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* (the MCMD), a system that maps traditional IPMC operations to either use a new point-to-point UDP multisend operation, or to a traditional IPMC address. The MCMD is designed to optimize resource allocations, 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 We argue that with the resulting range of options, IPMC no longer represents a threat and could therefore be used much more widely.

1 Introduction

As data centers scale up, IP multicast (IPMC) [8] has an obvious appeal. Publish-subscribe and data distribution layers [6, 7] generate multicast distribution patterns; IPMC permits each message to be sent using a single I/O operation, reducing latency both for senders and receivers (especially, for the last receiver in a large group). Clustered application servers [1, 4, 3] need to replicate state updates and heartbeats between server instances. Distributed caching infrastructures [2, 5] 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. Routers must maintain routing state and perform a costly per-group translation[11, 9]. Many NICs can only handle a few IPMC addresses; costs soar if too many are used. Multicast flow control is also a black art. When things go awry, a multicast storm can occur, disrupting the whole data center. Perhaps most importantly, management of multicast use is practically unsupported.

Our paper introduces *Dr. Multicast* (the MCMD), a technology that permits data center operators to 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 mapped into one of two cases:

- A true IPMC address is allocated to the group.
- Communication to the group is performed using point-to-point UDP messages to individual receivers, using a new *multi-send* system call.

The MCMD tracks group membership, using a gossip protocol. It translates each send operation on a multicast group into one or more send operations, optimized for system objectives. Finally, to implement this optimization policy, it instantiates in a fault-tolerant fashion a service that computes the best allocation of IPMC addresses to groups (or to overlapping sets of groups), adapting as use changes over time.

Users benefit 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: The MCMD approximates the performance of IPMC, using it directly where possible. When a multicast request must be translated into UDP sends, the multi-send system call reduces overheads.
- 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.
- Robustness: The MCMD is implemented as a distributed, fault-tolerant service.

We provide and evaluate effective heuristics for the optimization problem of allocating the limited number of IPMC addresses, although brevity limits us to a very terse review of the framework, the underlying (NP-complete) optimization question, and the models used in the evaluation.

*This work was supported by grants from AFRL, AFOSR, NSF, Cisco and Intel.

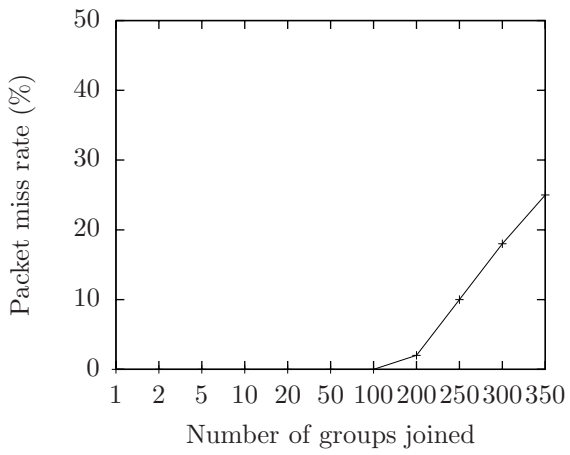

Figure 1: Receiver packet miss rate vs. number of IPMC groups joined

2 IPMC in the Data Center

Modern data centers often have policies legislating against the use of IPMC, despite the fact that multicast is a 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.

The hardware issue relates to imperfect filtering. A common scheme used to map IP group addresses to Ethernet group addresses involves placing the low-order 23 bits of the IP address into the low-order 23 bits of the Ethernet address [8]. Since there are 28 significant bits in the IP address, more than one IP address can map to an Ethernet address. The NIC maintains the set of Ethernet mappings for joined groups and forwards packets to the kernel only if the destination group maps to one of those Ethernet addresses. As a result, with large numbers of groups, the NIC may accept undesired packets, which the kernel must discard.

Figure 1 illustrates the issue. In this experiment, a multicast transmitter transmits on $2k$ multicast groups, whereas the receiver listens to k multicast groups. We varied the number of multicast groups k and measured the CPU consumption as well as the packet loss at the receiver. The transmitter transmits at a constant rate of 15,000 packets/sec, with a packet size of 8,000 bytes 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. Figure 1 shows that the critical threshold that the particular NIC can handle is roughly 100 IPMC groups, after which throughput begins to fall off.

The issue isn't confined to the NIC. Performance of modern 10Gbps switches was evaluated in a recent review [10] which found that their IGMPv3 group capacity ranged between as little as 70 and 1,500. Less than half of the switches tested were able to support 500 multicast groups under stress without flooding receivers with all multicast traffic.

The MCMD addresses these problems in two ways. First, by letting the operator limit the number of IPMC addresses in use, the system ensures that whatever the limits in the data center may be, they will not be exceeded. Second, by optimizing to use IPMC addresses as efficiently as possible, the MCMD arranges that the IPMC addresses actually used will be valuable ones – large groups that receive high traffic. As seen below, this is done not just by optimizing across the groups as given, but also by discovering ways to aggregate overlapping groups into structures within which IPMC addresses are shared by multiple groups, permitting even greater efficiencies.

The perception that IPMC is an unstable technology is harder to demonstrate in simple experiments: as noted earlier, many applications are perfectly stable under most patterns of load and scale, yet capable of being extraordinarily disruptive. The story often runs something like this. An application uses IPMC to send to large numbers 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 load surge, exacerbating the original problem. A multicast storm ensues, saturating the network with redundant retransmission requests and duplicative multicasts. With MCMD the operator can safely deploy such an application: if it works well, it will be permitted to use IPMC; if it becomes problematic, it can be mapped to UDP merely by changing the acceptable use policy. More broadly, the MCMD encourages developers to express intent in a higher-level form, rather than hand-coding what is essentially an administrative policy.

3 Design

The basic operation of MCMD is simple. It translates an application-level multicast address used by

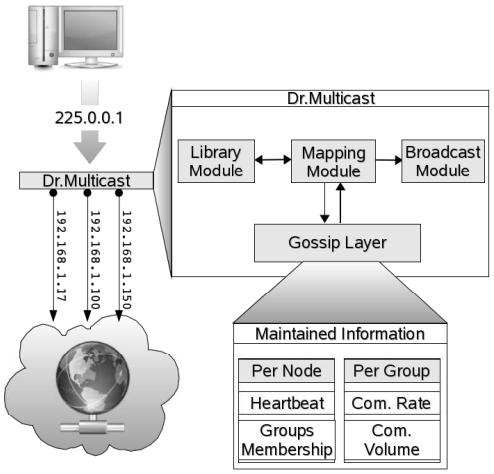

Figure 2: Overview of the MCMD architecture

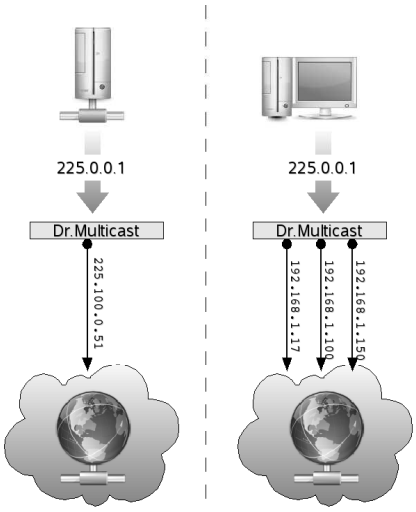

Figure 3: Two under-the-hood mappings in MCMD, a direct IPMC mapping on the left and point-to-point mapping on the right.

an application to a set of unicast addresses and network-level multicast addresses. MCMD has two components (see figure 2):

- 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.
- A *mapping* module responsible for the *policy* of translation. It determines the mapping from each application-level address and a set of unicast and network-level multicast addresses.

3.1 Library Module

The MCMD library module exports a `<sockets.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 from the kernel.

The library module interacts with the mapping module via a UNIX socket. It pulls the translations for each application-level group from the mapping module. Simultaneously, it pushes information and statistics about grouping and traffic patterns used by the application to the local mapping module.

3.2 Mapping Module

The mapping module plays two important roles:

- It acts as a Group Membership Service (GMS), maintaining the membership set of each application-level group in the system.
- It allocates a limited set of IPMC addresses to different sets of machines in the data center and orchestrates the IGMP joins and leaves required to maintain these IPMC groups within the network.

The mapping module uses a gossip-based control plane using techniques described in [13]. The gossip control plane is extremely resilient to failures and includes a decentralized failure detector that can be used to locate and eject faulty, i.e. irresponsive, machines. It imposes a stable and constant overhead on the system and has no central bottleneck, irrespective of the number of nodes.

The gossip-based control plane essentially replicates mapping and grouping information slowly and continuously throughout the system. As a result, the mapping module on any single node has a global view of the system and can immediately resolve an application-level address to a set of unicast and multicast addresses without any extra communication. The size of this global view is not prohibitive; for example, we can store membership and mapping information for a 1000-node data center within a few

MB of memory. For now, we're targetting systems with a low enough rate of joins, leaves and failures per second to allow for global replication of control information. In the future, we'll replace the global replication scheme with a more focused one to eliminate this restriction.

Knowledge of global group membership is sufficient for the mapping module at each node to translate application-level group addresses into network-level unicast addresses. To fulfill the second function of allocating IPMC addresses in the system, an instance of the specific mapping module running on a particular node in the system acts as a leader. It aggregates information from other mapping modules (via the gossip control plane) and calculates appropriate allocations of IPMC addresses to mandate within the data center. The leader can be chosen according to different strategies – one simple expedient is to query the gossip layer for the oldest node in the system. The failure of the leader is automatically detected by the gossip layer's inbuilt failure detector, which also naturally updates the pointer to the oldest node.

3.2.1 Gossip Control Plane

We first describe an implementation of the mapping module using only the gossip-based control plane. However, 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. Consequently, we then describe approaches to add extra control traffic for certain kinds of critical updates – in particular, IPMC mappings and group joins – that need to be distributed through the system at low latencies.

Gossip-based Failure Detector: The MCMD control plane is a simple and powerful gossip-based failure detector identical to the one described by van Renesse [13]. Each node maintains its own version of a global table, mapping every node in the data center to a timestamp 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 timestamp 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 timestamp 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.

Gossip-based Communication: Thus far, we have described a decentralized gossip-based failure detector. Significantly, such a 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; 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 control communication layer has many benefits. It provides extreme resilience and robustness for control traffic, eliminating any single points of failure. It provides extremely 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.

Group Membership Service: The mapping module uses the gossip layer to maintain group membership information for different application-level groups in the system. Each node maintains in its gossip entry – along with its heartbeat timestamp – the set of groups it belongs to, updating this whenever the library module intercepts a join or a leave. A simple scan of the map is sufficient to generate an alternative representation of the membership information, mapping each group in the system

to all the nodes that belong to it. If a node fails, its entry is removed from the gossip map; as a result, a subsequent scan of the map generates a groups-to-nodes table that excludes the node from all the groups it belonged to.

Mapping Module Leader: As mentioned previously, the gossip layer informs the mapping module of the identity of the oldest node in the system, which is then elected as a leader and allocates IPMC addresses. To distribute these allocations back into the system, the leader can just update its own entry in the gossip map with the extra IPMC information. When a receiver is informed of a relevant new mapping, it issues the appropriate IGMP messages required to join or leave the IPMC group as mandated by the mapping module.

A "pure" gossip protocol can have large propagation delays, resulting in undesirable effects such as senders transmitting to IPMC groups before receivers can join them. To mitigate these latency effects, the leader periodically broadcasts mappings at a fixed, low rate to the entire data center. The rate of these broadcasts is tunable; we expect typical values to be a few packets every second. The broadcast acts purely as a latency optimization over the gossip layer; if a broadcast message is lost at a node, the mapping is eventually delivered to it via gossip.

Latency Optimization of Joins: We are also interested in minimizing the latency of a join to an application-level multicast group; i.e., after a node issues a join request to a group, how much time elapses before it receives data from all the senders to that group? While the gossip layer will eventually update senders of the new membership of the group, its latency may be too high to support applications that need fast membership operations. The latency of leave operations is less critical, since a receiver that has left a group can filter out messages arriving in that group from senders who have stale membership information until the gossip layer propagates the change.

In MCMD, we explore two options to speed up joins. The first method is to have receivers broadcast joins to the entire data center. For most data center settings, this is a viable option since the rate of joins in the system is typically quite low. This approach is drawn on figure 2. The second method is meant for handling higher churn; it involves explicitly tracking the set of senders for each group via the gossip layer. Since each node in the system knows the set of senders for every group, a receiver joining a group can directly send the join using multiple unicasts to the senders of that group. The second option incurs more space and communication overhead in the gossip layer but is more scalable in terms of churn and system size.

Switching between these two options can be done by a human administrator or automatically by a designated node, such as the mapping module, simply by observing the rate of membership updates in the system via the gossip layer. Once again, the broadcasts or direct unicasts do not have to be reliable, since the gossip layer will eventually propagate joins throughout the system.

3.3 Kernel Multi-send System Call

Sending a single packet to a physical IPMC group is cheap since the one-to-many multiplexing is done on a lower level by routing or switching hardware in the network. However, when IPMC resources are exhausted, the group-address mapping in MCMD will map a logical IPMC group to a set of unicast addresses corresponding to its members. Thus a single `sendto()`-call at the interface would produce a series of sends at the library and kernel level of identical packets to a number of physical addresses. We modified the kernel to help alleviate the overhead caused by context-switching during the list of sends. We implemented a *multi-send* system call on the Linux 2.6.24 kernel with a `sendto()`-like interface that sends a message to multiple destinations.

4 Optimizing Resource Use

Beyond making IPMC controllable and hence safe, the MCMD incorporates a further innovation. We noted that our goal is to optimize the limited use of IPMC addresses. Such optimization problems are often hard, and indeed the optimization problem that arises here we have proven to be NP-complete (details omitted for brevity). Particularly difficult is the problem of mapping multiple application-level groups to the same IPMC address: doing so shares the address across a potentially large set of groups, which is a good thing, but finding the optimal pattern for sharing the addresses is hard.

A *topic* is a logical multicast group. Our algorithm can be summarized as follows.

- Find and merge all identically overlapping topics into *groups*, aggregating the traffic reports.
- Sort groups in descending order by the product of the reported traffic rate and topic size.
- For each group G, assign an IPMC address to topic G, unless the global or user address quota for ≥ 3 members have been exceeded.

- Enlist all remaining users in G for point-to-point communication over unicast.

Thus a large topic with high traffic is more likely to be allocated a dedicated IPMC address. Other groups might communicate over both IPMC and point-to-point unicast for members that have exceeded their NIC IPMC capacity, and yet others might perform multicast over point-to-point unicast entirely.

5 Related Work

Brevity prevents a detailed comparison of our work with previous work of [14, 15]; key differences stem from our narrow focus on data center settings. Our mathematical framework extends that of [12], but instead of inexact channelization we investigate zero filtering.

6 Conclusion

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

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

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

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

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

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

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

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

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

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

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

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

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

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