU utilization for the sender application with and without MCMD

We also evaluate our system in some of the bad-case scenarios outlined in section 2 by comparing it before and after policy changes. Our results suggest that MCMD provides fast and scalable control of IP multicast with negligible overhead.

6.1 Methodology

We implemented a prototype of MCMD consisting of 14,000 lines of C and C++ code, and deployed it on the Emulab testbed. All nodes had an Intel Pentium 3.0GHz processor and 1GB of RAM. Unless explicitly mentioned, the network configuration was a star topology with 100Mbps links between nodes. Each node in the testbed ran the MCMD agent, along with one of the following applications:

- A *sender application* joins k logical groups, waits for 2 seconds, then transmits 100,000 1KB packets using `sendto()` to the k groups in a round-robin fashion as fast as possible. The application can be configured to rate-limit the send to 5,000 packets/sec.

- A *receiver application* joins the same k logical groups, and waits for incoming packets in a `recv()` loop.

The rate of gossip or *epoch length* for the agent was set to 1 exchange per second, unless otherwise specified. Error bars represent one standard deviation, and are omitted if they are too small to be clearly visible.

6.2 Application Overhead

We measured the difference in maximum throughput for the sender application for varying k with and without the MCMD library. We considered both the case where MCMD maps each application-level address to a single network-level IPMC address, and also the case where each address resolves to 5 or 10 unicast addresses. Recall that in the IBM Websphere trace, the average group size was around 10.

As shown in figure 4, there is a 10-15% reduction in maximum throughput by running the sender application with MCMD over one-to-one address mapping, depending on number of logical groups. Our system supports sending over 200,000 packets per second. We also measured CPU utilization for the sender application with and without MCMD active. Figure 5 shows an increase of no more than 10% independent of the number of groups. The relative overhead increases slightly with greater number of groups due to collisions in the hash-based data structure for the look-up map in the library. The overhead was reduced by removing system calls on the critical path of the overloaded socket calls in the MCMD library, such as checking for urgent notifications, and moving those to a separate thread via `clone()` in the library.

The performance of point-to-point unicast meet our expectations, realizing approximately $1/r$ of the maximum possible throughput when each application-level group is mapped to r physical addresses.

6.3 Network Overhead

The network overhead of the MCMD protocol is shown in figure 6. In this experiment the network constitutes 16

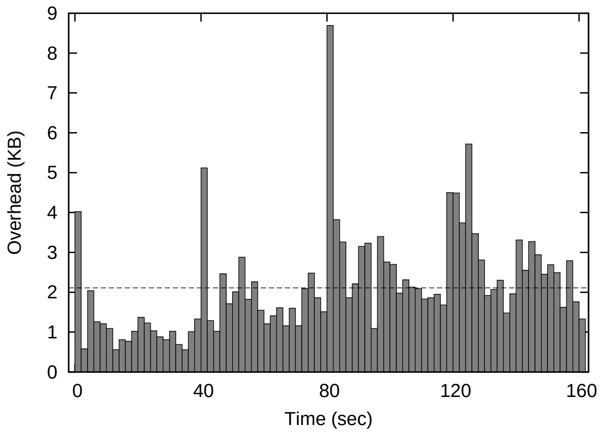

Figure 6: Network background overhead for MCMD. The dotted line denotes the 2.1KB/s average overhead.

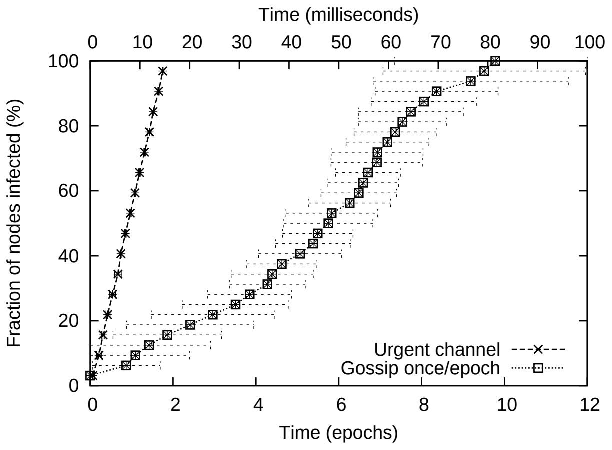

Figure 7: Latency of updates using regular gossip (epochs) and with the urgent notification channel enabled (ms).

nodes. The graph shows the amount of traffic transmitted and received by the most loaded node with respect to network traffic, namely the leader. Initially, there are 6 nodes running both a sender and receiver, joined by 5 more nodes at time 40 and 6 more at 80. At time 120, a new translation is computed by the leader, and an urgent notification is transmitted to the appropriate nodes.

By design, the gossip module in the MCMD agent produces configurable constant background traffic. At no point does MCMD control traffic exceed 11 KB/sec, even when the urgent notification channel is enabled.

6.4 Latency

We now measure the latency of updates between nodes, namely membership and mapping changes. This determines, for instance, how fast a new receiver starts receiving messages from senders, or how long it keeps receiving messages after leaving a logical group. As discussed earlier, solutions need to trade-off latency and scalability. We compare the scalable gossip control plane per se to the fast version of MCMD that deploys an urgent notification channel on top of the gossip mechanism. In figure 7 we can see how fast new updates propagate through a 32-node network with and without the urgent notification channel. In this experiment, the gossip module has propagated the update everywhere after 10 epochs, and follows the well-studied epidemic replication curve [25]. When urgent notifications are used, the latency becomes at most 15 ms round-trip time in this experiment.

6.5 Policy Control

We revisit the *Multicast DoS* scenario from section 2 in which a malfunctioning sender suddenly starts sending large amounts of traffic in a loop to a logical group, thus overloading the receivers. Consider a network of 16 nodes that are sending and receiving low rates of traffic over a single IPMC group. At time 20, one of the senders starts bombarding the group with traffic. The administrator changes the policy at time 40 to remove the faulty sender from the group. Alternatively, the administrator could have put a rate-limit on sends to this particular group.

In figure 8, we see the CPU utilization of a receiver in the group, averaged over 10 trials of running this experiment. The CPU utilization increases substantially when the DoS begins, and decreases almost instantly after the new administrative policy is issued. In effect, the sender was commanded to leave the group via an urgent notification from the leader.

We also looked at the *Traffic Magnet* scenario where an unsuspecting node in cluster B joins a high-traffic multicast group in cluster A, increasing the load on the router between the two clusters substantially. We set up an experiment where 12 nodes in A each transmit 20 KB/s to a logical group that is mapped to a network-level IPMC by MCMD. We measured the average throughput over 10 trials between two regular nodes, one in each cluster, and show the results in figure 9. At time 20, a node n in cluster B joins the IPMC group, causing the throughput between the regular nodes to plummet to about 2.5MB/s, or a 75% drop. At time 40, the administrator changes the access policy and disallows node n from belonging to the logical group, causing MCMD to make n leave the network-level IPMC group. The network has recovered 5 seconds later.

Naturally, both of these episodes could have been prevented by specifying a complete administrative policy

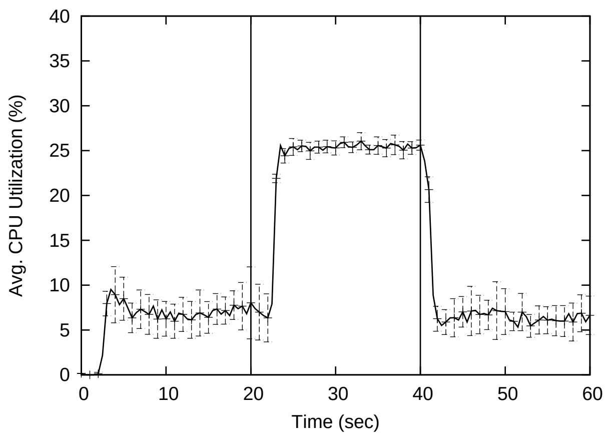

Figure 8: CPU utilization at a normal receiver. A malfunctioning node bombards the group at time 20, and the administrator restricts policy at time 40.

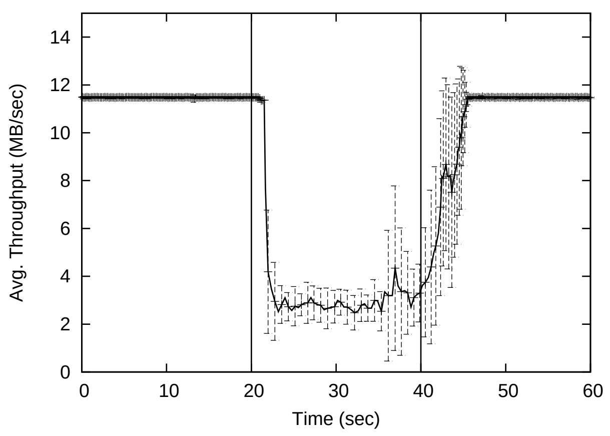

Figure 9: Average throughput between two nodes in separate subnets. At time 20, a node erroneously joins a high-traffic IPMC group in the other subnet, and the administrator corrects the access control policy at time 40.

with access restrictions and rate-limits for senders, as described in section 3.

7 Related Work

In the two decades since IP Multicast was first introduced [15], researchers have extensively examined its security, stability and scalability characteristics. Much of this work has attempted to scale and secure multicast in the wide area.

7.1 Stability and Security

Work on secure multicast has focused on achieving two properties in the wide-area: secrecy and authentication [13, 21]. Secrecy implies that only legal receivers in the group can correctly receive data sent to the group, and authentication implies that only legal senders can transmit data to the group [21]. Both these properties are typically obtained by using cryptographic keys, and much of the work in this area has focused on the task of group key management. The security issues examined by MCMD are orthogonal to this existing body of work — within a data center, we are not concerned with either secrecy or authentication. Achieving these properties would not alleviate the performance problems of IP Multicast; for instance, a sender could still spam a group with nonsense data that fails to authenticate but nevertheless overloads receivers.

Access control for multicast has been proposed before as a solution for achieving secure multicast [10, 19]; once again, this work is aimed at wide-area scenarios and focuses on the secure implementation of the access control mechanism. SSM [11] is an IP Multicast variant that allows receivers to subscribe to individual senders within multicast groups, eliminating the problem of arbitrary machines launching DoS attacks on a group.

Reliable multicast is a research sub-area in itself, and many papers have looked specifically at the stability of reliability mechanisms. SRM [17] is a well-known protocol that's known to inject massive multicast overheads in certain loss conditions, triggering further loss events at receivers [12]. MCMD operates at the routing layer and is oblivious to end-to-end reliability mechanisms, but can help mitigate the ill-effects of these protocols, as described previously.

7.2 Scalability

The scalability of IP Multicast in the number of groups in the system is limited by the space available in router tables [23]. The impact of adding IPMC state to network routers has been analyzed by Wong, Katz and McCanne [26, 27].

Prior work on *imprecise channelization* [9, 24] explores the question at the core of the MCMD optimization problem. This body of research considers the case where end-host filtering for redundant or irrelevant packets is employed, and the goal is to minimize the amount of duplicate transmissions along with the total bandwidth overhead. Cost of this form of receiver filtering can be pretty high. The channelization problem does not address the fact that end-host NIC performance degrades with large numbers of multicast groups. Thus, an optimal solution to the channelization problem may require receivers to join a large number of groups.

8 Future Work

One avenue of future research involves allowing administrators to mandate custom transports for groups — for example, using point-to-point TCP connections or overlay graphs to transfer data to receivers. We are also interested in extending MCMD to scenarios with multiple data centers, separate different policies can be defined for traffic that crosses administrative boundaries. Lastly, we are in the process of examining grouping patterns induced by other real-world applications, to develop more generalizable heuristics that MCMD can use to allocate IPMC groups.

9 Conclusion

Many major data center operators legislate against the use of IP multicast: the technology is perceived as disruptive and insecure. Yet IPMC offers attractive performance and scalability benefits. Our paper proposes Dr. Multicast (MCMD), a remedy to this conundrum. By permitting operators to define an acceptable use policy (and to modify it at runtime if needed), MCMD permits active management of multicast use. Moreover, by introducing a novel scheme for sharing scarce IPMC addresses among logical groups, MCMD can reduce the number of IPMC addresses needed sharply, and ensures that the technology is only used in situations where it offers significant benefits.

10 Acknowledgments

We thank Tudor Marian, Robbert van Renesse, Gregory Chockler and Danny Dolev for valuable input. This work was supported by grants from AFRL, AFOSR, NSF, Cisco and Intel Corporation.

References

[1] BEA Weblogic. `http://www.bea.com/framework.jsp?CNT=features.htm&FP=/content/products/weblogic/server/`, 2008.

[2] GEMSTONE GemFire. `http://www.gemstone.com/products/gemfire/enterprise.php`, 2008.

[3] IBM Tivoli. `www.ibm.com/tivoli`, 2008.

[4] IBM WebSphere. `http://www-01.ibm.com/software/webservers/appserv/was/`, 2008.

[5] JBoss Application Server. `http://www.jboss.org/`, 2008.

[6] Oracle Coherence. `http://www.oracle.com/technology/products/coherence/index.html`, 2008.

[7] Real Time Innovations Data Distribution Service. `http://www.rti.com/products/data_distribution/`, 2008.

[8] TIBCO Rendezvous. `http://www.tibco.com/software/messaging/rendezvous/default.jsp`, 2008.

[9] M. Adler, Z. Ge, J. F. Kurose, D. F. Towsley, and S. Zabele. Channelization problem in large scale data dissemination. In *ICNP*, pages 100–109. IEEE Computer Society, 2001.

[10] T. Ballardie and J. Crowcroft. Multicast-specific security threats and counter-measures. In *SNDSS'95: Proceedings of the Symposium on Network and Distributed System Security*, page 2, Washington, DC, USA, 1995. IEEE Computer Society.

[11] S. Bhattacharyya, C. Diot, and L. Giuliano. An Overview of Source-Specific Multicast (SSM). Technical report, RFC 3569, July 2003.

[12] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky. Bimodal Multicast. *ACM Transactions on Computer Systems (TOCS)*, 17(2):41–88, 1999.

[13] R. Canetti, J. Garay, G. Itkis, D. Micciancio, M. Naor, B. Pinkas, I. Center, and Y. Heights. Multicast security: a taxonomy and some efficient constructions. In *INFOCOM'99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, 1999.

[14] A. Clauset, C. R. Shalizi, and M. E. J. Newman. Power-law distributions in empirical data. `http://arxiv.org/abs/0706.1062v1`, June 2007.

[15] S. Deering. Host Extensions for IP Multicasting. *Network Working Request for Comments 1112*, August 1989.

[16] A. Fei, J. Cui, M. Gerla, and M. Faloutsos. Aggregated multicast: an approach to reduce multicast state. *Global Telecommunications Conference, 2001. GLOBECOM'01. IEEE*, 3, 2001.

[17] 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 (TON)*, 5(6):784–803, 1997.

[18] T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM Journal on Computing*, 22(6):1117–1141, 1993.

[19] P. Judge and M. Ammar. Gothic: a group access control architecture for secure multicast and anycast. In *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, 2002.

[20] H. Liu, V. Ramasubramanian, and E. G. Sirer. Client behavior and feed characteristics of RSS, a publish-subscribe system for web micronews. In *IMC '05: Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*, pages 3–3, Berkeley, CA, USA, 2005. USENIX Association.

[21] M. Moyer, J. Rao, and P. Rohatgi. A survey of security issues in multicast communications. *Network, IEEE*, 13(6):12–23, 1999.

[22] D. Newman. Multicast performance differentiates across switches. `http://www.networkworld.com/reviews/2008/032408-switch-test-performance.html`, 2008.

[23] P. Rosenzweig, M. Kadansky, and S. Hanna. The Java Reliable Multicast Service: A Reliable Multicast Library. *Sun Labs*, 1997.

[24] Y. Tock, N. Naaman, A. Harpaz, and G. Gershinsky. Hierarchical clustering of message flows in a multicast data dissemination system. In S. Q. Zheng, editor, *IASTED PDCS*, pages 320–326. IASTED/ACTA Press, 2005.

[25] R. van Renesse, Y. Minsky, and M. Hayden. A gossip-based failure detection service. In *Middleware'98, IFIP International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 55–70, England, September 1998.

[26] T. Wong and R. Katz. An analysis of multicast forwarding state scalability. In *ICNP '00: Proceedings of the 2000 International Conference on Network Protocols*, page 105, Washington, DC, USA, 2000. IEEE Computer Society.

[27] T. Wong, R. H. Katz, and S. McCanne. An evaluation on using preference clustering in large-scale multicast applications. In *INFOCOM (2)*, pages 451–460, 2000.