In Search of Simplicity: A Self-Organizing Group Communication Overlay

Matei Ripeanu, Adriana Iamnitchi, Ian Foster, Anne Rogers

Abstract

Group communication primitives have broad utility as building blocks for distributed applications. The challenge is to create and maintain the distributed structures that support these primitives while accounting for volatile end-nodes and variable network characteristics. Most solutions proposed to date rely on complex algorithms or global information, thus limiting the scale of deployments and acceptance outside the academic realm.

This article introduces a low-complexity, self-organizing solution for maintaining multicast trees, that we refer to as UMM (Unstructured Multi-source Multicast). UMM uses traditional distributed systems techniques: layering, soft-state, and passive data collection to adapt to the dynamics of the physical network and maintain data dissemination trees. The result is a simple, adaptive system with lower overheads than more complex alternatives. We have implemented UMM and evaluated it on up to 1024-node emulated ModelNet networks and on the PlanetLab testbed.. Extensive experimental evaluations and quantitative comparisons with alternative solutions demonstrate UMM's low overhead, efficient network usage, and ability to quickly adapt to network changes and to recover from failures.

1. Introduction

Collaborative applications such as conferencing, shared virtual workspaces, or multi-player games rely on multi-source multicast functionality. Recently, this functionality has been used in contexts such as peer-to-peer resource discovery [2] or resource monitoring [3]. Limited native multicast support in the Internet [4] and application-specific requirements (e.g., reliability, message stream semantics) make application overlays an attractive alternative to IP-layer multicast [5-10]. Application overlays [5] employ participating end-systems to implement functionality not provided by the underlying communication layer or to improve the characteristics of services provided by lower network layers. When providing multicast functionality, overlays serve as support for extracting dissemination trees used to send data from each source to all destinations.

Two broad classes of overlays have emerged, differentiated by the existence of global rules (or the lack of thereof) that restrict their geometry. *Unstructured overlays* allow unrestricted interaction patterns between participating nodes, while *structured overlays* impose global, regular structures. Intuitively, unstructured overlays are less expensive to create and maintain and are more flexible to map on inherently heterogeneous sets of end-nodes linked by diverse physical network topologies [11], [12]. However, existing solutions for multi-source multicast based on unstructured overlays have limited scalability [13, 14] or incur large overheads [2]. On the other side, the scalability advantage offered by structured overlays [15, 16] is offset by less efficient usage of underlying network resources and high protocol complexity [17].

The structured vs. unstructured overlay debate is particularly relevant in a data-distribution context because of the tension between scalability and efficient network usage. This debate is also fueled by the challenges in evaluating distributed systems [18] and de-facto academic performance criteria that favor novelty at (sometimes) the cost of systematic evaluation and quantitative comparison with alternative solutions.

To inform the structured vs. unstructured overlay debate, we address the problem of multisource multicast and propose a simple self-organizing solution based on unstructured overlays and we extensively evaluate it against the best alternatives proposed so far.

As such, we do not claim the novelty of the solution as our main contribution. Indeed, although novel as a whole solution, the multi-source multicast infrastructure we propose builds on traditional, well understood building blocks. We have reached this design in our quest for *simplicity*. We use recently proposed heuristics [19, 20] to build and optimize an unstructured base overlay. On top of it, we use flood-and-prune techniques to select efficient, source-specific data distribution trees: we use the implicit information contained in the duplicate messages filter out redundant overlay paths. The result is a self-adaptive, self-organizing system that performs better than alternative solutions along different performance metrics.

In the following, we refer to the resulting Unstructured Multi-source Multicast solution as UMM. This solution:

- has *low complexity*: it relies on soft-state and passive data collection to extract multicast dissemination trees and adapt to physical network dynamics.
- is *self-organizing*: independent decisions made at each node based on partial, local information result in

desired global system behavior.

- is *scalable* with the number of sources and independent of the number of participants.
- *recovers* quickly from significant failures and *adapts* to changes in underlying network topology with minimal disruption to the service offered.
- is *independent of the topology of the base overlay* and of the mechanisms used to build it which makes it reusable in a wide array of situations.

This work makes multiple contributions. First, we combine a number of traditional, distributed systems techniques to build a simple, efficient data dissemination solution.

Second, and more importantly, we show, via extensive experimental evaluations over a live wide-area testbed (PlanetLab [21]) and large emulated networks (ModelNet [22]), that although UMM uses a low-complexity protocol and nodes maintain little state, it is more efficient than alternative solutions. Additionally, we confirm previous results regarding the relative performance of alternative overlay solutions obtained only through low-fidelity simulations by evaluating actual implementations over large-scale emulated networks.

Third, we identify optimal overlay layouts and use them to quantify the overheads introduced by UMM and alternative solutions. This offers an upper bound for potential gains achievable through better overlay mapping algorithms.

Fourth, we present an empirical method for choosing the parameters that drive the tradeoff between agility to adapt and stability, a tradeoff faced by all adaptive systems.

The rest of this article is organized as follows. Section 2 surveys the main challenges and performance metrics for application-layer overlays. Section 3 presents the main alternative overlay topologies used to provide multi-source multicast functionality. Section 4 presents the UMM design. Implementation details are presented in Section 5 and experimental evaluations in Section 6. Section 7 concludes.

2. Overview

This section presents challenges and success metrics associated with building a self-organizing multicast overlay and presents UMM design choices and solution.

2.1. Challenges and Success Metrics

The key challenges and the corresponding performance metrics are:

i.) Efficient usage of underlying network resources. Since application-level multicast overlays are implemented on top of IP, their efficiency should be compared with that of IP-multicast. An efficient dissemination tree minimizes: (1) *relative delay penalty* (RDP), defined as the ratio between the message propagation delay in the overlay and in the underlying network; and (2) *network stress*, which is the number of duplicate messages generated due to mapping multiple overlay tunnels over the same physical link. These metrics are defined with respect to a common IP-layer multicast base and have been traditionally used to evaluate multicast solutions.

ii.) Scalability is obtained by minimizing the overheads to maintain efficient distributed structures. Two main types of overheads can be identified: overheads for estimating the conditions of the underlying network (e.g., latency, bandwidth) and state maintenance overheads (e.g., routing table and connectivity maintenance, partition detection).

iii.) Resilience. When not bandwidth-constrained, a multicast infrastructure should deliver each message to all destinations in spite of node failures or adaptation events. To quantify resilience, we estimate *delivery rates* under various failure scenarios.

iv.) Adaptation. The infrastructure should build and maintain efficient delivery trees in spite of end-host failures and changes in the topology and state of the underlying network. Moreover, the overlay should adapt quickly to changes in the properties of participating resources and end-host and routing failures.

2.2. Solution Overview

We designed UMM as a *best effort service,* with minimal dependence on centralized, stable components, minimal state maintained at each participating host, and targeting *medium scale* groups with thousands rather than millions of sources.

UMM builds two layers on top of the physical network: the *base overlay* and the *source-specific multicast distribution trees.* UMM starts with a random base overlay and uses a 'short-long' heuristic to incrementally improve it (Section 4.2.). Additionally, the base overlay provides the basic bootstrap, fault-recovery, and connectivity maintenance (Sections 4.3 and 4.4) Consistent with our goal of layer independence, the heuristics and mechanisms used at the base overlay are independent of the mechanisms employed by higher layers.

UMM builds source-specific dissemination trees from the set of tunnels offered by the base overlay as follows (Section 4.1): a new multicast source starts by flooding the base overlay. Participating nodes observe this traffic and use the implicit information contained in duplicated messages to filter-out the redundant, low-quality tunnels that generate duplicate traffic.

3. Overlay Topology Choices

In order to place UMM in context, this section presents a minimal overview of overlay topologies used to support multi-source multicast.

Shared-tree overlays: Shared-tree overlays maintain a tree-structured topology [6, 10, 23] and generally target single-source multicast. The tree topology makes explicit routing mechanisms unnecessary: a node simply forwards each new message on all its tunnels except the tunnel from which it received the message. Shared-tree structures have major limitations: trees are fragile to node failures, introduce additional delay penalties, and do not use efficiently the bandwidth available in the underlying network [24].

Unstructured overlays: Narada [9] and Scattercast [13] build an unstructured, initially random, base overlay mesh and employ a distance-vector routing protocol (DVRMP [25]) to extract source-specific distribution trees. The routing protocol provides each node with information on *all* other system participants and on the optimal cost and path to reach each of them. The considerable volume of data transferred to build and maintain this large state at each node significantly limits scalability: Narada, for example, is designed for groups of up to a few hundred nodes [9].

However, two main advantages result from this approach: first, routes are optimal for a given cost function and base overlay; and second, once detected, connectivity failures can be fixed easily using the routing information. Additionally, the information collected by the routing protocol is used to incrementally improve the base overlay in a solution that couples two functionally independent layers: dissemination tree extraction and base overlay optimization.

Some peer-to-peer (P2P) applications (e.g., Gnutella) use flooding over a random base overlay to distribute messages. While simplicity is a clear advantage of this approach, its main drawback is inefficient network usage [2] due to duplicate messages resulting from ignoring the physical network topology.

Structured overlays: The regular overlay structures at the base of distributed hash tables (e.g., hypercube, Plaxton mesh) can be exploited to extract multicast distribution trees. Originally targeted for different purposes, there is now interest for using structured overlays to support application-layer multicast and wide-area data dissemination [15, 16, 26, 27].

Two important benefits have been asserted for this approach. First, good scalability through reduced state at each node: routing tables grow only logarithmically with the size of the network. Second, the ability to support multiple multicast groups and reuse the overlay structure. These benefits however, are subject to debate. In a recent paper, Bharambe et al. [17] argue that, under realistic deployment conditions involving heterogeneous end-host and link capacities, these benefits are drastically limited: optimized dissemination trees that achieve good performance must include a large number of tunnels that are not part of the original overlay, thus increasing the state maintained at each node. Additionally, these tunnels limit the benefits of route convergence and loop-free properties of the original overlay and generate additional maintenance costs, thus limiting scalability.

4. UMM Design

This section presents UMM's two functional layers and the techniques used to deal with node failure, adapt to network dynamics.. Extended descriptions of the techniques we use and the protocols used for bootstrapping and membership management can be found in our accompanying technical report [28].

4.1. The Base Overlay

UMM uses relatively simple heuristics to build and maintain the base overlay: the aim is simply to include efficient distribution trees that can be later selected by the upper layer. This section presents a high level view of the base overlay management operations while Appendix 10.1 presents the pseudocode.

While the base overlay is initially random, nodes optimize it incrementally, an approach common to other systems [9, 19, 29]. In keeping with our decentralization and self-organization design principles, nodes decide independently based only on local information what overlay tunnels to add or delete. As in Shen et al. [19], nodes maintain a fixed proportion P_{short} (defaulting at 50%) of their tunnels *short*, i.e., to nearby nodes, and the other tunnels *long*, to distant nodes. We use a latency threshold D_{short} (with a 10ms default value) to discriminate between short and long tunnels. At fixed time intervals, a node runs an optimizer task that randomly selects a small subset of participating nodes, evaluates new tunnels, and replaces its worst tunnel with a new one, if better. We limit changes to one tunnel replacement per optimizer iteration.

Short tunnels are optimized for latency: that is, a node replaces its worst short tunnel if a new shorter tunnel is found. Long tunnels are optimized for bandwidth: that is, a node replaces its worst long tunnel if it finds a new tunnel with better bandwidth. To avoid oscillations due to measurement errors or variations in network conditions, a threshold is used when deciding whether to replace an existing tunnel.

In addition, the overlay topology can adapt to heterogeneous node characteristics by defining the number of tunnels a node supports, and thus its load, to be proportional to node capacity (e.g., its access link bandwidth).

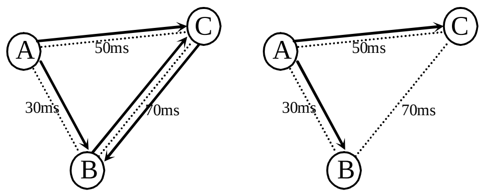

Figure 1: Extracting a dissemination tree rooted at A. *Left*: Flooded messages from source A before B and C detect that tunnel B-C is unnecessary. *Right*: B and C filter out the link BC for messages sourced at A.

4.2. Extracting Efficient Dissemination Trees

UMM extracts efficient, source-specific dissemination trees using the implicit information from duplicate messages resulting from flooding. The key idea is that an initial flooding stage allows each node to infer the properties of the base overlay just by detecting duplicate messages: duplicates arrive on paths with worse properties than those taken by the first message. Once duplicates are detected, this information can be used to filter out redundant paths for efficient message dissemination. The rest of this section presents an informative example while the Appendix 10.2 presents the data structures and the pseudocode for the protocol executed at each node.

In Figure 1 (left), nodes A, B, and C form an overlay. A is the source of a multicast message m_A that is flooded to B and C. B and C in turn flood the message on all their other tunnels. Nodes that receive a duplicate do not forward it further. Assume B receives m_A first from A and then from C. B can thus infer that it has a better path from A than the one that goes through C and can ask C not to forward further A-sourced messages on the C-B tunnel. The same mechanism, when used at C, causes B to be asked to stop forwarding A-sourced messages to C on the B-C link. Thus, nodes B (respectively C) install 'filters' for A-sourced messages for their tunnels to C (respectively B), and a distribution tree as in Figure 1 (right) is extracted using only the duplicate message information that flows into the network. Note that the distribution tree obtained with this mechanism is cost efficient with respect to the costs of sending A-sourced messages.

4.3. Adapting to Node and Tunnel Failure

Node and tunnel failures are the main potential problem once dissemination trees have been built by filtering out redundant tunnels as presented in the previous section.

Failure detection is straightforward: nodes paired by tunnels periodically exchange aliveness messages. If a node does not receive these messages for a predefined period of time, it infers that a failure occurred involving either the other end of the tunnel or the underlying network path.

Once a failure is detected, the node initiates the recovery process. First, to restore the connectivity of distribution trees that used the tunnel, the node that detects the tunnel failure resets all filters for tunnels ending at the node. Second, nodes must also deal with tunnel failures that occur further away in the overlay topology, to prevent local failures from destroying connectivity to remote nodes. To address this latter requirement, nodes that detect a failure flood a *resetRouteMessage* message with a small time-to-live that causes all receivers to reset their 'filters' and to eventually restart the process of tunnel selection described in the previous section. The corresponding pseudocode is presented in the Appendix 10.2.

Furthermore, filters are soft-state: they expire after a certain timeout. Thus, nodes regularly revisit their decision to include or exclude a tunnel from a distribution tree. Timeouts can be fixed parameters set based on application-specific knowledge, or can be computed adaptively, in a manner inspired by TCP timeouts. A related approach is employed in other P2P systems [2, 30] based on the intuition that the 'age' of a node is a good predictor for its future lifetime.

4.4. Keeping the Base Overlay Connected

UMM splits this task into two separate problems: (1) partition detection and (2) repair.

Partition detection. UMM uses heartbeats to detect partitions [13, 19]. A special heartbeat node is used to insert messages in the network at regular intervals T_{htbeat}. Nodes that do not receive enough heartbeats for some multiple of T_{hbeat} conclude that they have been disconnected and initiate the repair procedure. We use N heartbeat sources: a node is connected as long as it hears from more than $^N/_2$ sources. This solution tolerates $^N/_2$-1 simultaneous heartbeat source failures.

Repair. Once a node detects it is disconnected it first flushes the set of other nodes it knows about, and tries to rejoin the network, starting from the heartbeat nodes it did not hear from or from a bootstrap node. To avoid flooding the bootstrap node when multiple nodes simultaneously detect that they have been disconnected, a node waits for a random time before initiating the repair procedure.

An important optimization is preventing partitions induced by the continuous adaptation of the base overlay. To this end, nodes avoid dropping the tunnels over which they receive heartbeats. Similarly, when a node drops a tunnel to add a new neighbor, it makes sure the new neighbor does not receive heartbeats through the tunnel that it planned to drop.

5. Implementation

We implemented UMM in Java. The implementation includes 25 main application classes and 60 auxiliary classes for a total of 5055 lines of code.

To use UMM, an application uses a *send_message* non-blocking call that puts messages in the appropriate outgoing message queues at the local node. The application also needs to register a callback method to get notified when messages addressed to the application arrive. The application can also register callbacks triggered by base overlay or the dissemination tree change events.

The transport protocols used to implement UMM tunnels need to provide two properties: TCP friendliness and ability to express reliability characteristics in terms of application level frames. For convenience, in the experiments presented in this section we use TCP as a base for UMM transport protocol. We are now currently experimenting with a new, UDP-based transport protocol, that provides framing, implements TFRC [31] for congestion control, and allows application to control the tradeoff between transport reliability and timely frame propagation.

6. Experimental Evaluation

This section presents experiments that evaluate UMM techniques described above and the effectiveness of our UMM implementation: We present extensive controlled experiments in an emulated network environment (ModelNet [22]) that allow us to compare UMM with IP-layer multicast and with alternative solutions using relative delay penalty and network stress metrics. We performed additional experiments [28] on a live network (PlanetLab [21]) that confirm our findings presented here.

ModelNet is an emulation environment for wide-area networks in which target applications run unmodified on a set of cluster nodes. ModelNet extracts path delay and bottleneck bandwidth from user-provided network topologies and emulates network traffic conditions.

We experiment with UMM in this controlled environment with *three goals* in mind. First, we compare the quality of the dissemination trees produced by UMM with that of trees built by alternative approaches Second, we estimate the cost of self-organization: we benefit from having full information on the network topology and we compare UMM-generated dissemination trees with optimal trees generated by heuristics using global topological knowledge. Finally, we test UMM ability to operate and adapt when the set of participating nodes is dynamic.

6.1. Experimental Setup

We use ModelNet' suite of tools to generate physical network topologies: ModelNet in turn uses Brite [32] topology generator to generate the topology graph and assign link latencies. Modelnet classifies network links as Client-Stub, Stub-Stub, Transit-Stub, and Transit-Transit depending on their location in the network [24]. Bandwidth for each link is assigned depending on its type: Client-Stub links have bandwidth randomly assigned form the 2 to 8Mbps range, Stub-Stub links: 4 to 10Mbps, Stub-Transit links: 5 to 10Mbps, and Transit-Transit in the 10 to 20Mbps range.

For the experiments presented in this section, nodes are configured to initiate at most five tunnels and accept at most seven. Half of these tunnels are configured as *short*, thus optimized for delay, and half *long* and optimized for bandwidth. The timeout for a tunnel filter is fixed at 600s. The tunnel optimization thresholds mentioned in Section 4.2 are set at their default values: 2 ms for delay and 50% bandwidth improvement.

End-hosts are attached randomly to stub level routers. Overlay nodes then are picked randomly from these hosts. In the experiments presented in this section, for each overlay size we ran ten simulations on ten different physical network topologies. Each topology has 4,040 backbone routers. Simulated overlay size varies from 64 to 1,024 nodes.

6.2. Relative Delay Penalty and Network Stress

To compare the quality (in terms of relative delay penalty and network stress defined in Section 2.1) of the dissemination trees produced by UMM with that of trees built by alternative approaches, we use two sets of comparisons: First, we perform a head-to-head comparison of UMM with Macedon-generated [33] implementations of alternative approaches running on the same emulation platform. Second, we compare UMM performance with that of alternative approaches as reported in a previous simulation-based study by Jain et. al [1]. In this latter comparison, we use ModelNet and attempt to reproduce the physical topologies used in [1] in order to offer a meaningful comparison base.

Macedon [33] is a code-generation tool for overlays: it generates overlay implementations from high-level functional specification of overlay behavior. In the version we used (v1.2.1) Macedon offers specifications for multicast overlays based on shared trees (a random tree and a solution based on Overcast [6] protocol) and on structured overlays (two variants of the Scribe protocol [15]). We instrumented the generated code to extract the multicast dissemination trees built at runtime and compare them with UMM trees.

Figures 2 and 3 summarize the results of this comparison for overlays varying from 64 to 1024 nodes. These results show that UMM performs better than the alternative solutions mentioned above. In all experiments, the *90%-tile RDP offered by UMM is lower* (thus better) than that offered by alternative solutions (Figure 3). On average, the 90%-tile RDP is 29% better than structured overlay solutions based on Scribe, 58% better than for random trees, and more than five times better than for Overcast. We note that the two versions of Scribe available (implementing optimizations proposed by groups at Rice University and Microsoft, respectively) performed similarly. We were unable to run Scribe with more than 512 nodes due to a file descriptor leak in the Macedon implementation.

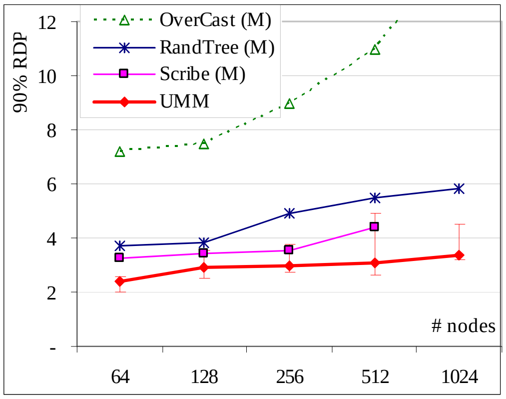

Figure 2: 90%-tile RDP for UMM and alternative solutions for 64 to 1024 node overlays. Each point presents, average values from at least 10 runs over different physical topologies. M stands for Macedon implementation.

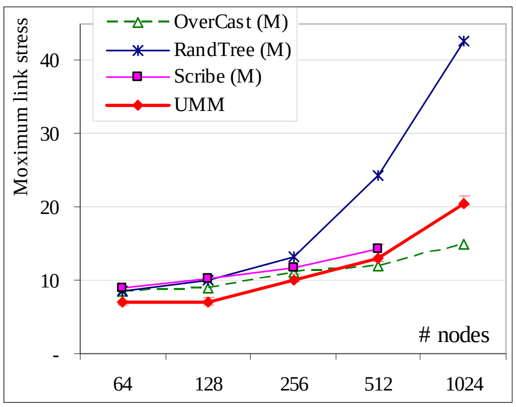

Figure 3: Maximum link stress for UMM and alternative solutions for 64 to 1024 node overlays. Each point presents average values from at least 10 runs (20 for UMM), while the error bars for UMM present min. and max. values.

In part due to our configuration that uses similar fan-outs for the dissemination trees, all solutions we test offer comparable *maximum link stress* (Figure 3). The exception is the random trees which present a much more accentuated growth of maximum link stress for large overlays. Note that, although Overcast appears to produce a slower increase in link stress for large overlays, this result is achieved using a distributed lock to sequence network measurements, a technique that is clearly non-scalable. (In fact, the time to bootstrap a 1,024-node Overcast overlay grows as large as six hours.) Without using the distributed lock mechanism, the link stress evolution is in line with that produced by UMM and Scribe.

The second set of experiments compares UMM performance with that of alternative approaches reported by Jain et al. [1] in a previous study. Jain considered three unstructured overlays (Narada [9], Nice [7], and a power-law overlay that is mapped randomly on the physical topology) and structured overlays employing various mapping heuristics. Figures 4 and 5 summarize the results of this comparison for overlays varying from 64 to 1024 nodes. These results show that UMM generally performs at least as well as other alternatives: The 90%-tile RDP offered by UMM is better than that obtained using *realistic* structured overlays (i.e., structured overlays that do not assume a global view when mapping on the physical network). Compared to solutions that assume global view for optimization (such as Narada and the *idealized* structured overlays), the *90%-tile RDP* offered by UMM is slightly higher, thus worse (Figure 4). In all experiments UMM offers lower (thus better) *maximum link stress* than Narada, structured overlays, and naïve flooding on power-law random graphs (Figure 5).

For clarity, Figures 4 and 5 do not present Nice [7] performance results reported by Jain et al. Jain reports

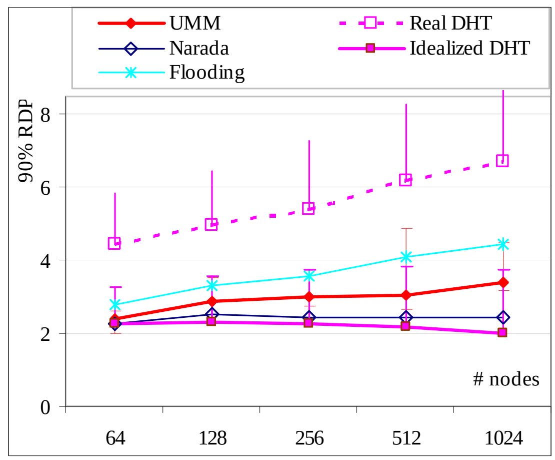

Figure 4: 90%-tile RDP for UMM and alternative solutions for overlay size varying from 64 to 1024 nodes. For UMM, Narada, and flooding, each point presents, average values from at least 9 runs (20 for UMM), while error bars present minimum and maximum values. Structured overlay results (labeled as DHT in the plot) present the average RDP for the best of three heuristics considered by Jain while the performance of alternative heuristics is presented in the error bars. Non-UMM results are extracted from Jain et al. [1].

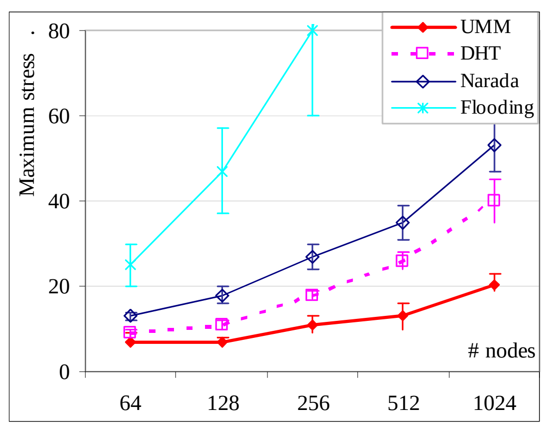

Figure 5: Maximum link stress for UMM and alternative solutions. Each point presents average values from at least 9 runs (20 for UMM), while the error bars present minimum and maximum values. Non-UMM results are extracted from Jain et al. [1].

that, compared to Narada, Nice incurs 22%-60% higher overheads in terms of RDP and performs similarly in terms of network stress. Thus Nice performs similarly to UMM in terms of RDP and worse in terms of stress.

We note that the Scribe performance observed in our experiments is slightly better than that predicted by the decentralized heuristics investigated by Jain (but still much worse than that of idealized heuristics using global knowledge). We attribute this situation to two reasons: first, Scribe enhances the structured overlay with 'shortcuts' to improve the characteristics of the dissemination trees extracted, a heuristic not explored by Jain et al. [1]. Second, the heuristics to improve overlay mapping and message routing in structured overlays have received significant attention since Jain et al. [1] was published.

We note that Castro et al. [34] conclude that dissemination trees based on Pastry (used by Scribe) offer better performance in terms of relative delay penalty and network stress than solutions based on M-CAN [26]. Since UMM performs better than Scribe, Castro's study gives us confidence that UMM performs better than a larger class of structured overlay solutions.

Summary**:** The experiments we have presented support two conclusions: First, we show that it is possible for passive data collection and local decisions to extract *efficient* multicast data dissemination from unstructured overlays. Second, we demonstrate that although UMM uses significantly simpler heuristics than those used by alternative solutions, the data dissemination trees extracted have better or comparable performance.

6.3. Cost of Self-Organization

This section attempts to answer two intertwined questions:

First, we attempt to estimate the cost of self-organization: How do dissemination trees built by the self-organizing UMM solution compare with optimal trees extracted using centralized heuristics that use full knowledge about the physical topology? The performance of these optimal trees serves as upper bound for the achievable performance.

Second, we attempt to understand where to direct future effort to improve UMM heuristics: Which of the two UMM layers, the base overlay or the tree extraction layer, introduce the largest efficiency losses?

In order to answer these questions, we compute, using full topological information, hypothetical dissemination trees optimized to minimize relative delay penalty as described below. We note that computing these trees are variations of the Steiner tree problem [35], and, although we have not proven it formally, we believe that finding optimal solutions is NP-complete. In the following, by 'optimal' solution we mean the best solution discovered by the polynomial-time approximation heuristic we describe.

- *The optimal spanning tree that can be extracted from the UMM base overlay* (labeled **BASE** in Figure 6) We start from base overlay topologies captured during UMM runs and use Dijkstra's shortest path algorithm to compute optimal dissemination trees with respect to propagation delays.
- *The optimal spanning tree that can be extracted from a base overlay whose construction has also been optimized using global knowledge heuristics* (labeled **BESTBASE** in Figure 6). The base overlay is built as follows: initially all nodes add maxdegree tunnels to their nearest neighbors in terms of delay; then, if the graph is not connected, the heuristic iteratively picks one node at random, deletes its worst tunnel and adds one to the next closest neighbor in terms of delay until the graph becomes connected. On top of this base overlay dissemination trees are built using Dijkstra
- *The optimal maxdegree out-degree limited spanning trees built on top of the complete graph* (labeled **OPT**). This heuristic starts from a designated root node and builds a shared tree in a greedy fashion by adding, at each step, the shortest possible tunnel that expands the tree. The only difference from the minimum spanning tree algorithm is that node degree is limited to *maxdegree.* Note that this heuristic will always produce trees with better characteristics than those extracted from a *maxdegree*-limited base overlay. This reflects the difference between single source and multi-source multicast overlays under the same maximum node degree constraint.

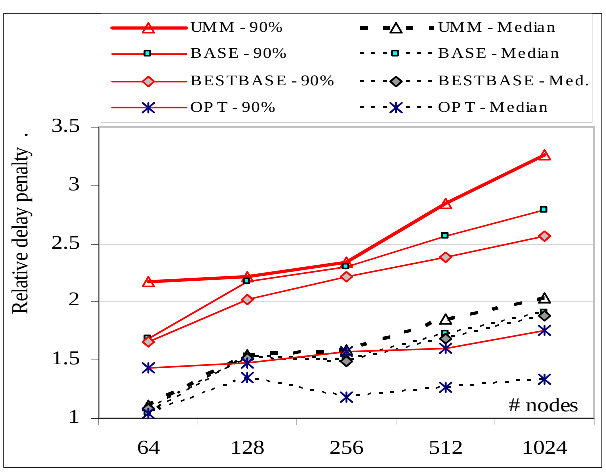

Figure 6: 90%-tile (solid lines) and median (dashed) RDP comparison with distribution trees extracted by heuristics using global knowledge

Figure 6 presents the average 90%-tile and median RDP for disseminations trees built using UMM on ModelNet and using the three heuristics described above for 20 multicast sources for each of the five overlay sizes.

These experiments present the distribution of performance loss generated by the two UMM layers. The yardstick is the performance of centralized, global-knowledge heuristics. Averaged over the five overlay sizes we experiment with, the 90%-itle RDP of UMM is 15.1% less than that of the optimal trees extracted from an optimal base overlay. This performance degradation has two components: 5.3% is due to the non-ideal base overlay and 9.8% is due to the non-ideal dissemination tree. In terms of median RDP however, the total performance loss is only 5.4%, almost entirely localized at the dissemination tree extraction layer. A second observation is that, as expected, dissemination trees optimized for a single source (OPT) offer significantly better performance in terms of RDP.

***Summary*:** These experiments support two conclusions: First, UMM has low overheads when compared to ideal solutions that use heuristics based on global views of overlay membership and network properties, which supports our claim that flooding and passive data collection can extract *efficient* multicast data dissemination from unstructured overlays. Second, two thirds of these overheads are concentrated at the dissemination tree extraction layer.

6.4. UMM Under Churn

The ability to operate in spite of component failures is an important characteristic of any distributed system designed to be deployed on a real-world platform. Here, we evaluate UMM's ability to deliver messages under 'churn', i.e., frequent node arrival and departure.

We measure performance as the ratio of messages delivered to the number of messages that should have been delivered under churn. In order to compute the number of messages that should have been delivered under churn, we keep track of all node join/crash events as well as all message generation events. For each receiver R and for each message m generated by source S at time $T_{S(m)}$ we estimate that the message should have been delivered at R if the node joined the network before the generation time $T_{S(m)}$ and left the network after $T_{S(m)}+T_{prop}$ where T_{prop} is an estimated upper bound for overlay propagation time (set conservatively at 5s in the experiments we report here). In other words, node R should receive a message issued at time $T_{S(m)}$ if: $T_{R(join)} < T_{S(m)} < T_{R(leave)} - T_{prop}$.

To model churn, we use independent node failures controlled by a Poisson process. This model is based on studies of user behavior in multicast groups on the MBone [36] and in file-sharing applications [37] and has been extensively employed. In our experiments, we vary the median node lifetime from 300s to 7200s to infinity (no failures). Studies of various P2P networks report median node lifetimes between 10 minutes [38] and one hour [37, 39]. A new node joins the overlay 10s after each failure to keep the overlay size constant.

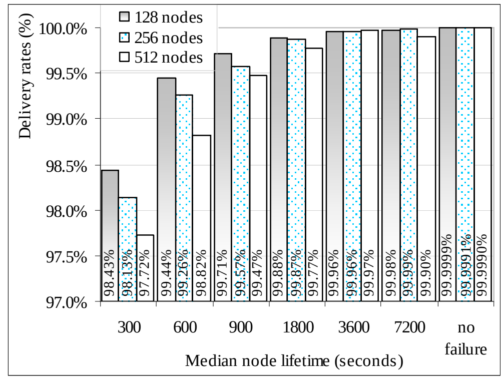

Figure 7: UMM delivery rates under churn. Average delivery rates for 128 to 512 node overlays under various mean lifetime distributions.

Consistent with our aim to gauge UMM behavior in an unfriendly environment, we model crash-like failures as opposed to graceful (announced) node leaves. This inherently leads to higher loss rates: unlike for announced leaves, under a crash model messages are lost when nodes crash with non-empty queues.

Figure 7 presents the results of three sets of experiments with 128, 256 and 512 nodes, respectively, running on ModelNet. Each node inserts small messages into the network at a rate of one message every 5 second (so the system will deliver up to 52k messages/second). We use small messages as we attempt to evaluate the ability to timely detect node crashes, repair the overlay structure, and maintain the overlay connected, while we are less interested here in the ability to evaluate the transport capacity.

As Figure 7 shows, for 512 nodes with 5-minute average lifetime (implying perhaps unrealistically high failure rates of 1.7 failures per second on average in the whole system) 97.72% messages are delivered on average. Under moderate failure rates of 1-hour average node lifetime (translating, to one failure every 7s on average in a 512 nodes system) more than 99.9% messages are delivered on average over the whole experiment. With no node crashes, fewer than one message in 100,000 is lost.

Summary: Although UMM does not include explicit resiliency mechanisms, it is able to recover quickly from node failures and performs well under churn.

6.5. Agility vs. Stability: Bootstrap Convergence

The ability to converge quickly to a stable state with good properties is an important characteristic of systems that operate in a dynamic environment. In this section, we assume the characteristics of the underlying network are stable and we investigate base overlay convergence after bootstrap. First, as a consequence of the greedy base overlay construction strategy where each node, at each time-step, improves its existing tunnels, the base overlay will always converge to a stable state when no external events occur (i.e., no node failures and no variation in underlying network characteristics).

Second, we examine the tradeoff between convergence time and the quality of the stable state to which the base overlay converges. The variables that determine the shape of this tradeoff space are the thresholds that trigger a tunnel change: As we mention in Section 4.2, a node replaces a short tunnel only if the new tunnel offers at least DELAY_THRESHOLD ms. lower delay and replaces a long tunnel only if the new tunnel offers at least BW_THRESHOLD more bandwidth.

Figures 8 and 9 explore the impact of these values on convergence time, number of reconfiguration events, and the quality of the stable state to which the base overlay converges. These figures present averages over 10 runs for a 1024-node overlay for the two extreme sets of threshold values: maximum responsiveness (DELAY_THRESHOLD=0.50ms, BW_THRESHOLD=10%) and stable overlay (8.0ms and 75% for delay and bandwidth thresholds respectively). At each iteration step, a node probes the network path to 10 other overlay nodes, and attempts to replace one short tunnel and one long tunnel. Results are presented in terms of number of iterations, to factor out the duration of a single iteration which is largely determined by the effectiveness of the probing technique.

As expected, higher thresholds for tunnel adjustments lead to faster convergence and a more stable overlay (i.e., fewer reconfiguration events). As Figure 8 shows, for high threshold values, the total rate of reconfiguration events drops to less than one event per time step for the 1024-node overlay after only 10 time-steps, while for low thresholds, the convergence rate is significantly slower: reconfiguration events rate drops to the same level only after 31 time-steps. The total number of reconfiguration events is reduced by more than half when using high thresholds.

The difference in convergence time is also reflected in the characteristics of the stable state obtained using the two configurations above. When using high threshold values, the average tunnel latency is 6.8% worse and the average tunnel bandwidth is 18% worse than when using low thresholds. However, this small degradation in the resulting base overlay quality is acceptable to obtain a more stable base overlay. Moreover, the loss in performance is attenuated when actually extracting dissemination trees from the stable base overlay: 90%-tile RDP is only 4.45% worse.

Summary: We present experiments that correlate stability with ability, i.e., the quality of the base overlay to which UMM converges with the speed of the convergence process. These results can be used to tune node sensitivity to changes in the environment in order to achieve desired levels of service.

7. Related work

In addition to the application layer solutions presented in Section 3, we acknowledge related work in two other contexts: flood-and-prune techniques and high-bandwidth data-distribution.

Flood-and-Prune Techniques have been used at network layers 2 and 3 with various degrees of success and reflecting specific deployment assumptions.

At the *data link layer* (OSI layer 2), bridges employ the spanning tree protocol [40] to extract a shared spanning tree from a non loop-free local network. UMM's solution is different in two respects: first, the trees constructed are source-specific while the algorithm above builds a shared tree. This way UMM offers better use of resources (e.g., transport capacity, delay) available at the lower layer at the cost of maintaining additional state at each node. Efficient use of these resources is of lower concern in LAN deployment scenarios. Second, UMM builds trees by passively listening to actual traffic, without using per-tree control messages. UMM's solution, based on caches of recently received message identifiers, is enabled by larger memory available at end-hosts and application-level frames, which enable a coarser granularity than IP.

At the *network layer*, two commonly used multicast routing protocols, DVMRP [25] and DM-PIM [41], use a flood-and-prune technique on top of already selected multicast dissemination trees. The main difference when compared with UMM lies in deployment assumptions: DVMRP and DM-PIM are deployed across the Internet to extract single-source group-specific dissemination trees. This way, all Internet routers participate in all the tree extraction mechanisms (which implies flooding and holding state for pruning) for all groups, even when routers do not serve any end-hosts interested in these groups. UMM, in contrast, builds per group overlays and state is maintained at only participating end-hosts: thus, as for all application-layer multicast solutions, state is moved to the edges of the network and hosts pay a direct cost only when participating.

High Bandwidth Data Distribution. A number of applications target high-bandwidth data distribution. Both Bullet [24] and SplitStream [42] assume a single source scenario. Bullet establishes multiple trees rooted at the same source and disseminate different chunks of data. By using multiple trees, Bullet reduces the need to perform expensive bandwidth probing for tree optimization. Bullet's solution works best for single source distribution of large-files: it assumes that the application is insensitive to additional transfer delays, that delivery order of various data blocks is not important, and that the additional overhead incurred by coordinating multi-path data delivery is amortized over a large file size. UMM is designed for multiple source applications and assumes perishable data (e.g., media streaming) in context where additional delays and message re-ordering are important concerns.

8. Conclusions

Group communication primitives are building blocks for a large set of distributed applications. However, creating and maintaining the distributed structures that support these primitives is challenging, particularly when network and node characteristics are transient.

UMM, the self-organizing, adaptive multi-source multicast system we propose, is based on a simple approach to extracting source-specific multicast trees from an unstructured overlay. This solution offers two important properties: first, it decouples the overlay construction and maintenance mechanism from the tree-extraction mechanism, allowing for separate component optimization. Second, it relies on soft-state and passive data collection to adapt to the dynamics of the physical network, resulting in low protocol complexity and low overheads.

Experimental and analytical evaluations demonstrate low communication overhead, efficient network usage compared to alternative solutions, and ability to adapt quickly to network changes and recover from node failures. Direct experimental comparisons with alternative solutions implemented by other researchers and deployed on ModelNet along with comparisons with previously published results support our claim that the dissemination trees produced by UMM have better characteristics (better RDP for similar stress) than those built by alternative approaches. Additionally, the performance degradation brought by the simple UMM heuristics based on *local* information is small when compared with heuristics that employ *global* topology information to extract optimized dissemination trees. Finally, we demonstrated the ability of UMM to operate in dynamic environments and adapt when the set of participating nodes is dynamic.

Most importantly, we show that a low-complexity design can lead to a self-organizing, scalable, and adaptive overlay with performance generally better than that offered by more sophisticated solutions and, additionally, with low overheads compared to idealized solutions based on global resource views. Reduced complexity is a highly desirable property of distributed, large-scale systems. When two mechanisms offer similar efficiency for comparable costs, the discriminating factor is complexity. Additionally, in production environments an increasing premium is placed on deployable, manageable, in other words *simple* systems.

9. References

[1] S. Jain, R. Mahajan, and D. Wetherall, "A Study of the Performance Potential of DHT-based Overlays," USENIX Symposium on Internet Technologies and Systems, 2003.

[2] M. Ripeanu, I. Foster, and A. Iamnitchi, "Mapping the Gnutella Network: Properties of Large-Scale Peer-to-Peer Systems and Implications for System Design," *Internet Computing Journal*, vol. 6, 2002.

[3] M. L. Massie, B. N. Chun, and D. E. Culler, "The Ganglia Distributed Monitoring System: Design, Implementation, and Experience," *Parallel Computing*, vol. 30, 2004.

[4] C. Diot, B. N. Levine, B. Lyles, et al., "Deployment Issues for the IP Multicast Service and Architecture," *IEEE Network*, 2000.

[5] J. D. Touch, "Overlay Networks," *Computer Networks*, vol. 36, pp. 115-116, 2001.

[6] J. Jannotti, D. K. Gifford, K. L. Johnson, et al., "Overcast: Reliable Multicasting with an Overlay Network," OSDI, San Diego, California, 2000.

[7] S. Banerjee, S. Lee, B. Bhattacharjee, et al., "Resilient Multicast using Overlays," ACM Sigmetrics 2003, San Diego, CA, 2003.

[8] A. Young, J. Chen, Z. Ma, et al., "Overlay Mesh Construction Using Interleaved Spanning Trees," INFOCOM, 2004.

[9] Y.-h. Chu, S. G. Rao, S. Seshan, et al., "A Case for End System Multicast," *IEEE Journal on Selected Areas in Communication (JSAC)*, vol. 20, 2002.

[10] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, "Scalable Application Layer Multicast," SIGCOMM, Pittsburgh, PA, 2002.

[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology," SIGCOMM 1999, 1999.

[12] S. Jin and A. Bestavros, "Small-world Characteristics of Internet Topologies and Multicast Scaling," MASCOTS, 2003.

[13] Y. Chawathe, "Scattercast: An Adaptable Broadcast Distribution Framework," *ACM Multimedia Systems Journal special issue on Multimedia Distribution*, 2002.

[14] Y.-h. Chu, S. G. Rao, S. Seshan, et al., "A Case for End System Multicast," SIGMETRICS, 2000.

[15] M. Castro, P. Druschel, A.-M. Kermarrec, et al., "SCRIBE: A large-scale and decentralised application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications (JSAC)*, 2002.

[16] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, et al., "Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination," NOSSDAV'01, Port Jefferson, NY, 2002.

[17] A. R. Bharambe, S. G. Rao, V. N. Padmanabhan, et al., "The Impact of Heterogeneous Bandwidth Constraints on DHT-Based Multicast Protocols," IPTPS, 2005.

[18] A. Haeberlenyz, A. Misloveyz, A. Postyz, et al., "Fallacies in evaluating decentralized systems," IPTPS, 2006.

[19] K. Shen, "Structure Management for Scalable Overlay Service Construction," NSDI'04, San Francisco, CA, 2004.

[20] S. Ratnasamy, M. Handley, R. Karp, et al., "Topologically-Aware Overlay Construction and Server Selection," INFOCOM'02, New York, 2002.

[21] A. Bavier, M. Bowman, B. Chun, et al., "Operating System Support for Planetary-Scale Network Services," NSDI'04.

[22] A. Vahdat, K. Yocum, K. Walsh, et al., "Scalability and Accuracy in a Large-Scale Network Emulator," OSDI, 2002.

[23] D. Pendarakis, S. Shi, D. Verma, et al., "ALMI: An Application Level Multicast Infrastructure," USITS'01, 2001.

[24] D. Kostic, A. Rodriguez, J. Albrecht, et al., "Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh," SOSP'03, Lake George, NY, 2003.

[25] S. Deering, "Multicast routing in internetworks and extended LANs.," SIGCOMM, Stanford, 1988.

[26] S. Ratnasamy, M. Handley, R. Karp, et al., "Application-level Multicast using Content-Addressable Networks," 2001.

[27] S. El-Ansary, L. O. Alima, P. Brand, et al., "Efficient Broadcast in Structured P2P Networks," IPTPS'03, 2003.

[28] M. Ripeanu, A. Iamnitchi, I. Foster, et al., "In Search of Simplicity: A Self-Organizing Group Communication Overlay," University of British Columbia TR-2007-05, 2007.

[29] S. Rhea, D. Geels, T. Roscoe, et al., "Handling churn in a DHT," USENIX'04, 2004.

[30] F. E. Bustamante and Y. Qiao, "Friendships that last: Peer lifespan and its role in P2P protocols," Workshop on Web Content Caching and Distribution, Hawthorne, NY, 2003.
[31] M. Handley, S. Floyd, J. Padhye, et al., "RFC 3448: TCP Friendly Rate Control (TFRC): Protocol Specification," 2003.
[32] A. Medina, A. Lakhina, I. Matta, et al., "BRITE: An Approach to Universal Topology Generation.," International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems- MASCOTS '01, 2001.
[33] A. Rodriguez, C. Killian, S. Bhat, et al., "MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks," OSDI, 2004.
[34] M. Castro, M. B. Jones, A.-M. Kermarrec, et al., "An Evaluation of Scalable Application-level Multicast Built Using Peer-to-peer Overlays," INFOCOM 2003, 2003.
[35] A compendium of NP optimization problems, 2005, accessed on:
[36] K. C. Almeroth and M. H. Ammar, "Collecting and modeling the join/leave behavior of multicast group members in the mbone," HPDC, 1996.
[37] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," Multimedia Computing and Networking Conference (MMCN), 2002.
[38] J. Chu, K. Labonte, and B. N. Levine, "Availability and locality measurements of peer-to-peer file systems," ITCom: Scalability and Traffic Control in IP Networks, 2002.
[39] R. Bhagwan, S. Savage, and G. Voelker, "Understanding Availability," 2nd International Workshop on Peer-to-Peer Systems (IPTPS '03), Berkeley, CA, 2003.
[40] A. S. Tanenbaum, *Computer Networks*: Prentice Hall PTR, 1996.
[41] RFC 3973 - Protocol Independent Multicast - Dense Mode (PIM-DM): Protocol Specification, 2005, accessed on:
[42] M. Castro, P. Druschel, A.-M. Kermarrec, et al., "SplitStream: High-Bandwidth Multicast in Cooperative Environments," SOSP'03, Lake George, NY, 2003.

10. Appendix

The appendix presents pseudocode for the two UMM layers: base overlay management and dissemination tree extraction and maintenance.

10.1. Base overlay management.

This section presents the pseudocode for the main thread in charge with the base overlay management

```
while (true) do:
    connCandidateSet = randomCandidates
(MAX_CANDIDATES)
    foreach conn in connCandidateSet do:
        // probe the network path for this possible connection
        // and use exponentially weighted averages add with
older data
        evaluatePotentialConnection (conn)

    // order the connection set in decreasing order of their
potential
    sort(connCandidateSet)

    foreach conn in connCandidateSet  do:
        // try to replace the worst existing short connection
        // uninitialized local tunnels have infinite delay
        if isShort(conn) then:
            if (conn.delay < worstCurrShortTunnel.delay –
                                    DELAY_THRESHOLD)
    then:
                if (createNewConnection (conn)) then:
                    adjustConnectionsNumber()
                    delete (worstCurrShortTunnel)
                    break;

        // try to replace the worst existing long connection
        // uninitialized local tunnels have 0 bandwidth
        if isLong(conn) {
            if (conn.bw < worstCurrLongConn.bw *
                            BANDWIDTH_THRESHOLD) then:
                if (createNewConnection (conn)) then:
                    adjustConnectionsNumber();
                    delete (worstCurrShortTunnel)
                    break;
    wait (OPTIMIZER_THREAD_DELAY)
end while
```

10.2. Dissemination tree extraction and maintenance

At this layer each node uses three soft-state state data stores. These data stores are soft-state in the sense that data items are inserted with associated timeouts. When a timeout expires, the data store simply flushes the associated data item.

- A data store for IDs of received messages and the tunnel they arrived: idStore (id, conn, timeout)
- A data store for tunnels that have been filtered out by remote nodes: tunnelFiltered (source, conn, timeout)
- A data store for tunnels that have been filtered out by local nodes: tunnelFilteredLocal (source, conn, timeout)

The pseudocode blow describe UMM's reaction to various events:

```
event dataMessage(m, conn):
    oldConn = idStore.put(m.ID, conn, TIMEOUT_IDSTORE)
    // if this was a new ID put into the idStore
    if (oldConn == NULL) then:
        // then deliver to application and propagate the msg.
        callCallback (deliverMessageCallback, m)
        if (m.ttl – – <= 0) then:
            return
        foreach c in localNodeConnections do:
            if (not c in tunnelsFiltered) and (c != conn) then:
                send (m, c)
    else { // this was a duplicate so:
    // ask the node at the other of the connection not to send
    // messages from the same source again
    // (if the connection is active)
    if oldConn.active() and (oldConn not in
    tunnelFilteredLocal) then:
        send (new dropRouteMessage (m.scr), conn)
        tunnelFilteredLocal.put (m.scr, conn,
    TIMOUT_TUNNEL)

event dropRouteMessage (m, conn):
    tunnelsFiltered.put (m.scr, conn, TIMOUT_TUNNEL)

event failedConnection (conn):
    // when a tunnel failure is detected then spread a
    resetRouteMessage
    // with a small time to live
    m = new resetRouteMessage (RESET_TTL)
    foreach c in localNodeConnections do:
        send (m, c)
        tunnelFilteredLocal.reset(c)
    tunnelFiltered.reset()

event resetRouteMessage (m, conn):
    // only for new messages
    if (NULL == idStore.put(m.ID, conn, TIMEOUT_IDSTORE))
then:
        tunnelFiltered.reset(conn)
        if (m.ttl – – <= 0) then:
            return
        foreach c in localNodeConnections do:
            send (m, c)
            tunnelFilteredLocal.reset(c)
```