Service Adaptive Multicast for Media Distribution Networks

Sujata Banerjee, Zhichen Xu, Sung-Ju Lee, and Chunqiang Tang*
Internet Systems & Storage Lab
Hewlett-Packard Laboratories
Palo Alto, CA 94304
{sujata,zhichen,sjlee}@hpl.hp.com; sarmor@cs.rochester.edu

Abstract

Multicast technology is highly efficient for the large scale multimedia content service delivery. Its efficiency is maximized when all the service recipients have identical needs. In reality however, the end users may have a heterogeneous set of requirements for different service levels as well as different service components, depending on their system and network capabilities. We propose the notion of Service Adaptive Multicast (SAM) that balances the tradeoffs between providing individualized service to each client and maintaining an efficient overlay multicast tree structure. The novel aspects of our approach are (a) the ability to augment and transform existing paths into service paths with the desired attributes; and (b) integration of two tree maintenance processes: a receiver-initiated just-in-time adaptation of the multicast service tree driven by application/user perceived QoS, and a demand-driven tree maintenance process geared towards long-term tree quality. We demonstrate the performance of our approach using simulations of large client population.

1. Introduction

Widespread use of electronic multimedia content over the Internet is rapidly increasing. Along with this trend, it is also apparent that different end users require the content delivered to them in different forms, wrapped in different services, depending on their personal preferences, end-system capabilities and network connectivity [2]. For example, one user requires an encrypted version of the content, while another user wants the audio portion of the content in a different language, and a third user wants an encrypted and transcoded-down version of a video to view on a mobile handheld device. While these kinds of services are offered today by only a handful of service providers, we anticipate that the number of service providers will grow exponentially as more individuals and small businesses offer specialized services for sale using a peer-to-peer infrastructure. For real-time multimedia services such as video streaming, providing acceptable end-to-end quality of service (QoS) is imperative. While it is possible to serve diverse and geographically separated users by creating a special service for each user group, this approach is highly inefficient from the standpoint of computing, storage, and networking resources. Our goal is to serve diverse user groups with a single efficient overlay service multicast that is capable of providing individualized service to users with the necessary QoS and maintaining high levels of efficient resource utilization.

Scalable and efficient multicast technology is essential to enable the above goal. Multicasting provides significant bandwidth savings and is particularly crucial for the dissemination of live as well as stored high fidelity multimedia content because of the sheer size of the content, the relatively long duration of the session, and the correspondingly high bandwidth requirements. Due to the lack of widespread deployment of network level IP multicast, overlay multicast protocols are being researched extensively. However, even if IP multicast were available everywhere, the above stated goals of service delivery could most likely only be achieved using the overlay technology.

We propose an efficient application layer Service Adaptive Multicast (SAM) infrastructure for real-time multimedia applications. Our approach rests on providing a global view of the system stored in a distributed hash table (DHT), which is scalable, fault-tolerant, and administration-free. The global view is generated from landmark clustering [21]. Combining the landmark information with a small number of round-trip time (RTT) measurements to locate physically close-by neighbors, our approach provides very fast, high quality tree construction and adaptation. Starting from a service definition, we build end-to-end service paths as defined in [12, 32]. As required, new service paths are built

*Chunqiang Tang is with Department of Computer Science, University of Rochester, Rochester, NY 14627.

from existing "close-by" service paths, thereby creating an efficient multicast service tree. Our goal is to create highly scalable mechanisms that handle scenarios with very large number of service providers as well as large numbers of service consumers. There are several key differences between our approach and that of prior work.

- Most existing overlay multicast schemes consider only a single homogeneous service — typically only packet delivery. We provide a new framework that considers the more realistic scenarios where different users have different service requirements when accessing the same content.
- None of the existing schemes address the problem of application quality disruption during the tree reconfiguration. Our goal is to develop mechanisms that transparently reconfigure the overlay tree in very short timescales such that the user's perceptual service quality does not suffer during the reconfiguration process. Our approach utilizes the landmark information stored in DHT and configures the tree quickly using concurrent network measurements. The aim is to perform tree reconfigurations under a second while producing a tree that is more efficient than existing algorithms and reasonably close to the optimal bandwidth efficiency.
- Unlike other schemes that advocate only periodic or event triggered (e.g., RTT increase) tree reconfigurations, our approach integrates two types of tree maintenance processes: a *just-in-time* reconfigurations driven by application/user perceived QoS, and a less frequent tree maintenance driven by network state change observed from a global information table atop a DHT. Unless the end application perceives a performance degradation, or a service path that can deliver the desired service has later become available, there is no need to reconfigure the tree, even when network metrics such as RTT has changed.

We focus on the construction and maintenance of service multicast trees as application driven just-in-time tree adaptation is introduced in [33]. To make the paper self-contained, we describe the essential components of our previous work when necessary. The paper is organized as follows. We define service paths and service trees and then describe our tree construction and reconfiguration techniques in Section 2. We next provide numerical results using simulations of large client population. Section 4 discusses the related work. The paper concludes with a brief discussion of open problems, ongoing work, and future directions.

2. Service Adaptive Multicast Infrastructure

Constructing an efficient multicast tree for rich media distribution is complicated by the heterogeneity of user needs and available resources. An important challenge is to deliver "personalized" end-to-end service that meets the individual needs while keeping the multicast tree structure as efficient as possible. To handle the above conflicting goals, we take a "user-centric" approach where each end user explicitly specifies the desired service and the QoS requirements. The tree construction algorithm attempts to satisfy the user's need by reusing or augmenting existing paths. To ensure that the resulting multicast service tree efficiently utilizes network resource, our algorithm relies on global information maintained in a DHT, and follows the three intuitive heuristics.

- Service paths should be reused to the extent possible.
- New service paths should be created from existing service paths using appropriate transformations to the extent possible.
- New service components should be placed as near as possible to the nodes requiring the service.

To minimize unnecessary disruptions to the end users, our tree maintenance process is receiver-initiated and just-in-time when the service quality no longer satisfies a user's need [33]. Furthermore, we take advantage of the abundance of computing resources (e.g., those that can be provided by the computing grid, perhaps using resources inside Internet data centers across the Internet) to insert service components appropriately in existing service paths. In the remainder of this section, we describe the basic concepts of a multicast service tree and the basic notations, and then illustrate multicast service tree construction and maintenance algorithms.

2.1. Basic Concepts and Notations

For simplicity, we assume that there is only a single content source and use O to denote the output from this single source. In practice, there could be multiple data streams from different sources that are merged to create composite media content.

- **Service**: A service is modeled as a function that operates on an input and produces an output. The letters f, g, and h are used to denote services. In the case of media content, examples of services include encryption, image repair and analysis, error correction, transcoding and so forth.[1] It should be noted that some services are reversible, i.e., the effect of the service on a given input can be undone (e.g., encryption). Other services such as transcoding are irreversible. For a reversible service f, we use f^{-1} to denote the service that can "undo" the effect of f.

[1] For simplicity, we omit other parameters of services. For example, for a transcoding service, an additional parameter could be the bit rate of the transcoded stream.

- **Service path expression**: Services are composable. For example, $f(g(O))$ denotes that service g is first applied to the original input, and the output of which is fed into the service f. We call formulas such as $f(g(O))$ that specify a list of composed services, a *service path expression*. The path between the origin source to the destination client node, with intermediate nodes that provide various services is termed a *service path* that delivers the composite service specified by the service path expression. While all nodes on a service path participate in the routing process, not all nodes on a service path provide a service. In service paths, the order in which the services are applied is significant. For example, a transcoding operation typically needs to be executed before an encryption operation. In this paper, we use the terms *service path expression* and *service path* interchangeably.
- **QoS-qualified service path expression**: We use service path expression to denote the services requested by an end-user. A service path expression can be "qualified" by QoS requirements to indicate user QoS requirements. Formally, a QoS-qualified service path expression is defined by a pair [service path expression: QoS requirements]. QoS requirements are represented as logical conjunctions of a list of atomic requirements such as (*delay*<100ms), (*bandwidth*>100kb/s), etc.
- **Service requirements**: A user request of encrypted and transcoded media content with an end-to-end delay requirement under 100ms can be represented as $(encryption(transcoding(O)) : delay < 100\text{ms})$. Again, the parameters of encryption and transcoding services are omitted for simplicity. The user can specify service priority by placing more important services earlier in the logical disjunction of multiple QoS-qualified service path specifications.

2.2. Multicast Service Tree Construction

Our tree construction algorithm relies on global state maintained in a DHT that maps "keys" onto "values." The DHT is implemented on infrastructure nodes that have good availability and network connectivity.

We represent the position of a node in the physical network using a *landmark vector* that is produced by measuring round-trip time against a set of well known landmark nodes [21]. The landmark vectors of the nodes define a coordinate space with distances among the landmark vectors reflecting the distances among the corresponding nodes in the physical network.

The global information of the nodes is stored on the DHT using the landmark vectors of the node as DHT keys. As a result, information about nodes that are physically near each other are stored close to each other on the DHT. A typical global table has the schema shown in Table 1.[2]

Table 1. Schema of global table.

Items	Description
`nodeID`	Identifier of the tree node.
`landmark_vector`	It represents the node's position in the physical network.
`node_metric`	Capacity, load, and the services the node provides (e.g., storage, computing, service capability, etc.).
`path_metric`	Characteristics of the path from the root to the node. Information may include delay to the root, bottleneck bandwidth, service components, nodeIDs of the nodes in the path to root, etc.

When new node n wants to join the multicast tree, it computes its own landmark vector and carries out the following steps.

(i) Node n submits its own landmark vector and its service requirements to the DHT infrastructure. The DHT infrastructure matches the service path requirements with the stored information to compute a set of candidates close to node n with which the service requirements can be satisfied directly or a new service path that meets the QoS requirements can be constructed. When no existing service path matches the requirement such as when the node requests the service $f(g(O))$ and the only service available is $g(O)$, the DHT sends to n a list of nodes that provide the service g and a list of nodes that provide the service f. In certain cases, some current services need to be undone to provide the desired service. For instance, if the service path $h(O)$ is available, then one possibility is to construct the required service by first reversing the effect of $h(.)$ and constructing the service path $f(g(h^{-1}(h(O))))$. A corresponding example could be when the new node needs unencrypted media stream but only the encrypted stream is available. The computation of the candidate sets is out of the scope of this paper.

(ii) Upon receiving the candidates from the DHT, the new node n performs additional measurements by estimating delay and bandwidth between n and the candidate nodes. In the case where a new service path needs to be constructed, node n instructs some candidate nodes to perform measurements among themselves to obtain delay and bandwidth information between some of the nodes.

(iii) Node n carries out a series of actions to either reuse an existing service path by attaching to a node as its child or constructs a new service path.

In the first step of the algorithm, when providing the new node n with the candidates, the DHT accounts for both the

[2] More details about landmark clustering and DHT techniques will be provided in Section 2.2.1 and Section 2.2.2.

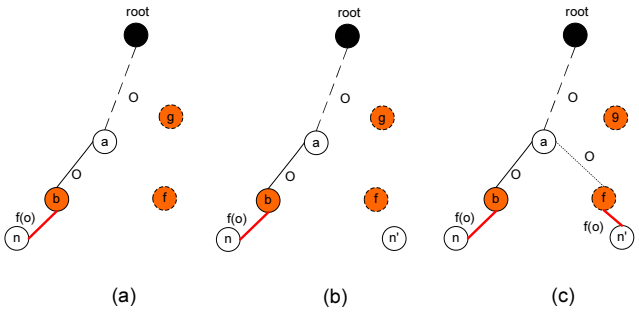


Figure 1. Tree construction.

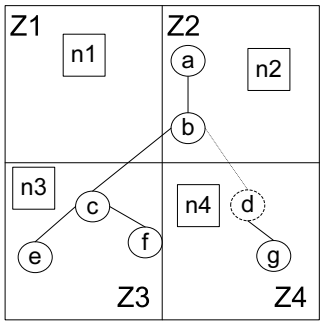


Figure 2. Storing and retrieving global state on a DHT.

user requirements and the tree quality. The DHT finds existing services that are near the new node when possible. However, when exact requested services do not exist, service components that can be used to construct a desired service path are provided close to the new node.

Figure 1 exemplifies our tree-construction algorithm. The new nodes n and n' request for service $f(O)$. In Figure 1 (a), the DHT finds an existing service path $(root, ..., a, b)$ near n and the new node n attaches to b as its child. In Figure 1 (b), when the new node n' wants to join the multicast tree, nodes b and n that offer the service are far away from n'. Consequently, node a that provides the original stream O and node f that is not on the tree but provides the service, are identified. A new service path $(root, ..., a, f, n')$ is constructed as shown in Figure 1 (c).

2.2.1. Landmark Clustering

Landmark clustering is based on the intuition that nodes close to each other are likely to have similar distances to a few selected landmark nodes. Our landmark clustering is based on [21], where a set of well known landmark nodes is first identified. The landmark nodes measure the RTT among themselves and use this information to compute a coordinate in a Cartesian space (i.e., landmark vector) for each of the landmark. These coordinates are then distributed to the clients, which measure RTTs to the landmark nodes and compute their own landmark vector.

Landmark clustering however, is only a coarse-grained approximation. It is not very effective in differentiating nodes within close distance. To remedy this effect, we propose two techniques: (i) combining landmark clustering with RTT measurements, and (ii) hierarchical landmarks .

With hierarchical landmarks, the top-level global landmarks provide a rough estimation of nodes' physical position and lower-level local landmarks further differentiate nodes that are in close distance. We are in the process of evaluating the effectiveness of several hierarchical schemes and plan to incorporate them in our tree construction and maintenance algorithms.

To obtain more accurate information, actual RTT measurements are performed against the set of nodes that is returned through landmark clustering.

2.2.2. Storing and Retrieving Global State on a DHT

DHT-based overlays, represented by Content Addressable Networks (CAN) [24], Chord [30], and Pastry [27], are scalable, fault-tolerant, and administration-free. Their basic functionality is to map "keys" onto "values." Our DHT stores the global state and is based on CAN, which provides a hash table abstraction over a Cartesian space. The Cartesian space is partitioned into zones, with one or more nodes serving as owner(s) of a zone. A key is a point in the space and the owner of the zone that contains the point stores the corresponding value. Since the landmark vectors define a coordinate space, we use the landmark vectors directly as the hash keys.

To take advantage of the physical network topology, we employ landmark clustering when constructing the DHT and storing the information. We build a topologically-aware CAN [25] where each node joins the Cartesian space owning a zone that contains its landmark vector.

When storing information about a node in the multicast service tree, we compute its landmark vector in the same way as we did for the DHT nodes, and use its landmark vector as the key to store the information according to the schema described in Section 2.2. This approach has two advantages: (i) information of a tree node are stored on a DHT node that is close to it with a high probability; and (ii) information of nodes that are near each other are stored close to each other on the DHT. Therefore, to find information about nodes that are close to a particular node, we first route to the zone using the node's landmark vector as the key and then perform a localized flooding.

Figure 2 illustrates how the global state is stored on a DHT based on landmark clustering using a two-dimensional CAN. In the figure, the x and y coordinates of the nodes are

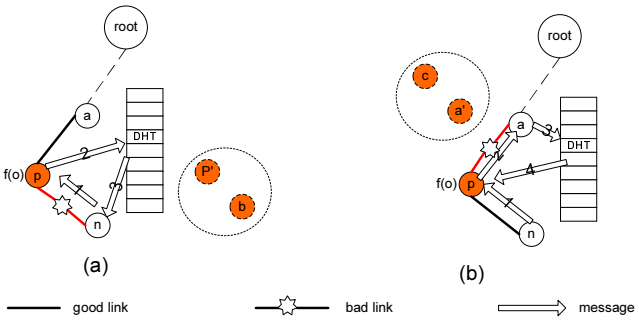


Figure 3. Tree adaptation.

drawn to reflect their landmark vectors. The Cartesian space of CAN is the coordinate space. The coordinate space is partitioned into four zones from $Z1$ to $Z4$ with DHT nodes $n1$ to $n4$ serving as their owners, respectively. Each DHT node owns a CAN zone in which its landmark vector falls into. The information about the tree nodes are stored on the DHT using their landmark vectors as keys. For example, the information of nodes a and b are stored on the DHT node $n2$. Similarly, the information of nodes c, e, and f are stored on the node $n3$.

2.3. Adaptation of Multicast Service Trees

Driven by the inherent dynamics in the underlying infrastructure, we propose two tree adaptation schemes—a *just-in-time* adaptation to address application quality, and a long-term adaptation to address tree quality driven by the network state change observed by the DHT.

2.3.1. Just-in-Time Adaptation

This algorithm, based on [33], is driven by application perceived QoS that is impacted not only by fluctuations in the link quality but also by the availability of the requested service. To provide the end users with desired services with reasonable QoS, the tree continuously adapts to the changing conditions and minimizes any service disruption to the end users. This translates into finding the best location to perform the adaptation and minimizing the latency for each repair. We assume that all tree nodes can monitor the availability of relevant services and the quality of the link, and translate these into user-perceived QoS.

We illustrate our tree adaptation algorithm in Figure 3. When node n perceives that the requested service is not available or QoS degrades over its tolerance threshold, it sends a complaint to its parent p in the tree along with its landmark vector and service requirements. If p is not responsive, n switches to a new parent by performing a new join process as described in Section 2.2. If p responses, we have two cases as shown in Figure 3 (a) and (b).

(i) p **is happy with the services it receives**, which indicates that the bottleneck link lies on the path (p, n). p forwards the complaint initiated by n directly to the DHT, which will provide n with a set of new candidates that are close to n, judging from the landmark vectors. This candidate set includes p' and b. n chooses its new parent, for example p', based on the measured RTTs to candidates and the QoS they provide. n then carries out the switching with the handoff process we describe in Section 2.4.

(ii) p **is also unhappy with its QoS**, which indicates that the bottleneck link exists on the upstream path, e.g., path (a, p) in our example. In this case, p starts its own complaint process by sending a message containing its landmark vector to its parent a. Note that by the time the complaint from n arrives at p, p may already have sent its own complaint to a. In this case, p suppresses n's complaint. These concurrent complaints save significant adaptation time.

In Figure 3 (b), because a is happy with the services it receives, it directs the complaint to the DHT, which will instruct p to switch to a new parent with the candidate set including a' and c. p then measures the RTTs to these nodes and switches to a'. During this process, n waits for the service to become available or the QoS to improve, or an instruction from the DHT to switch to a new parent. If it is still unhappy after a timeout, i.e., there are multiple bottleneck links on its upstream path, it starts the complaint process again.

Our tree-adaptation algorithm minimizes the overall disruption by locating the problematic link and having the node incident to that link to adapt. For instance, when the quality of a link close to the root degrades or a service close to the root becomes unavailable, our local repair algorithm requires only the node incident to that link to attach to a new parent, instead of having every downstream node of that link to find a new parent.

Our tree adaptation algorithm avoids the cyclic path because the DHT keeps the nodeIDs of the nodes from the root to every tree node. If the DHT finds that the ID of a complaining node is in the path from the root to a candidate it identifies, this candidate is not selected as the new parent. Tree adaptation could cause oscillations where a node keeps switching parents back and forth among a set of candidate nodes. To avoid this problem, each node caches the parent nodes of the recent past and does not choose a node in the cache as the new parent.

Our approach typically takes three steps to obtain a set of parent candidates. Using Figure 3 (a), assume that (n, p) and (n, p') distances are 20ms since n is close to p and p', and (p, DHT) and (n, DHT) are 100ms. Considering that routing in the DHT typically doubles the latency of IP routing [34], it takes approximately 320ms to obtain the candidate sets. Assume that we perform three rounds of con-

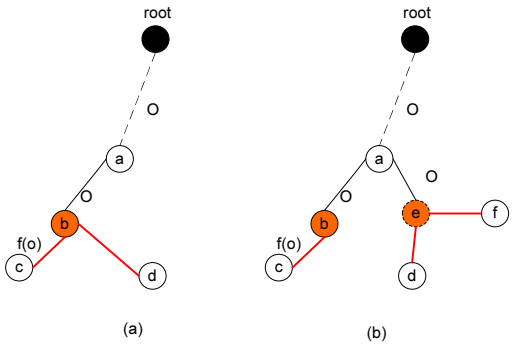


Figure 4. Tree maintenance.

current RTT measurements to all candidates and choose the candidate that has the lowest RTT. This process takes additional 120ms. This leaves us with 560ms to complete the entire switching under one second.

2.3.2. Demand-driven Maintenance

The basic tree construction algorithm is greedy in nature. The order in which the nodes join the tree can affect the tree quality. For example, when a node joins the tree, there may not be a close-by upstream node that provides the desired service but such an upstream node may later become available. Instead of relying on local repair, we leverage the global state kept in the DHT. The basic idea of our tree maintenance algorithm is described below:

- When node n joins the tree, the DHT maintains information including the requirements of this node (e.g., the services it requests and QoS requirements), and its upstream (and possibly downstream) nodes. Node n can also specify conditions (predicates) under which it is interested in getting notified.
- As nodes join and leave the system, the DHT continues to evaluate the predicates of the nodes in the system and notifies the appropriate nodes when a predicate becomes valid (e.g., there is a node near a particular node and is offering a desired service).
- After a node receives a notification from the DHT, it makes a local decision as to whether to reconstruct the tree by switching to a new parent.

We give an example of the tree maintenance algorithm in Figure 4. When node d first joined the system, the only choice it had was to attach to node b. Later on, when f joined the system, it decided that attaching to b or d will not satisfy its delay requirement. A new service path $(root, ..., a, e, f)$ was therefore constructed. The DHT remembers d's service requirements and notifies d the availability of e and f. Node d performs some measurements and attaches to e as its child.

2.4. Smooth Application Handoff

One of our goals is to develop methods to ensure that the application performance suffers minimally and the tree reconfiguration is conducted transparently. Because switching to a new parent may incur delay, it is essential to maintain the performance levels during the parent handoff process. For media applications, this is crucial as the user perceived media quality may suffer significantly when there is a sudden high loss rate or large delay period inflicted by the handoff. To minimize disruption, we use multi-homing at the multicast overlay layer during the handoff period, similar to [28]. The idea is to have a child connected to both the new and old parents, and receive application packets from both until the handoff is complete.

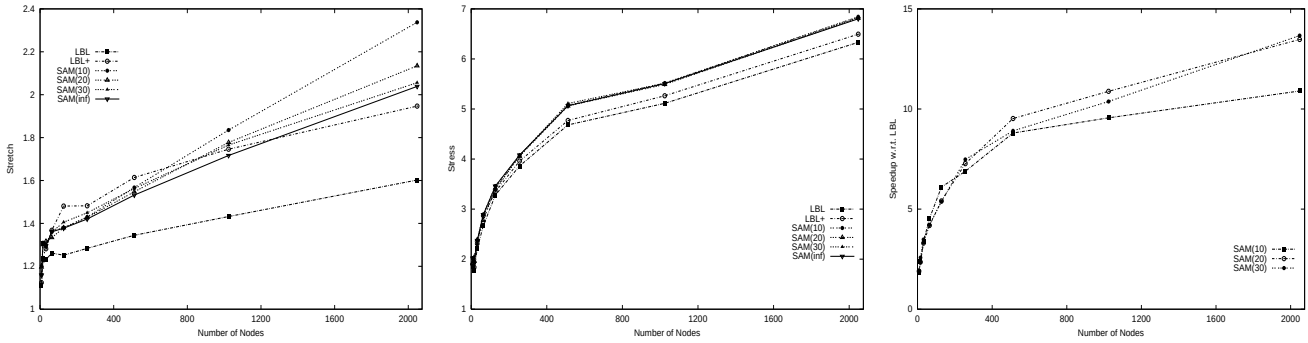
3. Experimental Results

To evaluate our algorithms, we conduct a simulation study on two transit-stub topologies produced by GT-ITM [4], both with approximately 10,000 nodes. The first topology, *small-transit*, has 25 transit domains, five transit nodes per transit domain, four stub domains attached to each transit node, and 20 nodes in each stub domain. The second topology, *large-transit*, has 228 transit domains and two nodes in each stub domain instead. With this second topology, we intend to simulate a network with a large backbone. The link latencies in this topology are set automatically by GT-ITM, whereas the link bandwidth is set manually according to the fan-outs of the nodes that a link connects to. A link connecting two nodes with a higher fan-out has a higher bandwidth than a link connecting two nodes with a lower fan-out.

We compare our algorithm against two other algorithms. The first algorithm is an abstraction of the existing level-by-level (LBL) tree construction schemes such as Host Multicast Tree Protocol (HMTP) [35] and Yoid [11]. LBL traverses down the tree level-by-level from the root to search for the node that is closest to the new node until it reaches the leaf. This algorithm does not account for the QoS requirements of the end users. The second algorithm, LBL+, is an enhancement to LBL that we propose. It performs level-by-level tree traversal, but also accounts for the QoS requirements of the end users. Since there are many LBL-based schemes, we propose LBL+ to maintain compatibility with the prior work. We use "SAM(C)" to denote our approach with RTT measurements to C closest candidates identified by the DHT.

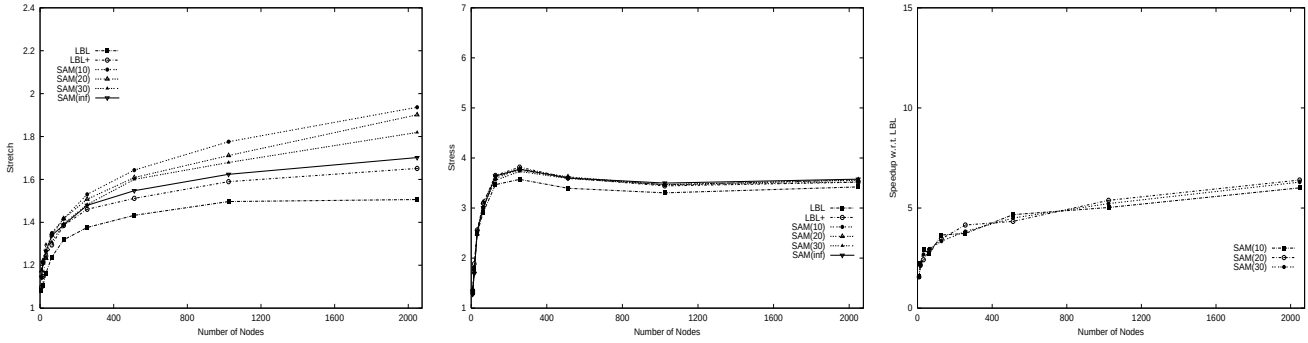
The performance metrics of interest are:

- **Tree quality metrics**: We use *stress* and *stretch* to measure the tree quality. Stretch is defined as the ratio of tree cost (the sum of the delay associated with the tree links)



(a) Stretch. (b) Stress. (c) Construction speedup.

Figure 5. Small-transit topology: stretch, stress, and speedup.



(a) Stretch. (b) Stress. (c) Construction speedup.

Figure 6. Large-transit topology: stretch, stress, and speedup.

to that of a minimal spanning tree. The stress of an overlay multicast tree is the average number of overlay links over a physical link in the underlying topology.

- **Tree construction metrics**: *Speedup* is used to measure the tree construction time of our algorithm against that of LBL.
- **Tree QoS metrics**: *Satisfaction* is the fraction of users that are satisfied with their bandwidth and delay requirements.

We vary the tree size from 8 to 2,048 nodes. For each tree size, we present the average results for 20 runs. Table 2 summarize the parameters, their default values, and the varied range. We constrain the degree of each node on the tree to simulate the fact that the number of flows that a node can serve are constrained by its processing capacity and network connectivity. For simplicity, we assume that there is only a single service, the original media stream. We set the QoS

Table 2. Parameter values.

Parameter	Default value	range
Tree size	-	$8 \sim 2,048$
Landmarks	15	-
RTTs	-	$10 \sim 30$
Maximum node degree	10	-

requirements for the different nodes as follows. For each individual node n, let $d(n)$ denote the shortest path latency from the root of the tree to n and $b(n)$ denote the bottleneck bandwidth from the tree root to n. The user requirement is set to ($delay < 2 \times d(n) \bigwedge bandwidth > 0.25 \times b(n)$).

The preliminary results of our experiments are shown in Figures 5 to 8, where Figures 5 and 6 compare the tree quality of the different algorithms and RTT measurements. They also show the speedup of our tree construction algo-

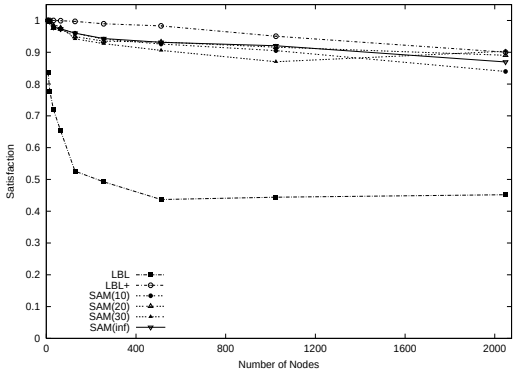


Figure 7. Satisfaction rate in small-transit topology.

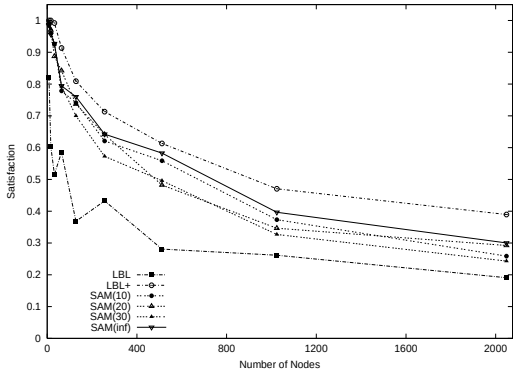


Figure 8. Satisfaction rate in large-transit topology.

rithm when compared with LBL and LBL+. Figures 7 and 8 compare the user-perceived QoS in the two topologies.

In Figures 5 and 6, we can see that our tree construction algorithm is an order of magnitude faster that LBL and LBL+ when the tree degree is constrained to ten. The tree construction times for LBL and LBL+ are similar. In fact, this speedup can be as large as 20 for a tree with 2,048 nodes when the tree degree constraint is set to five.

The figures also show that the stress and stretch of the trees produced by our algorithm is comparable to that of trees produced by LBL+. The quality of the LBL trees is better than others because LBL trees are not restricted by the quality-of-service constraints. The stress and stretch numbers in the small-transit topology are slightly worse than in the large-transit topology for the following reasons: (i) landmark+RTT scheme is less effective in the small-transit topology because of the difficulties in differentiating nodes within a close distance; and (ii) in the small-transit topology, there are more nodes in the same stub domains and shortest paths between nodes are more likely to share common links, therefore the worse stress numbers. Despite the seemingly larger stretch numbers for the small-transit topology, the absolute overhead is small, because the latencies are short and the penalty for not finding the closest node is small.

Figures 7 and 8 show the fraction of users that are satisfied by their needs. The satisfaction rate in the large-transit topology is lower than that in the small-transit topology. This is because when the nodes are sparely populated over a very large area network, the parent of a tree node is less likely to be on the shortest path from the tree root to the node. Note that the absolute number is not important, since it is a consequence of how we select the QoS requirement. What is significant is that SAM achieves better level of QoS and comparable tree quality as LBL at a much faster speed.

In our experiments, we have varied the number of RTT measurements. The number of RTTs does not affect the tree quality and user perceived QoS when it is greater than 20. Moreover, we use a very small number of landmarks to cluster the nodes for a very large transit-stub graph. We expect the hierarchical landmark clustering scheme described in Section 2.2.1 to further improve our tree quality and the user-perceived QoS these trees offer.

4. Related Work

4.1. Overlay Multicast Trees

Several application-level multicast schemes achieve data distribution by *implicitly* building a multicast structure. For instance, Scribe [6] is a multicast infrastructure built on top of Pastry [27]. In Scribe, the multicast tree is formed by the union of the Pastry routes from multicast members to the rendezvous point (RP). Bayeux [37] is an architecture built on top of Tapestry [36] and supports source-specific multicast. The Content-Addressable Network (CAN) framework [24] is extended for multicast in [26]. In this work, the multicast group members establish a mini-CAN and multicast data is distributed by flooding over the mini-CAN, without explicitly building a tree. The Scribe's tree-based approach and CAN's flood-based approach are compared in [7]. Their experiments show that the tree-based multicast consistently outperforms the flooding approach.

The "NICE is the Internet Cooperative Environment" (NICE) protocol [1] builds and maintains hierarchical topology of multicast members. The multicast routes are implicitly defined by the structure of the hierarchy. A protocol that uses Delaunay triangulation as an overlay network topology is proposed in [20]. With the distributed construction of a Delaunay triangulation, multicast paths are embedded

in the overlay without a routing protocol. Overlay Multicast Network Infrastructure (OMNI) [3] proposes a two-tier architecture and builds a multicast tree consisting of multicast service nodes (MSN) which in turn connect to clients. This distributed scheme is adaptive with changes in the client distribution and network conditions.

The following protocols *explicitly* form the multicast tree. Targeting at content distribution applications, overcast [15] builds a single source multicast tree rooted at the source. The optimization goal of its "up/down" protocol is to provide each tree node with a high bandwidth path to the root. Yoid [11] forms a shared multicast spanning tree across the end hosts. Yoid builds a mesh structure among members for routing stability. Similar to Yoid, Host Multicast Tree Protocol (HMTP) [35] builds a shared tree. When a new node joins, it probes the tree at each level, starting from the root, to find the nearest member node as a parent. CoopNet [22] focuses on using *multiple description coding* to handle flash crowd while reducing disruption. They rely on a centralized server for tree construction and maintenance. Application Level Multicast Infrastructure (ALMI) [23] uses a centralized approach to construct shared minimum spanning tree based on network measurements.

Narada [9] and Scattercast [8] build a mesh topology of all multicast members, and then compute a multicast spanning tree for each source. Both protocols periodically refresh the mesh to maintain the multicast topology.

SplitStream [5] addresses the problem of load imbalance of interior and leaf nodes in a multicast tree. It constructs a forest, rather than a single tree. The content is partitioned into multiple stripes using erasure coding or multiple description coding, with each stripe being multicasted on one of the trees. Each participating node serves as an interior node of a tree but as a leaf of some other trees in the forest. Similar to most other DHT-based multicast schemes, the trees in the forest are embedded in the DHT.

ZIGZAG [31] proposes a peer-to-peer multicast for streaming media based on an administrative organization in which peers are organized in a multi-layer hierarchy of clusters. Given the administrative logical organization, the multicast tree is built using three given rules. The tree is periodically reconfigured to balance the load based on the node degree and capacity.

The Scalable Adaptive Randomized Overlay (SARO) protocol [18] has been recently proposed, built on top of a Random Subsets ($RanSub$) utility. The $RanSub$ utility is used to deliver state information about a random subset of global nodes with each node selected in a subset with equal probability.

Our scheme differs from existing approaches in that previous P2P multicast systems embed the multicast trees in the overlay, and therefore are constrained by the logical structure of the P2P networks. In this aspect, SAM is similar to the ZIGZAG approach which decouples the administrative organization and the multicast data delivery paths.

In addition, with the exception of OMNI and ZIGZAG, none of the existing approaches take QoS into account in tree construction and maintenance. Unlike OMNI and ZIGZAG, the tree reconfiguration in our scheme is initiated by the receiver based on the application perceived QoS. The objectives of SARO are similar to that of SAM in terms of adapting quickly to network changes. However, SAM advocates the use of application QoS feedback to trigger tree transformations rather than the use of the periodic random subset distribution approach of SARO.

4.2. Service Paths

Ninja [13] and Composable, Adaptive Network Services (CANS) [12] are typical examples of infrastructures that support heterogeneous user devices and needs. Ninja is a distributed service architecture that builds paths of composed services. Active proxies are located between the service base and the user devices for dynamic service adaptation. CANS is very similar to Ninja, but one of the main differences is that CANS performs resource-aware service adaptation. Service On-Demand Architecture (SODA) [16] shares the same high-level goal of Ninja and CANS, but focuses on service virtualization by executing multiple User-Mode Linux atop a unmodified host OS to achieve fault isolation. Service Overlay Networks (SON) [10] is pieced together via service gateways, the logical connection between which is provided by the underlying network domain with certain QoS (e.g., bandwidth) guarantees. The Internet indirection infrastructure (I^3) [29] introduces a level of indirection to avoid the limitations of the current point-to-point communication model of the Internet. It provides the basic primitives to enable efficient multicast, anycast and service composition.

Service path in overlay networks is considered in Service Proxy Networks (SPY-Net) [32]. SPY-Net builds a highly connected mesh to maintain network resource conditions. It then performs a link-state algorithm on the mesh to construct overlay service paths. SPY-Net requires the proxy to have global network resource availability information. QoS-aware service aggregation [14] composes overlay service paths by mapping user request into resource requirements (e.g., bandwidth, processor, memory, etc.). The path that satisfies these resource requirements is selected and used. QoS-aware Routing in Overlay Networks (QRON) [19] builds QoS-satisfied hierarchical service paths using Dijkstra-like algorithms, with computational capacity and available bandwidth as the path weight.

Although the above schemes consider building service paths to accommodate heterogeneous users, none have ex-

plored multicast. We recently discovered work concurrent with ours that extends SPY-Net to multicast [17]. It proposes two algorithms: (i) shortest service path tree that is basically a union of unicast routes from the source to each multicast member, and (ii) longest match approach that stresses on path sharing. While the concept of multicast service trees is similar to our work, there are several differences. The primary difference is that [17] is geared towards an environment with a small number of service proxies, and uses a link-state like protocol to distribute service routing information between the proxies. This is clearly not scalable to an environment with a large number of service proxies, as we envision. The framework proposed in [17] assumes the existence of service proxies, while our algorithm deploys one if necessary based on available resources. To construct the most efficient and high quality service paths, we recognize that some services are reversible, which provides added flexibility in creating new service paths. In addition, we advocate just-in-time multicast service tree reconfiguration based on receiver perceived media quality as well as periodic longer term tree maintenance operations.

5. Discussion and Conclusion

We provided a new framework that considers scenarios where different users have different service requirements when accessing the same media content. The services can be provided by a small number of well known large service providers with multiple service offerings, and/or by a large number of small single-service providers over a peer-to-peer infrastructure. Unlike the big service providers that may host their services in data centers several network hops away from the end user, the small service providers may host their services closer to the network edge and thus the end user. This proximity to the end user may provide a higher perceived service quality in some cases and bears further investigation. This framework presents several difficult challenges in coordinating end-to-end service paths to deliver composite personalized services to end consumers. Our emphasis is on scalable mechanisms that works well in such large scale environments. While the mechanisms in this paper have been described in the context of rich media streaming applications, they can be applied to other types of applications and services as well.

Towards this end, we have presented the concept of Service Adaptive Multicast (SAM) that provides composite individualized services to consumers while balancing the need for an efficient infrastructure that maximizes resource utilization. Our approach rests on providing a global view of the system stored in a distributed hash table (DHT). The global view is generated from landmark clustering. Combining the landmark information with a small number of RTT measurements to locate physically close-by neighbors, our approach provides very fast, high quality tree construction and adaptation. Performance evaluation using simulations indicate that this approach is promising.

Our future work focuses on improving the accuracy of the landmark clustering scheme. Designing efficient service path computation algorithms is an ongoing research thread as well as a comprehensive evaluation of the framework presented.

6. Acknowledgments

We thank Zhiheng Wang of the University of Michigan for his valuable contributions to this research, while he was an intern at HP Laboratories.

References

[1] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, August 2002.

[2] S. Banerjee, J. Brassil, A. Dalal, S.-J. Lee, E. Perry, P. Sharma, and D. A. Thomas. CDNs for personal broadcasting and individualized reception. In *Proceedings of WCW*, Boulder, CO, August 2002.

[3] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of INFOCOM*, San Francisco, CA, April 2003.

[4] K. Calvert, M. Doar, and E. W. Zegura. Modeling Internet topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[5] M. Castro, P. Druschel, A.-M. Kermarrec, A. Rowstron, and A. Singh. SplitStream: High-bandwidth content distribution in cooperative environments. In *Proceedings of IPTPS*, Berkeley, CA, February 2003.

[6] M. Castro, P. Druschel, A.-M. Kermarrec, and A. I. T. Rowstron. Scribe: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8):1489–1499, October 2002.

[7] M. Castro, M. B. Jones, A.-M. Kermarrec, A. Rowstron, M. Theimer, H. Wang, and A. Wolman. An evaluation of scalable application-level multicast built using peer-to-peer overlays. In *Proceedings of INFOCOM*, San Francisco, CA, April 2003.

[8] Y. Chawathe. Scattercast: An adaptive broadcast distribution framework. *ACM Multimedia Systems Journal*, 2003.

[9] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. *IEEE JSAC*, 20(8):1456–1471, October 2002.

[10] Z. Duan, Z.-L. Zhang, and Y. T. Hou. Service overlay networks: SLAs, QoS and bandwidth provisioning. In *Proceedings of IEEE ICNP*, Paris, France, November 2002.

[11] P. Francis. Yoid: Your Own Internet Distribution, March 2001. http://www.isi.edi/div7/yoid/.

[12] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. CANS: Composable, adaptive network services infrastructure. In *Proceedings of USITS*, San Francisco, CA, March 2001.

[13] S. D. Gribble, M. Welsh, R. von Behren, E. A. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Joseph, R. H. Katz, Z. M. Mao, S. Ross, and B. Zhao. The Ninja architecture for robust Internet-scale systems and services. *Computer Networks*, 35(4):473–497, March 2001.

[14] X. Gu and K. Nahrstedt. A scalable QoS-aware service aggregation model for peer-to-peer computing grids. In *Proceedings of the IEEE HPDC-11*, Edinburgh, Scotland, July 2002.

[15] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. W. O'Toole, Jr. Overcast: Reliable multicasting with an overlay network. In *Proceedings of USENIX OSDI*, San Diego, CA, October 2000.

[16] X. Jiang and D. Xu. SODA: a service-on-demand architecture for application service hosting utility platforms. In *Proceedings of IEEE HPDC-12*, Seattle, WA, June 2003.

[17] J. Jin and K. Nahrstedt. On construction of service multicast trees. In *Proceedings of IEEE ICC*, Anchorage, AK, May 2003.

[18] D. Kostic, A. Rodriguez, J. Albrecht, A. Bhirud, and A. M. Vahdat. Using random subsets to build scalable network services. In *Proceedings of USITS*, Seattle, WA, March 2003.

[19] Z. Li and P. Mohapatra. QRON: QoS-aware routing in overlay networks. *IEEE JSAC*, 2003, to appear.

[20] J. Liebeherr, M. Nahas, and W. Si. Application-layer multicasting with delaunay triangulation overlays. *IEEE JSAC*, 20(8):1472–1488, October 2002.

[21] T. S. E. Ng and H. Zhang. Predicting Internet network distance with coordinates-based approaches. In *Proceedings of IEEE INFOCOM*, New York, NY, June 2002.

[22] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai. Distributing streaming media content using cooperative networking. In *Proceedings of ACM NOSSDAV*, Miami, FL, May 2002.

[23] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An application level multicast infrastructure. In *Proceedings of USITS*, San Francisco, CA, March 2001.

[24] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Shenker. A scalable content addressable network. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[25] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings of IEEE INFOCOM*, New York, NY, June 2002.

[26] S. Ratnaswamy, M. Handley, R. Karp, and S. Shenker. Application-level multicast using content-addressable networks. In *Proceedings of NGC*, London, UK, November 2001.

[27] A. I. T. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer sysstems. In *Proceedings of IFIP/ACM Middleware*, Heidelberg, Germany, November 2001.

[28] S. Roy, B. Shen, V. Sundaram, and R. Kumar. Application level hand-off support for mobile media transcoding sessions. In *Proceedings of ACM NOSSDAV*, Miami, FL, May 2002.

[29] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet indirection infrastructure. In *Proceedings of ACM SIGCOMM 2002*, Pittsburgh, PA, August 2002.

[30] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of ACM SIGCOMM*, San Diego, CA, August 2001.

[31] D. A. Tran, K. A. Hua, and T. Do. ZIGZAG: An efficient peer-to-peer scheme for media streaming. In *Proceedings of IEEE INFOCOM*, San Francisco, CA, April 2003.

[32] D. Xu and K. Nahrstedt. Finding service paths in a media service proxy network. In *Proceedings of SPIE/ACM MMCN*, San Jose, CA, January 2002.

[33] Z. Xu, C. Tang, , S. Banerjee, and S.-J. Lee. RITA: Receiver initiated just-in-time adaptation for rich media distribution. In *Proceedings of ACM NOSSDAV*, Monterey, CA, June 2003.

[34] Z. Xu, C. Tang, and Z. Zhang. Building topology-aware overlays using global soft-state. In *Proceedings of IEEE ICDCS*, Providence, RI, May 2003.

[35] B. Zhang, S. Jamin, and L. Zhang. Host multicast: A framework for delivering multicast to end users. In *Proceedings of IEEE INFOCOM*, New York, NY, June 2002.

[36] B. Y. Zhao, L. Huang, S. C. Rhea, J. Stribling, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A global-scale overlay for rapid service deployment. *IEEE JSAC*, 2003, to appear.

[37] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. H. Katz, and J. D. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In *Proceedings of ACM NOSSDAV*, Port Jefferson, NY, June 2001.