Scalability in Adaptive Multi-Metric Overlays *

Adolfo Rodriguez and Dejan Kostić
Dept. of Computer Science
Duke University
Box 90129
Durham, NC 27708
razor,dkostic@cs.duke.edu

Amin Vahdat †
Dept. of Computer Science and Engineering
University of California, San Diego
9500 Gilman Drive, Dept. 0114
La Jolla, CA 92093
vahdat@cs.ucsd.edu

Abstract

Increasing application requirements have placed heavy emphasis on building overlay networks to efficiently deliver data to multiple receivers. A key performance challenge is simultaneously achieving adaptivity to changing network conditions and scalability to large numbers of users. In addition, most current algorithms focus on a single performance metric, such as delay or bandwidth, particular to individual application requirements. In this paper, we introduce a two-fold approach for creating robust, high-performance overlays called Adaptive Multi-Metric Overlays (AMMO). First, AMMO uses an adaptive, highly-parallel, and metric-independent protocol, TreeMaint, to build and maintain overlay trees. Second, AMMO provides a mechanism for comparing overlay edges along specified application performance goals to guide TreeMaint transformations. We have used AMMO to implement and evaluate a single-metric (bandwidth-optimized) tree similar to Overcast and a two-metric (delay-constrained, cost-optimized) overlay.

1. Introduction

The growing popularity of distributed applications has led to demands for increasing functionality from the underlying network. One example is multicast, where data is simultaneously disseminated to multiple receivers. While some efforts retrofit such functionality into the network fabric (e.g. IP multicast), these have been largely limited by administrative, scalability, and security problems. Overlay networks, where nodes self-organize to create a logical network capable of efficiently disseminating data to receivers [1, 5, 6, 7, 11], are an increasingly viable option for delivering improved functionality and semantics. Here, a spanning tree is formed by selecting $n-1$ overlay edges of an n-node overlay from the n^2 possibilities based on perceived performance characteristics (e.g., bandwidth and delay). Each participant chooses a parent that allows it to satisfy performance criteria by comparing potential overlay edges. The sequential version of this problem when dealing with multiple metrics is typically NP-hard [20, 22] even with global knowledge under non-realistic, static network conditions. As a result, many efforts consider only a single performance metric. Further, researchers have been bothered by cumbersome overlay maintenance algorithms that tightly couple performance requirements with the mechanisms employed to improve the overlay. Our work focuses on the observation that performance specification and overlay maintenance are largely independent and should be decoupled.

Overlay edge weights are typically inferred through the use of coordinated probes that measure, for example, round-trip time or available bandwidth. To adapt to dynamic network conditions, nodes continuously probe other participants, constantly searching to improve the overlay. Given that nodes join and leave the overlay at high rates, we require fresh membership information to be readily available. To ensure scalability, however, no node can probe more than $O(\lg n)$ participants, nor can we require global group membership or global locking; no node can keep state about all other nodes in the overlay. An overlay's maintenance protocol cannot require $O(n)$ communication to ensure correctness. Naturally, requiring high probing and fresh group membership for adaptivity and low probing with limited group membership for scalability effects a trade-off in the quality of the overlay. Hence, our goal is to create an algorithm that delivers sufficient flexibility in adjusting to changing network conditions while not requiring large amounts of state or communication that would limit scalability.

* Supported in part by the National Science Foundation (EIA-99772879, ITR-0082912), Hewlett Packard, IBM, Intel, and Microsoft.
† Supported by an NSF CAREER award (CCR-9984328).

In this paper, we present the design of a family of distributed algorithms, AMMO (Adaptive Multi-Metric Overlays), aimed at bridging the gap between scalability and adaptivity in overlays. A key insight behind AMMO is the distributed construction of ordered random subsets of global overlay participants [10]. AMMO nodes use these subsets (restricted to $O(\lg n)$ size) to probe overlay peers for better connectivity. We develop an overlay maintenance protocol called TreeMaint to enable our overlays to quickly converge to a tree that meets a set of application-specific performance requirements without introducing loops into the overlay and without distributed locking. Each AMMO application specifies prioritized root-to-leaf metric constraints, $C_1 \ldots C_k$. For example, a video application could specify a maximum root-to-leaf delay of 200 ms. In this respect, AMMO attempts to approximate a shortest path tree (SPT) to within some constraint along the desired metric. Once a constraint is met, AMMO proceeds with lower priority optimization targets.

In addition, an application defines a performance metric function, F, of a directed overlay edge e:

$$F(e) = \sum_{i=1}^{k} A_i M_i(e) \tag{1}$$

where $M_i(e)$ represents e's weight along some metric i. For example, for a delay metric, $M_i(e)$ could represent the end-to-end delay from e's source to e's destination. A_i is a normalizing scalar that enables applications to favor certain performance metrics over others, describing their relative priority. In this regard, AMMO approximates a minimum spanning tree (MST) where each node attempts to greedily decrease its distance to its parent as measured by the metric function. Our goal is to minimize the sum of $F(e)$ for all overlay edges, while falling within application metric constraints. By specifying performance via constraints and a metric function, AMMO can provide the necessary maintenance for overlays such as Overcast [7] (with metric function consisting of bandwidth) and SARO [10] (with delay constraint and bandwidth metric function). As such, one can view AMMO as a superset of these algorithms, providing scalability and adaptivity to tree-based protocols.

The remainder of this paper is organized as follows. Section 2 places our work in the context of related efforts. Section 3 presents TreeMaint, a fully decentralized, metric-independent algorithm for overlay transformations and provides a proof of its correctness. Next, Section 4 describes the metric function and shows how systems such as Overcast and SARO can use it to describe the performance metrics they aim to optimize. We present the details of a fully functional distributed implementation of AMMO in Section 5. We evaluate our AMMO implementation using 1000-node overlays running across an accurately emulated wide-area network [21] and show that AMMO quickly converges to nearly-optimal trees for a variety of network conditions and performance targets. We conclude in Section 6.

2. Related Work

AMMO is closely related to RanSub [10], which provides a mechanism for uniform [1] random subset delivery and has been used to build scalable network services such as SARO [10] and Bullet [11]. RanSub periodically collects random subsets from member nodes and distributes them to each participant. To show the utility of RanSub, SARO used it to construct a delay-constrained, bandwidth optimized tree. In contrast, this work extends overlay construction to an arbitrary number of performance metrics. We describe TreeMaint, our technique for building loop-free trees using RanSub in full detail. We also provide a proof of TreeMaint's correctness. Finally, we use AMMO to implement and evaluate Overcast and a delay-constrained, cost-optimized tree.

The problem we address is related to multicast QoS routing, where an algorithm may recruit intermediate points (routers). In our case, intermediate routers cannot participate in the overlay. This reduction in freedom only aids the single-metric optimization problem (a polynomial sequential algorithm exists for MSTs). Wang and Crowcroft [22] have shown that the problem of finding a network path subject to multiple constraints is generally NP-hard. The exception is when bandwidth is one of the metrics, since the bandwidth among a path is the minimum of the bandwidths of constituent links. Most recent approximation efforts in bi-criteria network optimization problems offer sequential algorithms [3, 12] that require full topology information. Without global knowledge, these algorithms cannot be applied to large-scale overlay networks. Kompella et al. [8] have extended their work on delay-constrained minimum cost multicast trees [9] into the distributed setting. However, they assume the presence of a distance-vector algorithm that provides each node with delay to all other nodes. This approach does not scale or is not adaptive if routing updates are delayed to make it scalable.

Narada [5, 6] builds a mesh interconnecting all nodes on which it runs a standard multicast routing protocol. Narada only optimizes for two metrics, one of which is bandwidth, making the problem not NP-hard. Narada also does not allow metric constraints as done by AMMO. Finally, Narada nodes maintain global knowledge about all group participants. While feasible for their target of video conferencing among a few dozen nodes, such a requirement does not scale to the systems we consider.

1 While RanSub's uniformity is emperically validated in [10], a small modification leads to a simple uniformity proof as shown in [16].

In Overcast [7], all nodes join at the root and migrate down the tree while not sacrificing bandwidth. Overcast periodically allows nodes to move up and down the tree to react to changing network conditions. To avoid loops, Overcast requires knowledge of a node's ancestors, limiting its ability to scale since its tree height tends to grow unbounded. An alternate approach is to require the use of distributed locks to synchronize transformations. While such an approach resolves the scalability problems of Overcast's maintenance algorithm, it comes at the price of decreased adaptivity. A third approach is to use AMMO for Overcast transformations. While Overcast uses a restrictive policy for choosing potential parents (grandparents and siblings), AMMO nodes can choose from a wider range of potential parents. For illustrative purposes, however, we have modified our RanSub implementation to only deliver siblings and grandparents in subsets and as such, have successfully implemented Overcast within the AMMO framework.

Finally, a number of recent efforts [14, 19, 24] propose building application-layer multicast on top of scalable peer-to-peer lookup infrastructures [4, 15, 18, 23]. While demonstrating that it is possible to probabilistically achieve good delay relative to native IP multicast, they are unable to provide performance bounds because of the probabilistic nature of the underlying peer-to-peer system. Further, they do not consider multi-metric network optimizations.

3. Overlay Transformations with TreeMaint

In this section, we describe TreeMaint, a distributed algorithm for building and maintaining scalable and adaptive overlays. Each TreeMaint node keeps state regarding addresses of its parent and children, sequence number of the current synchronization *epoch*, total number of overlay participants, and number of descendants in the subtree it roots. An algorithm that uses TreeMaint, such as AMMO (described in Section 4), maintains additional state to determine which overlay transformations should be made. Such information includes performance metrics, such as measures of delay or bandwidth, though their specification is orthogonal to TreeMaint operation.

3.1. RanSub

Given global knowledge and plentiful bandwidth, the operation of overlay nodes would be relatively straightforward. Each node would simply probe the paths between it and all remote participants to find the parent that best meets application goals. However, for scalability, we cannot impose the requirement that any node has such global knowledge. Hence, we rely on the probabilistic distribution of uniformly random subsets (of configurable size) to each node once per time epoch, using the RanSub [10] utility. The contents of these random subsets change from one epoch to the next. TreeMaint nodes use these size-limited random subsets to probe remote participants for better connectivity. In this way, we ensure that no more than $O(n \lg n)$ probes are performed during any epoch. For example, we find that a 1000-node AMMO tree using a small epoch imposes an acceptable per-node probing overhead of 2 KBps.

RanSub operation requires the use of an underlying overlay tree, which, in our case, is provided by TreeMaint itself. A RanSub epoch (identified by increasing sequence numbers) consists of two sequenced phases. A *collect phase* sweeps up the tree, where each node successively constructs a random *collect set* and sends it to its parent. A *distribute phase* sweeps down the tree, where a node constructs *distribute sets* for its children using previously received collect sets and its own distribute set received from its parent. When a leaf node receives a Distribute message with a random subset, it initiates a collect phase by sending to its parent a Collect message containing only itself. Parents having sent Distribute messages to children will receive Collect messages from them before propagating a Collect upward. A parent uses the contents of received Collect messages to construct an *ordered* random subset of nodes uniformly representative of all participants in the subtree it roots.

The root uses the contents of received Collect messages to send a Distribute message to each of its children. For this epoch, the root establishes a random ordering among its children. It includes in a Distribute message to a child only subsets received from children preceding the child in the current ordering. A node receiving a Distribute message in turn uses previously received collect sets and its transient ordering among its children to construct Distribute messages to propagate down the tree. The ordered flavor of RanSub ensures that a node receives subsets consisting only of nodes *preceding* it in the current transient total ordering (one ordering per epoch). This ultimately ensures TreeMaint's ability to make simultaneous transformations without introducing loops.

3.2. TreeMaint Operation

TreeMaint nodes attempt to initiate a tree transformation when directed to do so. While other algorithms that use TreeMaint are free to choose the appropriate conditions for transformations, the criteria for initiating AMMO transformations is further described in Section 4. When a node A wishes to move under S, it sends a *transformation request* to S along with the current epoch sequence number (received in the RanSub Distribute message). If this sequence number matches S's own current sequence number, the transformation is accepted and is acknowledged by S to Y via a *transformation response* message. The use of a sequence number and a total ordering among members of the random subset disallows any operation that would create a loop in the tree; a proof is provided in Section 3.3.

There are a number of potential interactions between the timing of epochs and TreeMaint transformations. The sequence number of the current epoch allows nodes to identify which total ordering was used when initiating a transformation. A node receiving a Distribute message with sequence $i+1$ receives a distribute set with sequence number i and is said to be *operating under ordering* O_i. If a potential parent receives a transformation request for a node operating under a different sequence number, it rejects the request. There may, however, be cases in which a node and the subtree it roots are not included in this total ordering because the node makes a transformation between distribute and collect phases or during failure recovery scenarios. Additionally, TreeMaint may suppress subset inclusion in cases where a subtree might be reported in more than one location in the total ordering.

Consider a scenario where a node C moves from X to Y using ordering O_i (having received a Distribute message with sequence O_{i+1}). Three interesting cases arise. First, node C could have moved after transmitting a Collect message with sequence $i+1$ to X. In this case, C's subtree is reported to X for ordering O_{i+1}. Upon making the move under Y, C will not send Y a Collect message with sequence $i+1$, nor will Y expect such a message from C. Alternatively, node C could have moved before it had a chance to transmit a Collect message with sequence $i+1$ to X. Node C will have an outstanding distribute sequence in its subtree. Upon accepting C, Y will not expect a collect sequence i from C, but will receive it anyway because the distribute is outstanding in C's subtree. Y will ignore this message. X, which was expecting to receive a Collect message with sequence $i+1$ from C, will resolve this expectation at the time C is removed from X. In this case, the subtree rooted at C will be excluded from the total ordering of this sequence. Finally, a failure recovery scenario could lead to neither X nor Y receiving a Collect message with sequence $i+1$ in which case C and the subtree it roots are excluded from the total ordering O_{i+1}.

3.3. TreeMaint Correctness

In this section, we prove that TreeMaint maintains a connected tree without loops under concurrent transformations. Because of space limitations, we omit some details of the proof. We use the following notation: if B precedes A in the total ordering, then we write $B \prec A$. Likewise, if B does not precede A, we write $B \nprec A$.

Lemma 1 *If A is in a subtree that moves under B using O_i, then $A \nprec B$ in O_{i+1}.*

Proof: Let G be the ancestor of A that moves under B and H be G's old parent. Then for this move, B and G are operating under O_i and have not yet received O_{i+1}. There are three cases to consider. If H received collect $i+1$ from G before the move, any move that could push H in front of B will result in the squashing of collect $i+1$ with G potentially in it. It follows that $G \nprec B$ in O_{i+1}. If B received collect $i+1$ from G, then clearly, $G \nprec B$ in $i+1$. Finally, if no node processed a valid collect $i+1$ from G. then G is not present at all in O_{i+1} and therefore $G \nprec B$ in O_{i+1}.

Lemma 2 *If node A is operating with O_i, its descendant B is operating with O_{i-1} or O_i. Likewise, if B is operating under O_i, its ancestor A is operating with O_i or O_{i+1}.*

Lemma 3 *If A is a descendant of B, then $A \nprec B$ in A's current operating ordering O_i.*

Proof: If A sent a Collect i with O_i after its was a descendant of B, then $A \nprec B$. If A sent Collect i via some other node, it must have been operating under O_{i-1}. If it moved while operating under O_{i-1}, then $A \nprec B$ in O_i. If it moved while operating under O_i, then $B \prec A$ in O_i and hence $A \nprec B$ in i.

Lemma 4 *If a Distribute $i+1$ is in flight from ancestor A to descendant B, either B receives this Distribute or moves away from under A.*

Theorem 1 *No node can be a child of its own descendant.*

Proof by contradiction: Assume that a node R has made a move using O_i and has become a child of one of its descendants, Z. This means that $Z \prec R$ in O_i and some move occurred that made Z a descendant of R. Namely, some ancestor Y of Z (perhaps Z itself) became the child of some descendant S of R (perhaps R itself). If Z was in a subtree that moved under R using O_k, then $Z \nprec R$ in O_{k+1}. Hence, $k \neq i-1$, because $Z \prec R$ in O_i. (Lemma 1)
Additionally, because at the time of the move R was operating under O_{i-1} or O_i, S and Y were operating under O_{i-2} or O_i (not $i-1$). (Lemma 2)
If R was operating under O_{i-1}, then S was operating under O_{i-2}. The only way that could happen is if a distribute delivering ordering O_{i-1} was in flight from R to S. Because S must either receive this distribute or move away from being a descendant of R, it must have received it. Hence, S received at least O_{i-1} and $k \neq i-2$. (Lemma 4)
Since $k \neq i-1$ and $k \neq i-2$, then $k = i$. Nodes S and Y operated under O_i. Hence $S \prec Y$ in O_i. Now since Z is a descendant of Y and is operating under O_i, O_i must show $Y \prec Z$. (Lemma 3)
Likewise, since S is a descendant of R and is operating under O_i, O_i must show $R \prec S$. Thus, $R \prec S$, $S \prec Y$, $Y \prec Z$ in O_i, implying that $R \prec Z$ in O_i, which contradicts our initial assumption that $Z \prec R$ in O_i.

3.4. Dealing With Constrained Degree

To limit the amount of per-node resources, TreeMaint allows a degree constraint to be placed on individual nodes. The algorithm disallows transformations that would violate this degree constraint. In doing so, the greedy nature of the algorithm may lead to sub-optimal overlays. Consider the following situation. A node A with a particular degree constraint has a full complement of children. However, a node B somewhere else in the overlay can only achieve its metric constraint by becoming a child of A. Finally, one of A's children, C, is best served by A but it would still be able to achieve its metric constraint as a child of some fourth node D. As described thus far, TreeMaint would become stuck in a "local minimum" in this situation. We address this situation by adding *wean* operations in which a child is purposely asked to leave a specific parent in favor of a less attractive for the greater good of the overlay. A child that is being weaned operates normally in subsequent epochs with the exception that it will attempt a transformation even if the target is not strictly better than its current parent. Note that a wean may or may not succeed (an appropriate alternate parent may not exist). The wean operation expires after a configurable number of epochs.

4. Multi-Metric Transformations

AMMO makes use of the overlay maintenance mechanism provided by TreeMaint to build scalable and adaptive overlays that are optimized for a plurality of performance characteristics, such as bandwidth, delay, cost, and packet loss rate. To this end, AMMO nodes keep state regarding their current performance measures in the overlay. In general, we assume that potential overlay edges have multiple independent weights representing these dynamically changing metrics.

An AMMO node, Q, maintains $U_{Q,i}$ which is its current estimate for the cumulative metric on the root-to-node path, for each metric i. For example, for a delay metric, this would be the sum of delays at each overlay hop from the root to Q. In addition, the node maintains L_{Q_i} which is the estimate of the maximum node-to-leaf cumulative metric for the subtree rooted at this node. AMMO augments RanSub Collect and Distribute messages to contain fields necessary to maintain this node state. Table 1 shows a summary of the additional fields used to construct AMMO trees.

4.1. Types of Metrics

There are many types of performance metrics that an application may consider. Our approach identifies two measurement properties. First, we assume that for each metric, a measure ω is "better" than a measure ϕ if $\omega < \phi$. Second, each measure has an associated $+'$ operator that allows measures of two edges to be combined into one cumulative measure. We now briefly outline the metrics that AMMO currently supports:

1. Delay captures both queuing and propagation delay incurred from sending a packet across a link. It can be estimated through "pings". The $+'$ is defined as $+$ since delays through an overlay are the sum individual overlay hop delays.
2. The bandwidth complement is the difference between some pre-defined bandwidth expectation and bandwidth through an overlay edge. We define the $\omega +' \phi$ operation as $max(\omega, \phi)$ since the bandwidth achievable through two contiguous overlay edges is the minimum of the bandwidth achievable through each constituent overlay edge (and hence the maximum of the bandwidth complements).
3. Packet loss rate refers to the fraction of packets lost when sending data over an overlay edge. Packet loss rate can be determined via probing. Its $\omega +' \phi$ operation is defined as $1 - (1 - \omega)(1 - \phi)$ since the probability of a packet loss across two contiguous edges is the complement of the product of the delivery probabilities of each edge.
4. Cost is a somewhat more ambiguous metric that may be assigned arbitrarily, for instance, representing the performance of a link, the actual price paid to an ISP or the relative desire to use one link over another. In our implementation, the cost of an overlay edge equals the sum of the costs of the underlying IP edges. Note that, in general, cost need not be tied to the underlying topology and could be specified via administrative policies. Its $+'$ operator is defined as $+$.

4.2. Application Goals

An application using AMMO specifies its performance criteria using a metric function, performance constraints, and constraint priorities. The metric function $F()$ assigns a metric weight to an edge as described in Equation 1. In this equation, $M_i(e)$ refers to the measure of edge e along metric i and A_i encodes an application's relative weight for each metric. This metric function is used to guide overlay transformations once the overlay has fallen within application constraints.

Root-to-leaf metric constraints indicate to AMMO the minimal requirements of the overlay. Exceeding these constraints results in an unusable overlay. These constraints are prioritized so that AMMO transforms the overlay until it meets each constraint, starting with the most important constraint. The constraints are denoted $C_1 \dots C_k$ such that if $m < n$, C_m has a higher priority than C_n.

Collect		Distribute	
$L_{Q,1} \ldots L_{Q,k}$	For each metric, estimate of worst descendant's node-to-leaf measure	$U_{Q,1} \ldots U_{Q,k}$	For each metric, estimate of root's measure to this node via overlay
$G_{Q,1} \ldots G_{Q,k}$	For each metric, estimate of gain from moving to best alternate parent	$L_{root,1} \ldots L_{root,k}$	For each metric, root's current node-to-leaf estimate of worst node

Table 1. Fields augmented to RanSub's Collect and Distribute messages by a sending node, Q.

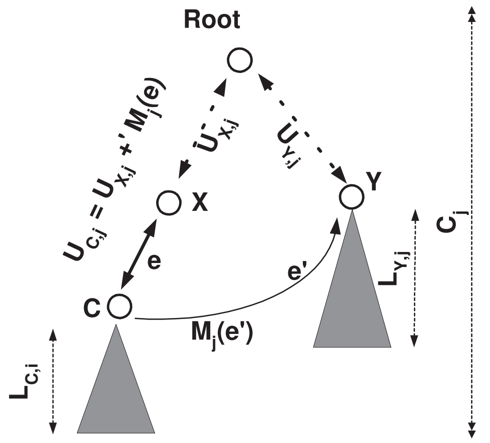

Figure 1. Sample network illustrating a possible transformation.

4.3. Overlay Transformations

Figure 1 describes the necessary state maintained by nodes performing transformations. There are three types of AMMO overlay transformations, CONSTRAINED, TREE-CONSTRAINED, and UNCONSTRAINED, each of which have two different flavors, RESTRICTED and UNRESTRICTED. In this example, a node C is considering a transformation from its current parent X to a new parent Y. The application has specified its performance metric constraints, summarized by the notation C_j (for each metric j). Node C maintains an estimate of the greatest (worst) node C-to-leaf cumulative metric measure which it stores in $L_{C,j}$. It also calculates an estimate of the cumulative metric measure from the root to it and stores that in $U_{C,j}$. These values are used when determining whether to rotate underneath a new parent.

The CONSTRAINED (CON) transformation is effected when a node C's $U_{C,j}$ is greater than a constraint C_j for any metric j. In this case, C has violated an application-specified constraint. It attempts to transform the overlay in order to satisfy the constraint. C determines which node (if any) can help reduce its measure from root for the violated performance metric. Hence, node C would make the move under Y if $U_{Y,j} +' M_j(e') < U_{C,j}$. If multiple metrics are violated, C respects the relative priority specified by the application.

The UNCONSTRAINED (UNC) transformation is designed to reduce the overall tree metric function. This transformation searches for nodes that have a more attractive overlay edge e' than the edge to its current parent e in terms of the metric function. That is, $F(e') < F(e)$. In addition, for each metric j, we ensure that $U_{Y,j} +' M_j(e') +' L_{C,j} < C_j$, meaning that making the transformation will not violate any metric's constraint.

Recall that AMMO delivers current estimates of the root $L_{root,j}$ values via the Distribute message. This allows nodes to determine which constraints are in violation by any node in the tree. If a metric constraint is violated elsewhere, C attempts to make a TREE-CONSTRAINED (TC) transformation to reduce the metric in hopes that the violated node can come within bounds. The nodes are still free to perform UNCONSTRAINED transformations since they, themselves, are within bounds. However, any such transformation must not increase the violated constraint metrics. We offer the details on effects of simultaneous transformations, along with the description of RESTRICTED and UNRESTRICTED transformations, in section 4.4.

In addition to these transformations, AMMO can direct TreeMaint to perform a wean of a child when it has reached its maximum degree. A parent determines the wean target based on the child estimated to lose the least in terms of the best alternate parent. This information is passed by a node X to its parent P in the $G_{X,1} \ldots G_{X,k}$ array in the Collect message as shown in Table 1.

4.4. The Effect of Simultaneous Transformations

Simultaneous transformations may temporarily violate one or more of the overlay's constraints. Consider the following example where an overlay has constrained delay and the metric function considers only cost. A node, A, performs a transformation to a new parent that improves cost but somewhat increases delay. This increased delay does not violate the metric constraints of any of A's descendants. However, a node B may simultaneously (in the same epoch) perform a transformation to node C, node A's descendant, increasing the delay to B and B's descendants, but still within bounds based on C's currently stale $L_{C,delay}$ value.

The cumulative effect of simultaneous delay increases to A and to B may result in delay constraint violations for B or some subset of its descendants. While such temporary violations are acceptable given that we do not provide hard performance guarantees, we mitigate this effect by probabilistically limiting the number of simultaneous *UNRESTRICTED* transformations in any one epoch. Transformations that are *RESTRICTED* have the additional limitation that any move to a new parent Y must not increase Y's $L_{Y,j}$ value for any metric j. Note that simultaneous RESTRICTED transformations will not violate any node Z's $L_{Z,j}$ values. In AMMO, each node performs UNRESTRICTED transformations with probability $\frac{1}{\lg n}$ during each epoch. The rationale is that $\lg n$ is roughly the height of the tree and we wish to avoid multiple transformations within the same path.

5. Evaluating AMMO

We have implemented and evaluated AMMO in the *ns* network simulator [13] and within MACEDON [17] to create TCP/IP code that functions in the ModelNet [21] emulation environment. While we only show ModelNet results, our ns results are qualitatively similar. For most experiments, we use 20,000-node INET generated topologies [2]. We randomly assign 1,000 clients to one-degree stub nodes in the topology and randomly choose a node to root the AMMO tree. We typically set link bandwidths to 100-155 Mbps for transit-transit links, 10-100 Mbps for transit-stub links and 1-10 Mbps for stub-stub and client-stub links. We set delays among nodes based on the relative placement in the plane as determined by INET. To illustrate the effectiveness of AMMO, we have run experiments with constraints and metric functions corresponding to Overcast (bandwidth metric function) and a delay-constrained, cost-optimized flavor which we refer to as $AMMO_{DC}$ In the case of Overcast, there is no metric constraint and the metric function depends only on the bandwidth metric:

$$F(e) = M_{bandwidth}(e) \tag{2}$$

To estimate bandwidth, Overcast uses a download times of 140 Kb of data. For $AMMO_{DC}$, delay is constrained and F is given by:

$$F(e) = M_{cost}(e) \tag{3}$$

We set the cost of physical links to be uniformly distributed between 20-40 for transit-transit links, 10-20 for transit-stub links, and 1-5 for stub-stub and client-stub links. We then set the cost of an overlay edge to be the sum of the costs of its constituents physical links in the underlying IP network. Note that a contribution of our work is not an accurate model for assigning link cost. Rather, we wish to demonstrate that AMMO can optimize for overlay cost independent of the technique used to assign it.

5.1. Overlay Convergence Performance Results

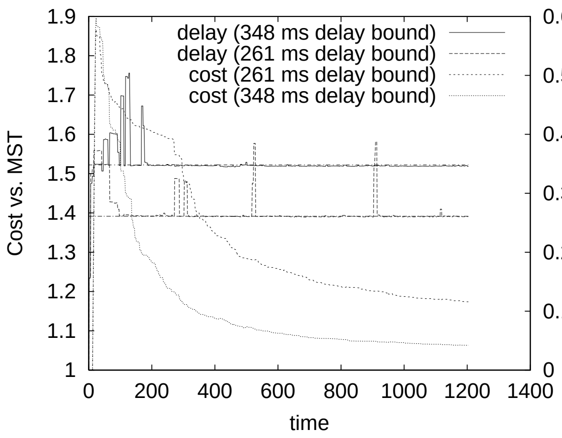

Figure 2. $AMMO_{DC}$ cost and delay convergence as a function of time for two different delay constraints.

Figure 2 shows the maximum root-to-leaf delay and cumulative cost for two $AMMO_{DC}$ overlays for a representative topology Nodes join the overlay during the first 20 seconds. In effect, at the beginning of the experiment we pessimistically create a random overlay. We show cost of the overlay relative to the minimum cost spanning tree (calculated offline) on the left-hand y-axis as a function of time progressing on the x-axis. A "Cost vs. MST" of 1.0 corresponds to the degree-unbounded optimal. We use a probe set of 15 and a maximum degree of 40. For this example, a shortest path tree (constructed from the overlay participants) achieves a 174 ms worst-case root-to-leaf delay. We observe the behavior of the system for two different delay constraints, 261 ms (corresponding to $1.5SPT$) and 348 ms ($2.0SPT$). The right-hand y-axis shows the achieved delay as a function of time. AMMO quickly converges to the specified delay constraint in both cases, despite starting with random overlay connectivity. Once the delay constraint is satisfied, cost steadily decreases to within 20% of the MST optimum.

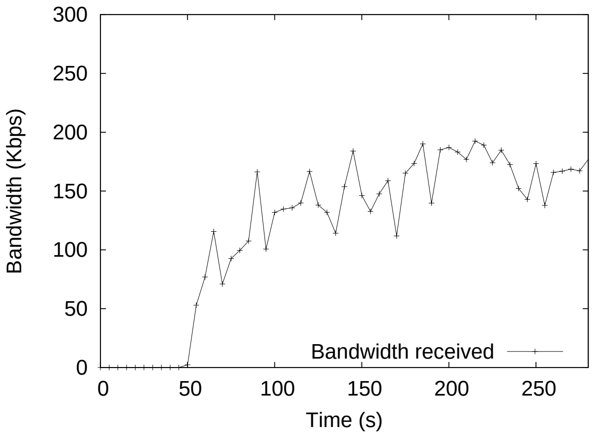

Figure 3. Overcast achieved bandwidth over time.

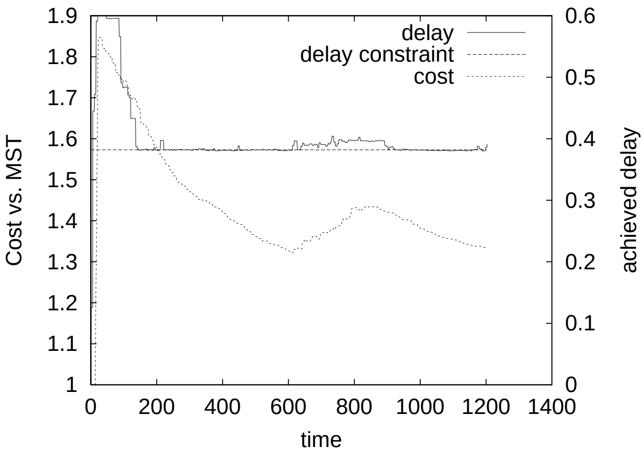

Figure 4. Adaptivity of AMMO overlays in response to pronounced increase in network delay.

Next, we use our Overcast implementation to stream data over the resulting tree. We decreased topology bandwidth to better illustrate Overcast's convergence; we use 2000-4000 Kbps for transit-transit, 1000-2000 Kbps for transit-stub, 500-1000 Kbps for stub-stub, and 300-600 Kbps for client-stub links. Our AMMO Overcast implementation uses a modified version of RanSub to only deliver random subsets containing only siblings and grandparents (as employed by the original Overcast algorithm). We plot the average achieved bandwidth over time in figure 3. Bandwidth increases as nodes continuously move to find more bandwidth. Eventually, the average stabilizes at 180 Kbps.

5.2. Adaptivity

An important aspect of AMMO is its ability to dynamically react to changing network conditions. Here, we subject a steady-state $AMMO_{DC}$ overlay to widespread and sustained change in network delay, where 13% of network links are increase their delays by 0 to 25% of the original link delay. This occurs every 25 seconds starting at time 600 for 200 seconds. The idea behind this experiment is to evaluate the overlay as the network degrades under conditions much worse than those typically found on the Internet.

Figure 4 shows the overlay quickly converging to the delay constraint (382 ms). Cost steadily decreases to within 30% of the optimal MST value when the network perturbation begins. At that point, AMMO's target constraints are violated and metric function operations subside in an attempt to restore the tree to within the delay constraint. AMMO takes an additional 100 seconds to meet the constraint after the network degradation subsides. Note, however, that the increased link delays are maintained at their elevated levels. Once it has again met the delay constraint, AMMO recovers the cost performance in 300 seconds.

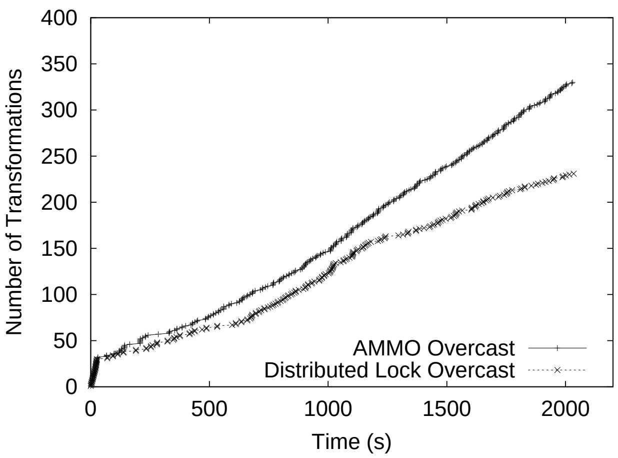

Figure 5. Number of transformations effected over time for AMMO-based Overcast and distributed lock-based Overcast.

To further illustrate AMMO's adaptivity, we compare AMMO Overcast to a similar implementation created with the use of small-scale distributed locks. In this implementation, a node simply locks its parent and siblings prior to requesting probe downloads. Once the probes are complete, the node decides whether a move should be made, performs the transformation, and releases any acquired locks (by sending locked nodes an unlock message). While the use of distributed locking can lead to cumbersome failure detection, we also note that it decreases adaptivity even without such failures. Figure 5 shows the number of transformations performed over time for both Overcast implementations. AMMO Overcast outperforms the lock-based Overcast by as much as 50%, hence enabling greater concurrency and improved adaptivity.

5.3. Dealing With Multiple Metrics

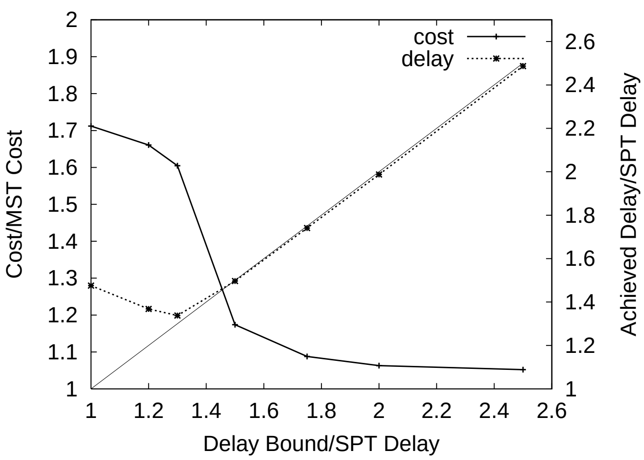

Figure 6. Cost and delay under varying delay bounds.

AMMO allows applications to effect a trade-off among the performance metrics they consider. In $AMMO_{DC}$, for example, a particular delay constraint will correspond to a particular achievable cost. By relaxing or tightening the delay constraint, applications can control incurred cost as shown in Figure 6. The x-axis varies the application-specified constraint as a multiple of the optimal SPT delay. The left-hand y-axis is the cost of the overlay relative to a minimum cost spanning tree while the right hand y-axis depicts the achieved delay as a multiple of the SPT delay. In this example, we use a probe set size of 15 and, more importantly, a degree constraint of 40. The degree constraint dictates how close AMMO is to the optimal (degree-unbounded) SPT delay.

We achieve the delay as long as it is at least 30% larger than the delay of the SPT. Though excluded for brevity, our experiments show that increasing the degree allows AMMO to achieve smaller delays while increasing the degree pushes this point upward. Overlay cost comes within 5% of the MST cost when the delay constraint is relaxed sufficiently. The knee of the curve for cost comes at approximately 70% of the optimal SPT. Relaxing the delay beyond this point does not result in significantly lower steady-state cost (though it could take longer to achieve such cost).

5.4. Effects of Probe Set Size

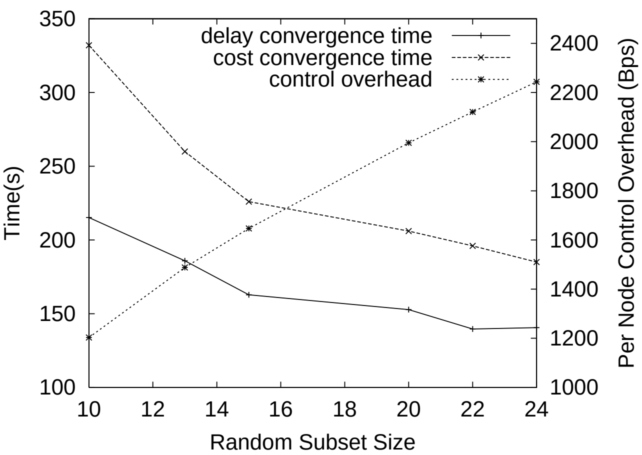

Figure 7. Cost and delay convergence times with resulting per-node probing overhead for varying probe set sizes.

Probe set size refers to how many nodes are included in RanSub subsets. Figure 7 plots the cost and delay convergence time as well as probing overhead for various probe set sizes. We define convergence time to be the time it takes to come within 50% of the optimal MST cost and the time it takes to fall within the delay constraint. We use $AMMO_{DC}$ with a delay constraint of 382 ms ($2.2SPT$) and a degree constraint of 10. Convergence time decreases as probe set size increases, eventually leveling off at around 22. Note that cost convergence is necessarily slower than delay convergence since the overlay must first meet the delay constraint before it can begin making transformations to reduce the metric function (cost in this example). Figure 7 also shows that per-node probing overhead is small, rising to less than 2300 Bps/node for a probe set size of 24. As expected this control overhead grows linearly with the probe set size. With a probe set size of 10, it takes approximately 200 seconds to bring 95% of participants within the delay bound (while details are omitted for brevity, this number is considerably smaller for smaller topologies).

6. Conclusions

This paper presents the design, implementation, and evaluation of an adaptive and scalable algorithm to build multi-metric overlay networks. This problem is NP-complete given centralized information and static network conditions. Thus, our challenge is to develop a distributed algorithm that approximates the global optimum under a variety of dynamic network conditions. In this context, we make a number of contributions. First, we describe a framework where an application can specify performance constraints and a metric function indicating the relative priority of performance metrics. We show how to use RanSub to build and maintain loop-free trees under high levels of concurrency.

A detailed system evaluation shows that AMMO can converge quickly to a close to optimal solution and that it can quickly react to changing network conditions. We provide evaluations of a SARO variant (constrained delay and cost metric function) and AMMO Overcast (bandwidth metric function). We further show how AMMO can strike a balance between conflicting performance metrics yielding an overlay that is highly tuned to application needs.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable Application Layer Multicast. In *Proceedings of ACM SIGCOMM*, August 2002.

[2] H. Chang, R. Govindan, S. Jamin, S. Shenker, and W. Willinger. Towards Capturing Representative AS-Level Internet Topologies. In *Proceedings of ACM SIGMETRICS*, June 2002.

[3] C. Chekuri, S. Khanna, and J. Naor. A deterministic algorithm for the cost-distance problem. In *Symposium on Discrete Algorithms*, pages 232–233, 2001.

[4] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area Cooperative Storage with CFS. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

[5] Y. hua Chu, S. Rao, and H. Zhang. A Case For End System Multicast. In *Proceedings of the ACM Sigmetrics 2000 International Conference on Measurement and Modeling of Computer Systems*, June 2000.

[6] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang. Enabling Conferencing Applications on the Internet using an Overlay Multicast Architecture. In *Proceedings of ACM SIGCOMM*, August 2001.

[7] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. James W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of Operating Systems Design and Implementation (OSDI)*, October 2000.

[8] V. Kompella, J. Pasquale, and G. Polyzos. Two distributed algorithms for multicasting multimedia information. In *Proc. 2nd Intl. Conf. on Computer Communications and Networks (ICCCN)*, 1993.

[9] V. P. Kompella, J. C. Pasquale, and G. C. Polyzos. Multicast routing for multimedia communication. *IEEE/ACM Transactions on Networking*, 1(3):286–292, 1993.

[10] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. Vahdat. Using Random Subsets to Build Scalable Network Services. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*, March 2003.

[11] D. Kostić, A. Rodriguez, J. Albrecht, and A. Vahdat. Bullet: High Bandwidth Data Dissemination Using an Overlay Mesh. In *Proceedings of the 19th ACM Symposium on Operating System Principles*, October 2003.

[12] F. Kuipers and P. V. Mieghem. Mamcra, a constrained-based multicast routing algorithm. *Computer Communications*, 25(8):801–810, 2002.

[13] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[14] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Application-level Multicast using Content-Addressable Networks. In *Third International Workshop on Networked Group Communication*, November 2001.

[15] S. Ratnasamy, P. F. M. Handley, R. Karp, and S. Shenker. A Content Addressable Network. In *Proceedings of SIGCOMM 2001*, August 2001.

[16] A. Rodriguez. *Building Scalable and Adaptive Network Services*. PhD Dissertation, Duke University, December 2003.

[17] A. Rodriguez, C. Killian, D. Kostić, S. Bhat, and A. Vahdat. MACEDON: Methodology for Automatically Creating, Evaluating, and Designing Overlay Networks. In *USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI)*, March 2004.

[18] A. Rowstron and P. Druschel. Storage Management and Caching in PAST, a Large-Scale, Persistent Peer-to-Peer Storage Utility. In *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP'01)*, October 2001.

[19] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-scale Event Notification Infrastructure. In *Third International Workshop on Networked Group Communication*, November 2001.

[20] H. Salama, Y. Viniotis, and D. Reeves. An Efficient Delay Constrained Minimum Spanning Tree Heuristic. In *Proceedings of the Fifth International Conference on Computer Communications and Networks*, 1996.

[21] A. Vahdat, K. Yocum, K. Walsh, P. Mahadevan, D. Kostić, J. Chase, and D. Becker. Scalability and Accuracy in a Large-Scale Network Emulator. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI)*, December 2002.

[22] Z. Wang and J. Crowcroft. Quality-of-Service Routing for Supporting Multimedia Applications. *IEEE Journal of Selected Areas in Communications*, 14(7):1228–1234, 1996.

[23] B. Y. Zhao, J. D. Kubiatowicz, and A. D. Joseph. Tapestry: An Infrastructure for Fault-tolerant Wide-area Location and Routing. Technical Report UCB/CSD-01-1141, UC Berkeley, April 2001.

[24] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. Kubiatowicz. Bayeux: An Architecture for Scalable and Fault-tolerant Wide-Area Data Dissemination. In *Proceedings of the Eleventh International Workshop on Network and Operating System Support for Digital Audio and Video*, 2001.