UMM: A dynamically adaptive, unstructured multicast overlay

Matei Ripeanu[*] Ian Foster[*+] Adriana Iamnitchi[±] Anne Rogers[*]
[*]The University of Chicago [+]Argonne National Laboratory [±]Duke University

***Abstract* - The simplicity of multicast as a communication primitive belies its broad utility as a building block for distributed applications. Nevertheless, creating and maintaining multicast structures can be challenging, particularly when networks are transient and/or dynamic. We introduce a new unstructured multi-source multicast (UMM) overlay approach that we argue is less complex than, but as efficient as, current state-of-the-art solutions based either on structured overlays or on running full routing protocols at the overlay level. UMM builds a base overlay independently from the routing mechanisms employed to route messages. On top of this base overlay, it selects distribution trees for each multicast source by first flooding the base overlay and then using the implicit information contained in duplicated messages to select and filter out redundant tunnels. Simple heuristics are used to maintain and evolve both the base overlay and the multicast distribution trees in response to changes in the set of overlay participants or in underlying network conditions. We experiment on a 65-node PlanetLab deployment and on ModelNet emulated distributed platforms to quantify the overheads associated with UMM operation and to explore its performance and adaptability to changes in the underlying network conditions.**

I. INTRODUCTION

Multicast is a broadly useful communications primitive in distributed systems which provides the ability to deliver data to every member of a set of nodes.

Recently, considerable effort has been spent designing multicast solutions based on structured overlays [1-5]. Yet unstructured solutions may map more *naturally* on inherently heterogeneous sets of end-nodes linked by networks that displays scale-free [6] and small-world [7] characteristics, and are less expensive to create and maintain.

This paper argues that is possible to design an unstructured multi-source multicast (UMM) overlay that is less complex than, but as efficient as, current state-of-the-art solutions, based on structured overlays [1-4] or on full routing protocols at the overlay level [8, 9].

In brief, UMM works as follows: it builds a relatively rich base overlay and uses heuristics to adapt its topology to the state of the underlying network. On top of the base overlay, UMM selects efficient distribution trees for each source by first flooding the base overlay and then, using the implicit information contained in duplicate messages received at nodes, by selecting and filtering out unnecessary overlay tunnels.

This paper aims to show that UMM solution is efficient, simple, and adaptive:

- *Efficient.* We use traditional measures of stretch and stress to compare with IP layer multicast and, indirectly, with other proposed solutions. Experience gained with a deployment on over 60 PlanetLab nodes and on an emulated distributed testbed indicates that UMM performs well compared to alternative solutions.
- *Simple.* When two mechanisms offer similar efficiency for comparable costs, the discriminating factor is *complexity*. UMM preserves the simplicity of flooding based solutions and the flexibility of unstructured overlays. It decouples message routing and overlay construction so that the two can be optimized separately.
- *Adaptive.* UMM adapts to the underlying network topology, and recovers from node and network failures with minimal disruption to the multicast service offered.

The rest of this paper is organized as follows. The next section introduces multicast overlays and their challenges, while Section III surveys related work. Section IV presents our multicast solution for unstructured overlays, while Section V qualitatively compares UMM with alternative solutions. Section VI presents our implementation prototype and performance results obtained on a 65 PlanetLab nodes deployment and in a ModelNet emulated environment. We summarize and review ongoing work in Section VII.

II. MULTICAST OVERLAYS AND THEIR CHALLENGES

Multicast allows a sender to send data to multiple receivers. Generally, the sender does not know the identities of all receivers: it sends data to a multicast *channel*, to which receivers subscribe. A multi-source multicast is simply a multicast channel to which multiple senders can send data. In many concrete applications the sets of senders and receivers overlap.

Multi-source multicast has its origins in collaborative applications such as audio- and video-conferencing, shared virtual workspaces, and multi-player games. More recently, multi-source multicast has been used for discovery and replication in peer-to-peer systems. Another important application domain may be the rapid instantiation of virtual organizations [10] and their associated services. These and other contexts, require a multicast solution able to support medium-scale groups (hundreds to possibly tens of thousands of nodes) and applications that benefit from a message stream (as opposed to byte stream) semantic.

A. Why overlays to provide multicast?

After extensive efforts to provide multicast functionality in the networking infrastructure [11], at the IP level, it has become customary to build this functionality using application-level overlays. An overlay network builds logical *tunnels* between pairs of nodes connected by an underlying network. It then uses these tunnels to transport messages between connected nodes and implements its own routing mechanisms to deliver messages through the overlay.

Multiple arguments support this approach. The end-to-end principle [12] argues for implementing new functionality as high as possible in the protocol stack, as long as the associated performance penalty is lower than the complexity and cost associated with implementing it at lower levels. Experience suggests that significant complexity is associated with IP-level multicast. Thus, application-level overlays are an attractive alternative.

Application Level Framing [13] argues for the ability to express reliability requirements in terms of Application Level Units (ALU), e.g., a video frame, as opposed to the raw byte-stream semantic of IP multicast. ALU-based semantics are easier to implement at end-hosts (than at routers) due to larger buffering and storage capabilities, all under application control.

Finally, some group collaborations need the ability to assemble dynamically with minimal prior infrastructure deployment at participating nodes/sites. Given its slow deployment, alternative approaches to IP-layer multicast will facilitate adoption for group collaboration applications.

As an aside, we note that providing multicast functionality using application-level overlays does not prevent deploying proxies at key points in the network, to improve overlay efficiency or reliability. Moreover, once proven useful and once their architecture matures, these deployments can become part of the infrastructure.

B. Challenges and design choices

We summarize the challenges associated with building a multicast overlay and enumerates success metrics employed to estimate how well various solutions techniques address these challenges.

i.) *Efficient usage of underlying network resources.* Since overlays are implemented on top of IP, their efficiency should be compared with that of the best IP-level solution namely IP-multicast. An efficient overlay tries to minimize:

- *relative delay penalty* (RDP: also called *stretch*), the ratio between the message propagation delay in the overlay and in the underlying network; and,
- *stress*, the number of duplicate messages generated due to multiple logical tunnels being mapped over the same physical link.

These metrics are defined with respect to a common IP-level multicast base and have been used to evaluate most previous multicast projects. Thus, we can use these metrics to compare, albeit indirectly, the performance of UMM and other approaches.

ii.) Scalability. Scalability is obtained by keeping low the overheads to discover and maintain an efficient topology. Ideally we would like these overheads to increase at most linearly with the number of nodes. Two main types of overheads can be identified:

- Overheads for estimating the conditions of the underlying network: e.g., for estimating latency and available bandwidth of a certain Internet path.
- Overheads due to node state maintenance (e.g., state storage, or control messages to maintain routing tables, maintain connectivity, detect partitions, and bootstrap) or due to duplicate messages in the overlay itself.

iii.) *Adaptation.* UMM should adapt to changes in the topology and state of the underlying network and maintain efficiency, as defined above, despite node or routing failures and changes in network characteristics.

The overlay should recover quickly from failures of both participating nodes and the underlying network (e.g. routing failures). If adaptation is a process by which a change (e.g., failure) is responded to, we can measure the 'success' of adaptation at any point by tracking the change in terms of metrics like stress, average RDP, or delivered data rates. Presumably we will typically see these metrics first drop and then increase to a new stable point. The time required to cross a certain threshold of effectiveness is a measure of *agility*, as is the time required to stabilize. There are obvious tradeoffs between the measurement overheads to detect failures and the agility of the system.

We summarize the salient design choices/goals of our solution:

- *Correctness.* Ideally, all nodes receive all multicasted messages. However, node crashes, IP-level routing failures, overlay self-organization events, or insufficient transport capacity in the underlying network may cause lost messages. We aim simply to provide a best effort service; reliable delivery and flow control can be implemented at a higher layer if necessary.
- *Decentralization.* For our target applications, solutions that rely heavily on centralized, stable components implicitly raise adoption barriers due to the need to define, deploy, and operate the central component(s). We seek minimal (ideally no) dependence on centralized, stable components.

- *Semantic-free abstractions.* Although not the focus of this study, we also seek "semantic free" abstractions: that is, abstractions that: (1) enable multiple applications to share the same overlay or, if independent overlays are used for multiple applications a node participates to, these applications should share parts of the infrastructure to construct the overlay (e.g., network measurement information [14], or reuse entire components [15]), and (2) enable applications to express and implement their, application-specific, reliability and control-flow requirements.

III. Related Work

We introduce a few terms: a *tunnel* links two overlay nodes that exchange messages directly; we use *link* for physical network links. In fact, one can talk about an overlay system at three layers (Figure 1):

1. *the underlying physical network* (e.g., the Internet);
2. *the base overlay*: the set of all tunnels that link pairs of nodes using the physical network; and
3. *the application overlay*: the subset of the tunnels in the base overlay used by the routing mechanisms to deliver multicast data for each specific source.

For approaches that do not make a distinction between the base and the application-overlay we use overlay to refer to both.

With so many metrics to optimize for, it is not surprising that many approaches have been explored. We organize the related work according to the structure of the overlay used.

A.) Mesh-based, unstructured overlays.

Narada [9, 16, 17] and Scattercast [8] build an unstructured overlay over which they run a distance-vector routing protocol (DVMRP [18]) to extract efficient source-routed distribution trees, one per source. This protocol provides each node with information on all other system participants and on the optimal cost and path to each. This information is also used to evaluate new candidate links for improving the base overlay, and to ensure that the base overlay remains connected.

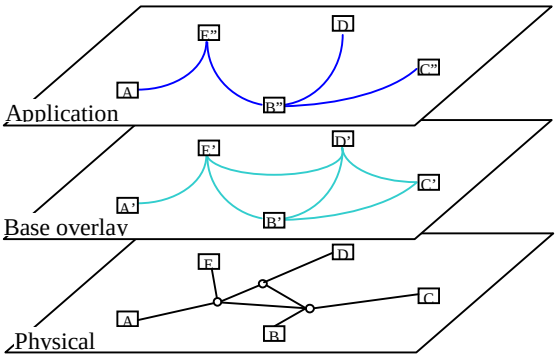

Figure 1: Layered view of five nodes linked in a physical topology, a base overlay, then an application-level overlay used to multicast data from node B.

Two main advantages result from this approach: (1) routes are 'optimal' for a given base overlay and cost function, and (2) connectivity failures can be fixed easily, once detected, using the routing information.

The price paid for these advantages is the overhead of maintaining routing information: nodes periodically exchange routing tables that are proportional in size to the number of nodes in the network. Thus, at each node, space overhead is proportional to the product of the size of the network and the number of links a node maintains.

Some P2P applications (e.g., Gnutella) take a different approach: they do not attempt to optimize the unstructured overlay and do not collect routing information but instead use flooding to distribute messages. The resulting protocol is simple and there is no need to maintain routing tables. The drawback is inefficient network usage due to (1) overhead traffic for each message, the generated traffic is proportional to the number of tunnels in the overlay, not the number of nodes, and (2) the high network stress that results from the random overlay construction.

B.) Tree-based overlays

Overlays that maintain a tree-structured topology [19-22] generally target single-source multicast applications, and generally assumes a reliable node (the multicast source) orchestrating the tree construction. The tree topology makes an explicit routing mechanism unnecessary: each node simply forwards each incoming message down all outgoing tunnels.

However, the same mechanisms cannot be used effectively for multi-source multicast. The main problem is the centralized architecture a root node implies. In the single-source scenario, a failure of the root node would not hurt since it would also imply an inability to originate new messages; this is no longer true for a multi-source scenario. Additionally, the 'root' node might prove a bottleneck, and the repair penalty for a node failure is generally higher in a tree.

C.) Structured overlays (also known as DHTs)

This approach builds overlays with a regular structure—e.g., hypercube [23], Plaxton mesh [24, 25] or ring [26]—and exploits this structure to extract efficient distribution trees. As Jain et al. observe [27], while originally targeted for different purposes, structured overlays are now targeting the same

applications as the unstructured overlays presented above, most visibly application-level multicast [1-5].

Regular structure makes dealing with churn (node volatility) and efficiently mapping the overlay to the underlying network inherently more complex and potentially less efficient than for unstructured approaches. Jain et al. [27] explore the efficiency limits of structured overlay approaches using global knowledge to map a structured overlay topology optimally to the infrastructure topology. The (idealized) performance reported is close to that of unstructured overlays in terms of stretch and RDP. This result should be seen as an optimistic performance upper bound: the performance reported for structured overlays constructed *without* global knowledge is two to three times worse than Narada in terms of RDP and similar in terms of maximum stress. However, there are benefits to the structured approach: better scalability through reduced overheads compared to Narada (routing tables grow only logarithmically with the size of the network) and better reliability compared to tree-based overlays, through decentralization.

A number of projects [28, 29] build and maintain an (unstructured) base overlay independently from the mechanisms employed to build the application overlay, or provide generic infrastructures or components for building overlays [14, 15, 30]. As we describe later we have built on this experience.

Epidemic techniques [31] can be introduced to decrease the rate of undelivered messages, albeit at the cost of increased overhead traffic.

IV. UMM – AN UNSTRUCTURED MULTI-SOURCE MULTICAST OVERLAY

UMM builds a base overlay independently from the mechanisms employed at the application overlay. On top of this base overlay, it selects distribution trees for each multicast source by first flooding the base overlay and then using the implicit information contained in duplicated messages to select and filter out redundant tunnels.

UMM uses a relatively dense base overlay and employs heuristics to optimize at the base layer. Goals at this layer are to include efficient distribution trees that will be extracted later by the application overlay layer and to provide basic adaptation and fault recovery functionalities.

We build the application overlay (i.e., the multicast distribution trees routed at each source) using the observation that duplicate messages resulting from flooding contain implicit information about the quality of the network paths in the base overlay. This information, acquired passively, is used to select the best base overlay tunnels to assemble distribution trees for each multicast source.

The remainder of this section is organized as follows: we start with describing the techniques to extract efficient distribution trees from the base overlay as we believe this key to evaluate the rest of the paper. Subsections D to F describe the heuristics UMM uses to build, adapt, and bootstrap the base overlay.

C. Routing: the application overlay

We extract efficient distribution trees, routed at each source, using the information that flows into the base overlay. An important observation is that a flooding-like routing algorithm can allow a node to infer some properties of the base overlay just by detecting duplicate messages.

For example, in Figure 2 (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 message 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 a distribution tree as in Figure 1 (right) is extracted only from 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 forwarding A-sourced messages. That is, the tree is optimal unless cross-traffic or link failures cause transitory delays that affect the initial message.

Dealing with node or tunnel failures is the main problem that appears once messages are filtered and tunnels are eliminated selectively from the initial flooding as just described. We assume that a node located at either end of a tunnel can detect a tunnel failure (regardless of its cause). Such a node can then initiate the process of reinstating eliminated tunnels ending at the node. Nodes must also deal with failures that happen further away in the overlay topology, so as

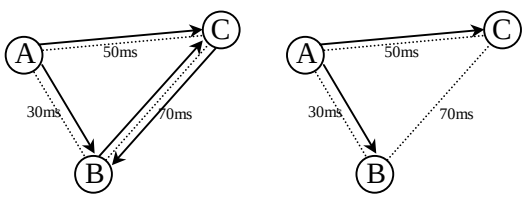

Figure 2: Routing messages from source A before and after B and C detect that tunnel BC is unnecessary. A, B, and C are three overlay nodes.

to prevent local failures from destroying connectivity to remote nodes. To address this latter requirement, nodes that detect failures flood a FAIL message with a small TTL that causes all receivers to reset their 'filters.' Furthermore filters are soft-state: they expire after a certain timeout. Thus nodes regularly revisit their decision whether to exclude a tunnel from a distribution tree. Timeouts can be fixed configuration parameters set based on application-specific knowledge, or can be computed adaptively, in a manner inspired by TCP timeouts. The latter approach is employed by some P2P systems [32, 33] and is based on the intuition that the current 'age' of a node is a good predictor for its future lifetime. Thus, if tunnel L is filtered out with timeout T, and, after a timeout is determined to be again redundant, then the tunnel will be filtered out again with timeout $ratio*T$; otherwise, when the tunnel is filtered out again later its timeout is reinitialized to T_{init}. (Remember that filtering out a link is done per source.)

D. Topology maintenance: the base overlay

We use a few heuristics to obtain a rich base overlay from which efficient distribution trees can be selected. While the overlay is initially random, nodes optimize it continuously. In keeping with our goal of decentralization, UMM uses heuristics that merely avoids 'bad' base overlays, and that use only uncoordinated decisions at each node based on local information to improve the overlay.

Our current implementation adopts an idea from Saxons [28]: nodes maintain a fixed proportion P_{short} (currently 50%) of their tunnels *short*, i.e., to nearby nodes, and the other tunnels *long* to distant nodes. We use a threshold D_{short} (currently 10ms) to discriminate between short and long tunnels.

Every T_{opt} time interval, each node runs an optimizer task that randomly selects a small subset of the nodes that it knows about but to which it does not have direct tunnels, and evaluates the potential new tunnels it could build to each of these nodes. We limit the changes to one tunnel replacement per optimizer iteration.

- Short tunnels are optimized for latency. That is, a node replaces its worst short tunnel if it finds a new tunnel shorter than one of its existing tunnels. To avoid oscillations due to latency measurement errors or to variations in network conditions, a $S_{threshold}$ threshold (1ms in our experiments) is used.
- Long tunnels are optimized for bandwidth. That is, a node replaces its worst long tunnel if it finds a new tunnel that offers better bandwidth than one of its existing tunnels. Again, to avoid oscillations, a $L_{threshold}$ threshold (50% improvement in our experiments) is used.

Additionally we make the number of tunnels each node supports, and thus the node load, proportional to its bandwidth to the Internet. Since the number of tunnels selected in the application overlay (and thus the node load) is generally proportional to the number of base overlay tunnels, this heuristic indirectly allocates load proportional to node characteristics.

E. Keeping the base overlay connected

We split this task into two separate problems: (1) partition detection and (2) repair following detection.

Partition detection. One intuitive approach (also employed elsewhere [8, 28]) is to have a special heartbeat node that inserts messages in the network at regular intervals T_{htbeat}. Nodes that do not receive the heartbeat for some multiple of T_{hbeat} conclude that they have been disconnected and initiate the repair procedure. If the heartbeat monitor dies, Saxons [28] suggests an election protocol to elect a new heartbeat source. As this protocol is unlikely to work at our target scale, we use N heartbeat sources: a node is connected as long 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 was disconnected it first flushes the set of other nodes it knows about, and then tries to rejoin the network, starting from the heartbeat nodes it did not hear from. Nodes wait for a random time before initiating the repair procedure, in order not to overload the bootstrap node (see Section F) when multiple nodes detect they have been disconnected at the same time (e.g., if an overlay partitioning occurs).

An important improvement is preventing partitions induced by the continuous adaptation (optimization) of the network. To this end, nodes avoid dropping the tunnel over which they receive heartbeats. Also, when a node does drop a tunnel to add a new neighbor, it makes sure the new neighbor does not receive its heartbeats through the tunnel that it planned to drop.

F. Bootstrap

We employ a Gnutella-like [33] solution for bootstrap and membership information maintenance at each node. In addition to using a bootstrap node, when two arbitrary nodes contact each other, they also exchange a small random subset of identifiers of other participants in the overlay. This solution, akin to epidemic communication of membership information, has scaled well in large deployments (Gnutella, for example, has more than 100k nodes), and has been studied extensively [8, 28].

V. ANALYSIS

UMM can be thought of as finding a middle ground between Narada's expensive routing tables and thus limited scalability and Gnutella's poor use of network resources due to random overlay construction and naïve flooding. We preserve Gnutella's decoupling between routing and overlay construction, but we use the traffic flowing into the network to build efficient distribution trees dynamically, and we continuously optimize and adapt the base overlay. The rest of this section compares UMM's overheads with those of other solutions and argues that our solution is indeed appealing.

A. Overheads comparison: Size of the node state

Each UMM node maintains two internal structures: (1) its routing tables that describe which tunnels are filtered out for each source, and (2) a cache to store message identifiers (IDs) for recently processed messages.

A node's routing tables hold up to O(*S*links*) state where *S* is the number of multicast sources in the overlay. This space requirement is worse than in structured overlays where nodes maintain O(*log N*) routing information. The overhead is similar to or lower than for Narada or Scattercast which store O(*N*links*) sized routing tables. We note, that for the system scales we are targeting, the space to store the routing tables are relatively small.

The message IDs cache stores 8 bytes for each message, thus giving the node the ability to detect duplicates. The cache is sized to store IDs for all messages generated during a small multiple of the maximum propagation delay in the overlay. We argue that this size is manageable: first, even high data rates (tens of Mbps) produce relatively low message rates as applications send (variable size) application-level units that will often be larger than an IP packet. For example the sending rate for high quality video stream is 30 frames/sec only. Second, the maximum propagation delay in the base overlay is in the order of tens of seconds at most.

B. Overheads comparison: Communication

Communication overheads generally arise when building and maintaining node state and when estimating the performance of the underlying network.

UMM has no direct overhead to maintain state: state is inferred from flowing traffic. However, some overhead traffic results from duplicate messages when a new source starts or when a tunnel filter times out. Our measurements show that overhead traffic generated by duplicate messages is on average less than 1.5%.

In Narada and Scattercast, nodes periodically exchange routing tables. Thus, for each overlay link, the overhead is proportional to routing table size. Thus the generated load per node is *O(N*links)* and the overhead traffic generated by the whole overlay is $O(N^2*links)$. In proposed structured overlay schemes [34-37], the overhead associated with state maintenance is at most proportional to *O(log N)*, the number of entries in the routing table.

In order to adapt to the conditions of the underlying network, a node must select from a random set of potential candidates when adding a new tunnel. Thus, the node needs latency and bandwidth estimates to each node in that set. However, this requirement arises in all solutions considered above, which furthermore we expect to converge at similar rates given the same number of nodes to be probed and the same probe frequency. Thus, while the effectiveness of the probing techniques employed is a concern, it is not the discriminating factor between various overlay-based multicast solutions.

C. Why is UMM appealing?

Efficiency. As the data in the next section also supports, UMM offers topologies with quality (in terms of RDP and stress) comparable to those offered by significantly more complex schemes. Generally, using flooding the shortest path between two nodes is discovered first. Combined with the heuristics to build the base overlay, this results in UMM relative delay penalty at least as good as other solutions offer. Additionally our results show maximum link stress at par with data reported for alternative solutions.

Simplicity. UMM preserves the simplicity of flooding based solutions and the flexibility of unstructured overlays. UMM decouples base overlay construction from the routing mechanisms employed by the application overlays. This separation of concerns allows the two spaces to be explored independently, without the complexities of one space impinging on the other.

Scalability. Although not aiming to scale to millions of nodes like structured overlays, UMM scales better than Narada to medium-size groups. One major advantage is that node state is collected by simply inspecting the message flow; thus, no active state collection protocol (e.g., exchanging routing tables) is used. Additionally, the (soft) state stored at each node is proportional to the activity in the overlay: if only 10% of the overlay participants are multicasting, then routing tables are one tenth of those employed by Narada.

Intrinsic adaptivity. UMM adapts at two levels. The base overlay adapts to node and routing failures and changes in underlying network status. The timeout

mechanism for tunnel filters enables the application overlay to adapt to changes in message stream characteristics. Additionally, each flow from a data source can be tagged differently and UMM can build different distribution trees optimized for the characteristics of each flow.

The distinction UMM makes between 'short' and 'long' links and the heuristic it employs to build the base overlay yield a small-world topology. We justify this assertion as follows. First, the triangle inequality generally holds in the Internet [38]. Thus, two nodes that have short tunnels to a common third node are likely to be close to each other and, given our overlay construction algorithm, to have a tunnel to each other. Second, the long links add a degree of randomness that results in a low diameter for the base overlay. The small world property has two effects:.

- The base overlay has a low average path length, characteristic of random graphs, that grows logarithmically with the overlay size.
- Wats [39], using a simplified lattice model, then Jin [7], using AS and router-level Internet topologies, suggest that one cause for small-world topologies observed in these networks is preference for local connectivity.

We conjecture that this heuristic leads to a base overlay that is likely to match the small-world property of the Internet topology itself [7].

VI. IMPLEMENTATION EVALUATION

We evaluate our implementation in multiple settings. We first investigate isolated node performance to verify that it can sustain required throughputs. We then present two sets of experiments: first, we explore UMM behavior when deployed on live network testbed (PlanetLab). Second, we use an emulated network to perform controlled experiments that allow us to compare UMM performance and that of alternative solutions and to gauge its behavior when the set of participating nodes is dynamic.

In our current implementation overlay tunnels are implemented TCP connections. For the experiments we describe in this section nodes are configured to initiate at most 3 tunnels and accept at most 5. The timeout for a tunnel filter is configured at 600s.

A. Isolated Node performance

We implemented UMM as a Java standalone application. Figure 3 presents the maximum loss-free routing performance of one UMM node for various message sizes. For this experiment we use a 2GHz AMD machine with 256MB of memory connected through a 100Mbps Ethernet card to the local LAN. The UMM node receives data on one incoming overlay tunnel and forwards it on three other outgoing tunnels. For each message size, we increase the sending rate until the queues at the node start building up and the node starts to drop messages. For message sizes larger than 4kB, performance is limited by the 100Mbps network link while for smaller message sizes the bottleneck we believe is the JVM thread switching.

We believe this performance is satisfactory for a broad range of applications. For example, a HDTV-resolution video-stream generates 30 frames/sec, while a low quality video-stream generates 8 to 15 frames/sec. Data rates depend on multiple factors, including the encoding scheme used, display size, and color depth. A compressed 15 frames/sec, 240x340 pixels video clip might generate 100Kbps.

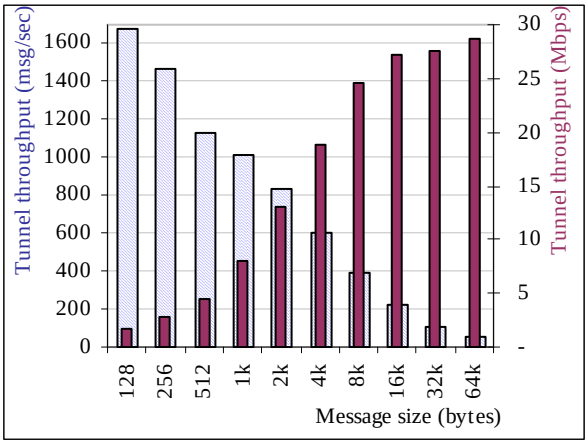

Figure 3: Node routing performance. Maximum loss-free tunnel throughput as a function of message size: messages/sec (stripes, left scale) and bandwidth (full bars, right scale). Messages from a single incoming tunnel are forwarded on each of three outgoing tunnel. For messages larger than 4kB the bottleneck is the network card on the node.

B. Real world performance: experiments on PlanetLab

We have deployed and tested UMM on the PlanetLab [40] wide-area, shared network testbed. In the following subsection we characterize the subset of PlanetLab nodes we use for tests aiming to give a baseline against which UMM performance can be evaluated. Sub-sections ii and iii present two sets of experiments testing UMM in single- and multi-source multicast scenarios.

i.) PlanetLab: testbed characterization

For our tests we have used more than 60 PlanetLab nodes distributed over more than 55 sites, and one additional, non-PlanetLab node at the University of

Chicago (UC). While the bulk of our nodes are located on the American continent, 12 nodes are located in Europe and 2 in Asia.

Since the UC node is used as the source in next section's single-source multicast experiments, Figure 4 and 5 present the characteristics of unicast paths between the UC node and each other node in our testbed. We use *netperf* [41] to estimate latency and available bandwidth both in the data reported below and in our UMM deployment. Figure 5 shows that estimates of available bandwidth change little when using probes that send data for 10 seconds compared to sending data for one second only.

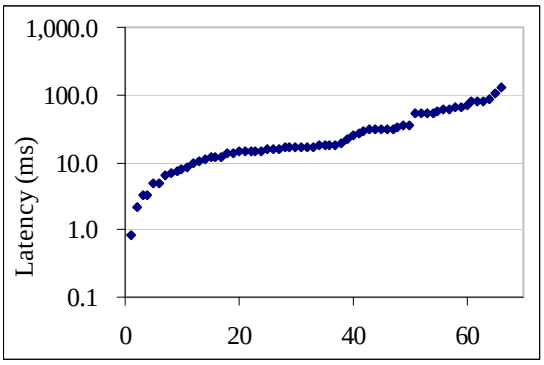

Figure 4: Latency distribution. Latency distribution from UC node to all nodes used. Nodes are ordered in increasing order of their latency to the source. (log scale on y-axis).

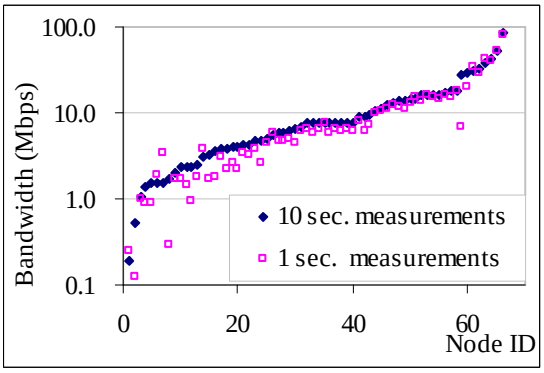

Figure 5. Available bandwidth distribution as measured with *netstat* from the UC node to all other nodes used. No significant loss of accuracy is observed when the probing time is one second only. Nodes are ordered in increasing order of their bandwidth to the UC node as reported by 10s measurements.

Figure 6 presents the cumulative distribution function for latency and available bandwidth between all pairs of nodes used in experiments. We note the heterogeneity offered by the testbed: estimated values for both metrics stretch over more than two orders of magnitude.

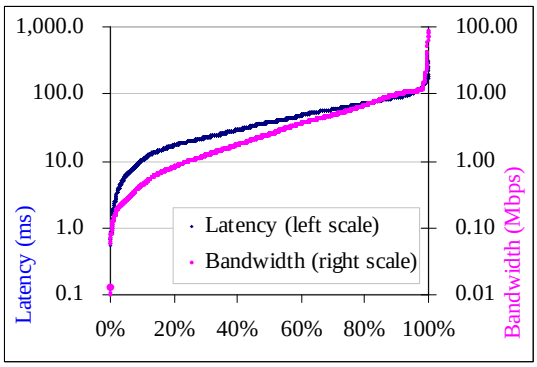

Figure 6: All pair cumulative distributions for estimated latency (upper plot, left axis) and bandwidth (lower plot, right axis)**.** Logarithmic scale on both y-axes.

ii.) Single-source multicast performance

Since UMM builds distribution that trees are independent for each source we first evaluate UMM's single-source multicast performance. Figure 7 presents received data rates at each destination after streaming data from a single source (UC node) at rates varying from 128kbps to 2048kbps. Experiments take place 20 minutes after bootstrap to allow the base overlay to adapt and stabilize. After each experiment, that is, after trying each sending rate, the application overlay is reset.

Depending on the streaming rate, between 78% and 95% of all nodes receive all data sent. Between 85% and 97% of all nodes receive more than 95% of all data sent. Note that, for a few nodes, UMM achieves a delivery rates better than the TCP unicast performance directly from the source (this is visible when comparing Figure 5 and the streaming performance presented in Figure 7).

Dropped packets may occur for two reasons even in a 'perfect' overlay. First, a destination node might not be able to keep up with the source's sending rate, either because its uplink is congested or, with similar effects, due to per-user send/receive rate caps imposed at some PlanetLab nodes. In this case, tunnel queues that lead to such nodes build up, and messages are eventually dropped. Second, if the adaptation thread kicks in during the test and tunnels are removed, some destinations might find themselves disconnected from the source in the application overlay for a short period.

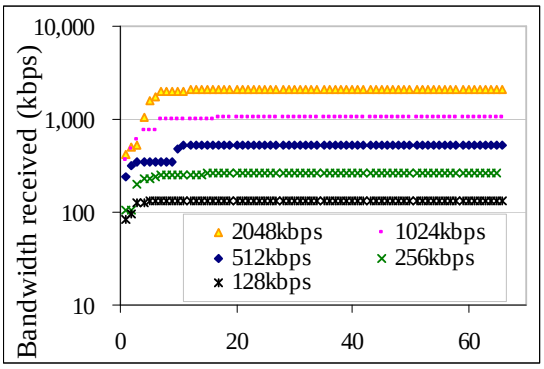

Figure 7: Single-source multicast performance for sending data rates from 128kbps up to 2048kbps. The graph presents received data rates at each destination. Nodes are ordered in increasing order of their received rates.

At the application overlay level, our scheme generates two types of overheads. First, a few duplicate messages flow in the overlay each time a new multicast source kicks in and each time a tunnel's filter times out. Second, control messages are generated to extract the application overlay from the base overlay, to announce new nodes in the network, and to maintain connectivity. Figure 8 presents these overheads in terms of both the number of overhead messages and consumed bandwidth. The average bandwidth overhead is about 1.5%. We believe that some control messages can be piggybacked on data messages thus reducing the overhead in terms on generated messages. This the focus of ongoing work.

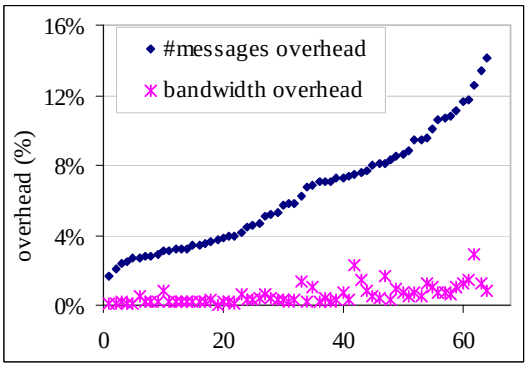

Figure 8. Overheads distribution in terms of number of messages (blue) and bandwidth (red). Sending rate: 256kbps.

iii.) Multi-source multicast performance

Figure 9 presents received data rates at each destination when all nodes send data at rates varying from 2kbps to 16kbps, for an aggregate between 120kbps and 960kbps. Note that there are fewer messages lost than for the single source multicast experiment presented in Figure 7, as network paths are loaded more evenly in this case. (Load on PlanetLab was too high before the deadline for this paper to attempt experiments with higher sending rates to match and exceed those used for the single-source case)

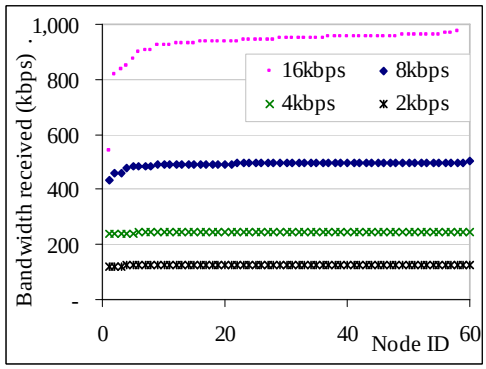

Figure 9: Multi-source multicast performance. Each of 60 nodes sends data at rates from 2kbps to 16kbps. The graph presents received data rates at each destination. Nodes are ordered in increasing order of their received rates. (Normal scale on y-axis, unlike in).

We now turn to the two measures of performance (RDP and stress) proposed earlier. Each point in Figure 10 represents a pair of nodes in the application overlay: the x-axis is the delay between the two nodes in the underlying IP network, while the y-axis is the estimated delay between the same two nodes in the overlay. Note that most points lie between the two gray lines representing RDP=1 and RDP=2. The few points that lie below the RRP=1 line correspond to cases in which the application overlay offers better delay than IP routing.

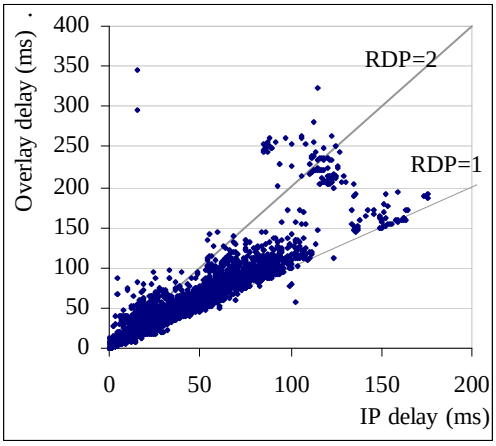

Figure 10. **All pairs delay** for multi-source multicast (60 nodes). Note that most points are between the lower, gray line (RDP=1) and the diagonal (RDP=2)

There are two approaches to estimating propagation delay in the application overlay. The *offline* approach is to extract a map of the overlay and combine it with all-pairs IP-level delay estimations to estimate (offline) the delay of an overlay path. We use this approach for the data presented in Figure 10 and for the red (left) plot in Figure 11. The *online* approach estimates the propagation delay through the overlay itself: once the application overlay is built, a node multicasts a special *ping* message. Each node receiving the ping forwards it further in the application overlay as if it were a normal

data message, but also sends back a *reply*. Nodes that receive the reply propagate it back to the source through the overlay. Once the source receives a reply, it can compute the overlay delay to the node that generated the reply. Delay estimates using this method are inherently more pessimistic: the source needs to process, in sequence, a potentially large number of replies. Additionally, because the shared PlanetLab nodes are heavily loaded (often load is above 10), the delays between the intervals when our application is scheduled can be large, sometimes in the same order of magnitude as IP-level delays. Consequently the RDP computed using online estimates of delay (blue, left plot in Figure 11) are significantly larger (see Table I). We have not observed such delays when running on ModelNet (as we discuss in Section C), even when running ten copies of our application concurrently on the same physical node. Thus, we believe that the poor online RDP estimates are simply a result of the high, sometimes transitory, load at testbed nodes.

Table I: The difference between offline and online RDP estimates. "95% RDP" is the value larger than 95% of all RDP estimates.

	Offline delay estimate	Online delay estimate
95% RDP	2.53	4.96
90% RDP	1.95	3.78
Median	1.22	2.11

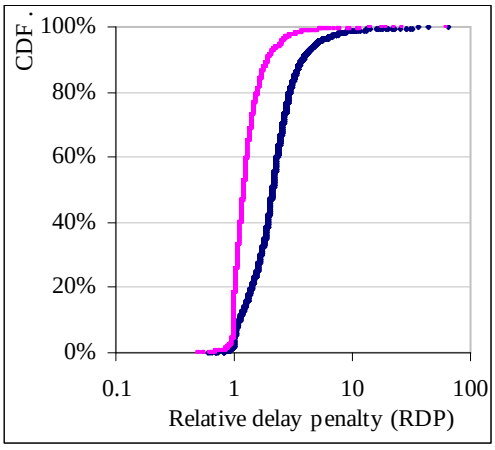

Figure 11: All pairs RDP cumulative distribution function for multi-source multicast (60 nodes). Red (left) plot: offline delay estimation; blue (right) plot: online delay estimation.

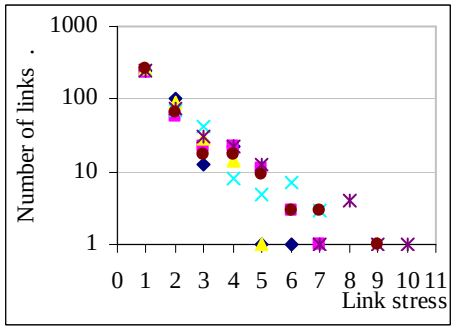

Figure 12: Link stress distribution for six different 60-node application overlays. Each set of points represents an estimated link stress distribution for one such overlay.

We turn now to estimate link stress distribution the second overlay efficiency metric proposed. To this goal, we use *tracepath* to discover intermediary routing hops and build an approximate map of the underlying network. However, about 16% of all tracepath runs fail at different stages. As a result, the inferred topological map and link stress distribution (presented in Figure 12) are only coarse estimates of actual values. The low stress values presented in Figure 12 are unsurprising since our deployment is relatively sparse for its wide-area distribution compared to the density of Internet.

iv.) Adaptation

Every $T_{optimize}$=90s, nodes run an optimizer thread that implements the adaptation techniques described in Section III.D. The optimizer randomly selects up to 10 nodes from the set of nodes known locally; refreshes, if necessary, latency and available bandwidth estimates to these nodes; and, based on the heuristics presented, decides whether to replace an existing tunnel.

Figure 13 presents relative delay penalty (RDP) evolution for the first 1200s of an overlay's life. Immediately after all nodes are started, and then every 240s, we measure RDP in the application overlay extracted for the UC source. As the figure shows, after 480s the RDP distribution is already close to its stable value.

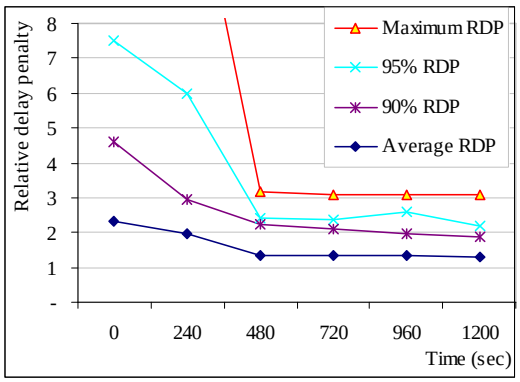

Figure 13: Adaptation during the first 1200s of the overlay life: evolution of the relative delay penalty (RDP) measured from the UC node to all other nodes. For clarity maximum RDP values (16.1 and 17.3 respectively) for the first two measurement intervals are not presented.

C. *Experiments in an emulated environment*

ModelNet [42] is a network emulation environment for wide-area systems. We experiment with UMM in this controlled environment with two goals in mind: first, to compare UMM's RDP and stress with those of other approaches reported in a previous study [27]; second, to test UMM's ability to operate and adapt when the set of nodes is dynamic.

Overlay performance is highly dependent on underlying network characteristics. We attempt to reproduce the physical topologies used by Jain et al. [27] to compare UMM and alternative solutions on a common base. summarizes the results of this comparison for a 64-node overlay. Although UMM compares favorably to alternative solutions, we see these results simply as an indication that reinforces our intuition that UMM generally performs *at least as well as* these alternatives. We plan to explore UMM at larger scales and with different physical network topologies while benchmarking against alternative solutions.

Table II: 90% RDP and maximum link stress for UMM and other multicast solutions for a 64-node overlay. Each pair of numbers is the minimum and maximum values over 10 runs. (All non-UMM results are extracted from Jain et al. [27]).

	90% RDP	Maximum link stress
Narada	2.0 - 2.6	12 - 14
DHTs	4.2 - 6.0	7 - 10
Idealized DHTs	2.1 - 3.5	
Flooding on power-law, random topology	2.6 - 3.2	20 - 30
UMM	2.2 - 2.6	6 - 10

Using the same physical topology and 64-node overlay, we explore UMM's ability to operate in a dynamic environment. We consider a case when 10 nodes fail and then later rejoin the overlay. Both failure and re-join events are distributed over a 120s interval. Figure 14 presents RDP distribution and maximum link stress evolution over time. 700s after bootstrap, RDP and stress are already close to their stable values. Since failures are distributed over a relatively large interval, the base overlay recovers from these failures quickly and RDP increase is not significant. Link stress actually decreases as fewer tunnels are mapped to the physical network. The impact of node joins is larger as joining nodes take time to discover existing nodes suitable to create overlay tunnels to.

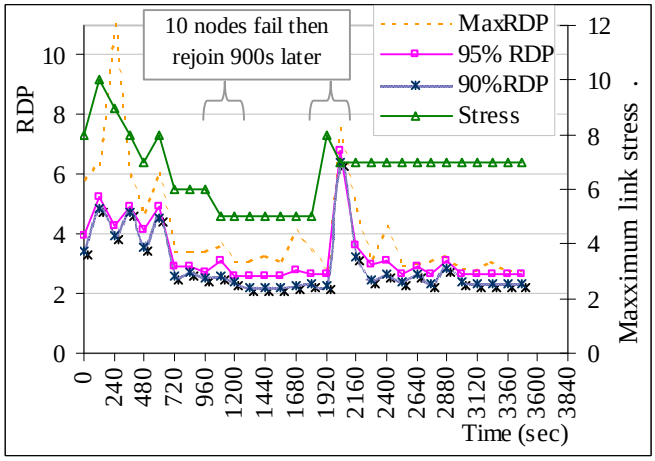

Figure 14: Adaptability. RDP and maximum link stress evolution for a distribution tree rooted at one random node. 900s after the 64-node overlay is started, 10 nodes fail during a 120s interval. The 10 nodes then rejoin 900s later, again during a 120s interval.

VII. SUMMARY AND FUTURE WORK

We have argued that is possible to design an unstructured multi-source multicast overlay that is less complex, but as efficient as, current state-of-the-art solutions, whether based on structured overlays or on running full routing protocols at the overlay level. In particular, we show that our UMM solution is:

- *Efficient.* Experience gained with deployments in live and emulated environments indicates that UMM both generates low overheads and performs well compared to alternative solutions in terms of RDP and link stress.
- *Simple.* UMM preserves the simplicity of flooding based solutions and the flexibility of unstructured overlays. It decouples message routing and overlay construction so that the two can be optimized separately. Node state is collected passively, by detecting duplicates in the message flow.
- *Adaptive.* UMM adapts to the underlying network topology, and performs well when the set of participating nodes is dynamic.

We are working to extend these results in multiple directions. We are using ModelNet to test UMM scalability limits and to compare with structured overlay approaches at larger scales. More work is required to evaluate the impact of varying the various parameters introduced in our algorithm, and/or to develop automated techniques for selecting and perhaps evolving their values. We also plan to explore, via simulations, additional heuristics to configure and adapt the base overlay. One intriguing idea is to investigate the benefits of adding preferential attachment [43] (nodes prefer to connect to well-connected nodes) to our heuristic.

We used link stress as one success metric for our overlay topologies. Building on the intuition that it is important to maintain low stress on bottleneck links, an additional, performance oriented metric is relative stress: the ratio between link capacity and its stress. We plan to explore heuristics that improve on this metric.

Finally, we plan to build a number of services (e.g., replica location [44] and other discovery services, file replication, video-stream delivery) on UMM to test its performance in realistic operational settings.

VIII. ACKNOWLEDGEMENTS

This work was supported in part by IBM and by the NSF GriPhyN project. We are grateful to Jennifer Schopf for comments on a draft of this paper.

IX. REFERENCES

1. Ratnasamy, S., et al. *Application-level Multicast using Content-Addressable Networks*. in *NGC 2001*. 2001.
2. Castro, M., et al., *SCRIBE: A large-scale and decentralised application-level multicast infrastructure.* IEEE Journal on Selected Areas in Communications (JSAC) (Special issue on Network Support for Multicast Communications), 2002.
3. Stoica, I., et al. *Internet Indirection Infrastructure*. in *ACM SIGCOMM'02*. 2002. Pittsburgh, PA.
4. Zhuang, S.Q., et al. *Bayeux: an architecture for scalable and fault-tolerant wide-area data dissemination.* in *NOSSDAV'01*. 2002. Port Jefferson, NY.
5. El-Ansary, S., et al. *Efficient Broadcast in Structured P2P Networks*. in *IPTPS'03*. 2003.
6. Faloutsos, M., P. Faloutsos, and C. Faloutsos. *On Power-Law Relationships of the Internet Topology*. in *SIGCOMM 1999*. 1999.
7. Jin, S. and A. Bestavros. *Small-world Characteristics of Internet Topologies and Multicast Scaling*. in *IEEE/ACM MASCOTS 2003*. 2003.
8. Chawathe, Y., *Scattercast: An Adaptable Broadcast Distribution Framework.* ACM Multimedia Systems Journal special issue on Multimedia Distribution, 2002.
9. Chu, Y.-h., et al. *A Case for End System Multicast*. in *SIGMETRICS*. 2000.
10. Foster, I., C. Kesselman, and S. Tuecke, *The Anatomy of the Grid: Enabling Scalable Virtual Organizations.* International Journal of High Performance Computing Applications, 2001. **15**(3): p. 200-222.
11. Diot, C., W. Dabbous, and J. Crowcroft, *Multipoint Communication: A Survey of Protocols, Functions, and Mechanisms.* JSAC, 1997. **15**(3).
12. Saltzer, J.H., D.P. Reed, and D.D. Clark, *End-to-end arguments in system design.* ACM Transactions in Computer Systems, 1984. **2**(4): p. 277-288.
13. Clark, D. and D. Tennenhouse. *Architectural Considerations for a New Generation of Protocols*. in *SIGCOMM*. 1990.
14. Nakao, A., L. Peterson, and A. Bavier. *A Routing Underlay for Overlay Networks*. in *SIGCOMM'03*. 2003. Karlshue, Germany.
15. Li, B., J. Guo, and M. Wang. *iOverlay: A Lightweight Middleware Infrastructure for Overlay Application Implementations*. in *Middleware 2004*. 2004. Toronto, Canada.
16. Chu, Y.-h., et al., *A Case for End System Multicast.* IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast, 2002. **20**(8).
17. Chu, Y.-h., et al. *Enabling Conferencing Applications on the Internet Using an Overlay Multicast Architecture*. in *SIGCOMM 2001*. 2001. San Diego, CA.
18. Deering, S. *Multicast routing in internetworks and extended LANs.* in *SIGCOMM*. 1988. Stanford.
19. Banerjee, S., B. Bhattacharjee, and C. Kommareddy. *Scalable Application Layer Multicast.* in *SIGCOMM2002*. 2002. Pittsburgh, PA.
20. Zhang, B., S. Jamin, and L. Zhang. *Host Multicast: A Framework for Delivering Multicast to End Users*. in *INFOCOM 2002*. 2002.
21. Jannotti, J., et al. *Overcast: Reliable Multicasting with an Overlay Network*. in *4th Symposium on Operating Systems Design and Implementation (OSDI 2000)*. 2000. San Diego, California.
22. Pendarakis, D., et al. *ALMI: An Application Level Multicast Infrastructure*. in *USITS'01*. 2001.
23. Ratnasamy, S., et al. *A Scalable Content-Addressable Network*. in *SIGCOMM 2001*. 2001. San Diego USA.
24. Rowstron, A. and P. Druschel. *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*. in *IFIP/ACM International Conference on Distributed Systems Platforms (Middleware)*. 2001. Heidelberg, Germany.
25. Zhao, B.Y., J.D. Kubiatowicz, and A.D. Joseph, *Tapestry: An infrastructure for fault-tolerant wide-area location and routing*. 2001, UC Berkeley.
26. Stoica, I., et al. *Chord: A Scalable Peer-to-peer Lookup Service for Internet Applications*. in *SIGCOMM 2001*. 2001. San Diego, USA.
27. Jain, S., R. Mahajan, and D. Wetherall. *A Study of the Performance Potential of DHT-based Overlays*. in *4th USENIX Symposium on Internet Technologies and Systems*. 2003. Seattle, WA.
28. Shen, K. *Structure Management for Scalable Overlay Service Construction*. in *First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI'04)*. 2004. San Francisco, CA.
29. Shen, K. and Y. Sun. *Distributed Hashtable on Pre-Structured Overlay Networks*. in *9th International Workshop on Web Caching and Content Distribution (WCW'04)*. 2004. Beijing, China.
30. Rodriguez, A., et al. *MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks*. in *OSDI*. 2004.
31. Birman, K.P., et al., *Bimodal Multicast.* ACM Transactions on Computer Systems (TOCS), 1999. **17**(2): p. 41 - 88.
32. Bustamante, F.E. and Y. Qiao. *Friendships that last: Peer lifespan and its role in P2P protocols*. in *Workshop on Web Content Caching and Distribution*. 2003. Hawthorne, NY.
33. Ripeanu, M., 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, 2002. **6**(1).
34. Castro, M., et al. *Topology-aware routing in structure peer-to-peer overlay network*. in *International Workshop on Future Directions in Distributed Computing (FuDiCo)*. 2002. Bertinoro, Italy.
35. Xu, Z., C. Tang, and Z. Zhang. *Building Topology-Aware Overlays using Global Soft-State*. in *23rd International Conference on Distributed Computing Systems (ICDCS)*. 2003. Providence, Rhode Island.
36. Ratnasamy, S., et al. *Topologically-Aware Overlay Construction and Server Selection*. in *INFOCOM'02*. 2002. New York.
37. Waldvogel, M. and R. Rinaldi, *Efficient topology-aware overlay network.* Computer Communication Review, 2003. **33**(1): p. 101-106.
38. Cox, R. and F. Dabek, *Learning Euclidean Coordinates for Internet Hosts*. 2002.
39. Watts, D. and S. Strogatz, *Collective dynamics of 'small-world' networks.* Nature, 1998. **393**.
40. Bavier, A., et al. *Operating System Support for Planetary-Scale Network Services*. in *NSDI'04*. 2004. San Francisco, CA.
41. . *http://www.netperf.org*. 2004.
42. Vahdat, A., et al. *Scalability and Accuracy in a Large-Scale Network Emulator*. in *OSDI*. 2002.
43. Barabasi, A.-L. and R. Albert, *Emergence of scaling in random networks.* Science, 1999. **286**: p. 509-512.
44. Foster, I., et al. *Giggle: A Framework for Constructing Scalable Replica Location Services*. in *SC2002*. 2002. Baltimore, Maryland