ERAS: Efficient, Robust, Adaptive, and Scalable Service Composition in Overlay Networks

Jingwen Jin Klara Nahrstedt
Dept. of Computer Science
University of Illinois at Urbana-Champaign
{jjin1, klara}@cs.uiuc.edu

***Abstract*— The importance of the component service model has been widely recognized in the Internet community due to its high flexibility in creating customized applications from primitive services in a plug-and-play manner. Deployment of such a service model in the wide area networks spawns a new type of routing problem - *QoS service-added routing*, that is far more complex than the traditional QoS routing problem, especially when multicast comes into play to save resources. We study this special routing problem in overlay networks, and provide an Efficient, Robust, Adaptive, and Scalable (ERAS) service composition framework that (1) optimizes composite services' runtime performance-related aspects (e.g., network bandwidths, path delay, machine resources), which is of great importance to wide-area applications, especially to those that are resource-consuming (e.g., multimedia applications); optimizes global resource usage in one-to-many application scenarios by means of multicast (service multicast and hybrid multicast), to eliminate redundancies in data delivery and service executions; (2) is robust against network failures; (3) adapts to network and membership dynamics; and (4) scales to large networks.**

***Keywords*— service composition, QoS, multicast, application-level routing, overlay networks, fault tolerance**

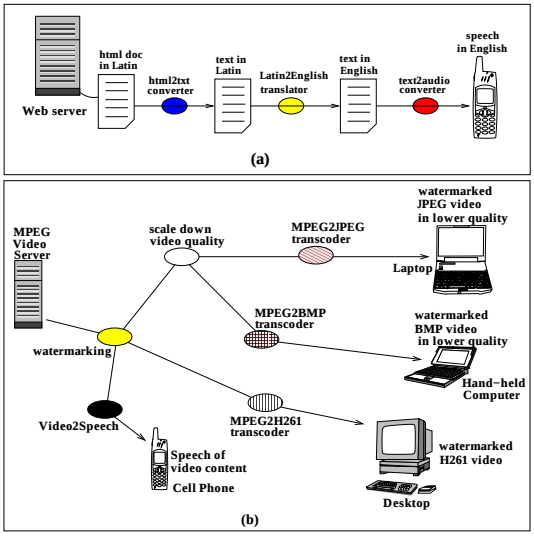

Fig. 1. Two scenarios that make use of composite services: (a) A mobile phone user retrieves a Web document written in Latin and hears it in English; (b) news video from CNN or Yahoo server is customized within a service network according to end users' network and machine capacities.

I. Introduction

The Internet has long been recognized as an environment with heterogeneities everywhere, happening in every aspect. The heterogeneity problem has been further exacerbated with the increasing popular use of small devices connecting to the Internet through wireless links in recent years. With a diverse spectrum of devices, ranging from powerful desktops, to less powerful and energy-sensitive laptops, hand-held computers, PDAs, and mobile phones, communicating over networks with varied bandwidths by using different protocols, there is a strong need to perform protocol and content translations between communicating parties to bridge the gaps. Value-added, transformational services have been proposed for such purposes [1], [2]. However, given the range of diversities involved in the Internet, developing monolithic transformational services to bridge all conceivable end-to-end heterogeneities would be, if not totally impossible, some task that requires tremendous amount of effort.

Fortunately, the *component service* model, which allows complex services to be dynamically aggregated from primitive ones, has been proposed and adopted in the Internet (e.g., in the Web and peer-to-peer networks) to achieve service flexibility and reusability [3], [4], [5], [6]. This new, flexible service model has triggered many interesting and useful Internet applications. We depict two different scenarios below.

- One-to-one scenario: Imagine a mobile phone user that wants to retrieve a Web document written in Latin and hear it in English. The original data can flow through a sequence of services (such as an html2txt converter, a Latin2English translator, and a text-to-speech converter) to get itself transformed before being delivered to the destination (Figure 1(a)). We call an end-to-end network path comprising a sequence of primitive service instances in a one-to-one scenario a *service path*.
- One-to-many scenario: Imagine a video distribution application that involves a single sender and multiple receivers, each of which requiring the original video content to be customized according to its own resource conditions (Figure 1(b)). Although it is feasible to transform and deliver the data through individually constructed end-to-end service paths, such a unicast delivery model may incur waste of bandwidths (due to redundancies in data delivery) and machine resources (due to redundancies in service exe-

This work was supported by NSF grants CCR-9988199, EIA 99-72884 EQ, and ANI 03-23434, and NASA grant NAG2-1406.

cution). We propose to use a single service tree (rather than multiple independent service paths) for transformation and delivery of the data, to save both network bandwidths and machine resources. We term such a group delivery model *service multicast*, to distinguish it from the traditional (data) multicast.

Service composition can be actually seen as the QoS routing problem with an additional requirement of paths/trees being functionally correct (i.e., paths and trees need to encompass required services in required dependencies). To differentiate the two delivery modes, hereafter we will use the terminologies *service-added unicast (routing)* and *service-added multicast (routing)* for routing in one-to-one and one-to-many service-oriented scenarios, respectively. We will see that the additional functionality requirement introduces unique challenges into the routing problem.

At deployment, for robustness and efficiency, each service needs to be replicated in multiple network locations (i.e., have multiple instances). Assuming an overlay network of nodes with services already deployed, we need to perform service composition at the runtime, by selecting service instances based on current network and machine conditions (for instance, ensuring that there is sufficient network bandwidth along end-to-end service paths).

For composite services to become widely accepted, automating the service routing process as well as coping with resource changes and failures is critical in enabling seamless provisioning of integrated services at the application layer despite the fact that an integrated service is actually distributed over multiple hosts in wide-area networks. In this work, we aim to devise an Efficient, Robust, Adaptive, and Scalable (ERAS) framework for the QoS service-added routing problem. Generally speaking, the built service paths/trees are efficient in terms of several routing criteria (such as delay, bandwidth, and computational resources), robust against network host failures[1], adaptive to current network and machine conditions as well as to dynamic multicast group membership. In addition, the solutions scale to large overlay networks.

The paper will be structured as follows. We first describe some background in Section II, followed by some related work in Section III. We briefly present our design overview in Section IV. In this paper, we employ a distributed hop-by-hop routing approach that does not rely on nodes having full routing states. Compared to traditional QoS routing, the problem of service routing is made much more complicated by the fact that now network nodes are injected multiple functionalities and the paths have to be functionally correct. Existing hop-by-hop approach for service unicasting is not satisfactory because the local-optimality property does not lead to global optimality (such as total delay) of the path. We enhance the solution in Section V by introducing geometric location awareness into Internet hosts, and have the geometric location information of the Internet hosts serve as guidance to compute more delay-efficient paths. In the one-to-many application domain, the problem of computing an optimal multicast tree in traditional routing has been recognized as an NP-complete problem. Having multiple functionalities in network nodes makes multicasting even more complicated. We build service multicast trees on an incremental basis in Section VI to naturally cope with dynamic membership features of a multicast group. Realizing that service multicasting has not fully explored network bandwidth sharing, in Section VII, we further propose to enhance service multicasting with data multicasting, thus providing a hybrid type of multicast solution. During the life time of a service path/tree, in order to keep the incrementally constructed service multicast tree near-optimal for the current membership and provide services at required QoS-level even at presence of failures, adaptation and failure recovery mechanisms are incorporated in the routing framework. We describe these issues in Section VIII. We validate the service-added QoS routing framework in the well-known network simulator *ns-2* and provide our performance results in Section IX. Section X gives some concluding remarks of this paper as well as directions for our future research.

[1]We assume only hardware failures (e.g., machine crashes) that are *fail-stop*, in the sense that failures are detectable by the timeout mechanism.

II. Background and General Issues

A. Service Model

An individual service component is associated with an input data QoS - Q_{in}, and an output data QoS - Q_{out}, where Q_{in} and Q_{out} are QoS vectors of multiple application-level QoS parameters such as image size, image resolution, video frame rate. Each service s has its resource usage function defined as $r_s : Q_{in} \times Q_{out} \rightarrow R$, that computes the amount of resources needed to deliver an output QoS Q_{out} when Q_{in} is the input QoS. When two services, s_i and s_j, are to be composed, then the output quality of s_i should equal the input quality of s_j. For instance, if two transcoders, MPEG2JPEG and JPEG2H261, are to be composed, then the output quality (e.g., frame rate, image solution, window size) of MPEG2JPEG must equal the input quality of JPEG2H261. The notation "$s_i \rightarrow s_j$" will be used to indicate that service s_i is followed by service s_j.

B. Background Components of the Framework Architecture

Our ERAS service composition framework helps to achieve maximum transparency to users at the application layer; the users should be unaware of the service component infrastructure as well as all or most negotiations related to solving the Internet heterogeneity problem. This can be achieved by delegating a user proxy to act on behalf of the user. Assuming the scenarios depicted in Figure 1, an end user may first contact a nearby proxy, which can then negotiate the QoS specifications between the server and client, to derive the Service Graph (SG) that is needed to bridge the gaps between the communicating parties. Once obtaining the SG, the proxy could further contact a service discovery agent to learn where instances of services are located. With this knowledge, depending on how much performance-related state information the proxy has about the network, the proxy can initiate service path computation. Below we depict the roles in more detail.

- *QoS compilation:* QoS compilation refers to a process of obtaining a Service Graph (SG) that is needed to bridge content and protocol gaps between two communicating ends based on their QoS specifications. A QoS compiler

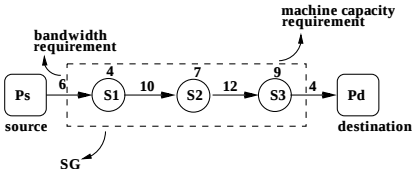

Fig. 2. A service request with linear service graph (SG): from the *source* to the *destination*, locate a QoS-satisfied path that encompasses $s_1 \rightarrow s_2 \rightarrow s_3$ in sequence.

such as [7] is capable of translating a user's request (e.g., get secure Video-on-Demand service with viewing quality of 30fps and in format of JPEG) into a linear or non-linear *service graph - SG*; i.e., a compositional service model or a *service request*[2]. In such an SG, all services have their input qualities and output qualities defined, according to the user's requirement. These application-level quality parameters will be further mapped, by the QoS compiler, into concrete resource requirements (e.g., in terms of CPU, memory, network bandwidth). Hereafter we will consider service graphs at resource level. A sample resource-level linear service request is shown in Figure 2[3].

- *service description and discovery:* Before a developed service component is deployed, it needs to be associated with an unambiguous name and/or an interface describing the component's inputs and outputs. WSDL (Web Service Description Language) is an XML-based language for describing Web services [8]. Service components need to be published and later on discovered before being composed. UDDI (Universal Description, Discovery and Integration) creates a standard interoperable platform that enables companies and applications to publish and find Web services [9]. Scalable ways of performing service discovery have been also investigated in peer-to-peer networks [10], [11].
- *service composition:* Since a service discovery system's task is only to locate instances of single services, and a QoS compiler's task is only to obtain a system-independent service graph, there needs to be a process, which we call *service-added routing*, that resides above these tasks and that can choose appropriate service instances (returned by a discovery system) for the logical components in a service request (returned by a QoS compiler), so that users at the application layer will perceive the application as an integrated service, rather than separate components (Figure 3). This routing substrate will be the focus of our study.

C. Notations and Terminology

Since server- and client-side applications rely on third-party entities (machines that are deployed with special services) to do transformations, we call these third-party entities *proxies*.

[2]The literature has used different terminologies, e.g., logical service path [3] and plan [5].

[3]Machine capacity refers to several issues, e.g., CPU and memory, and can be normally represented in an n-tuple vector. For simplicity, we represent the overall machine capacity by a single numerical value in the examples.

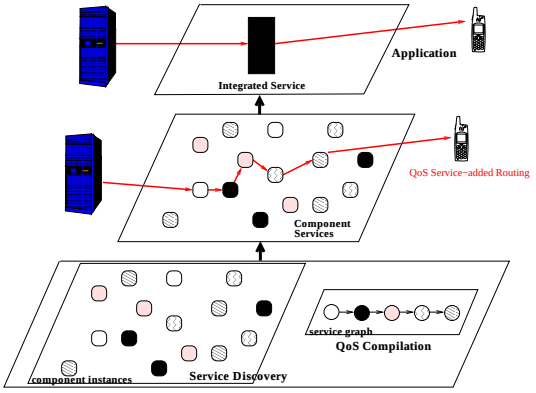

Fig. 3. The *service routing* substrate resides between the application layer and the service discovery/QoS compilation layer to make component services transparent to the application layer.

We denote the functional part of a service request as $r = (p_s, s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots, p_d)$. The request is for finding a service path between the source p_s and the destination p_d containing s_1, s_2, and $s_3 \ldots$, in sequence. A concrete service path actually represents a mapping from the service request to overlay nodes that are capable of providing the requested services at the required QoS level. A concrete service path will thus be denoted as $sp = (p_s \rightarrow s_1/p_\alpha \rightarrow s_2/p_\beta \rightarrow s_3/p_\gamma \rightarrow \ldots \rightarrow p_d)$ (s_i/p_θ, which we call a service node, means service s_i is provided by or mapped onto proxy p_θ).

Note that different from the traditional data routing, where paths should be loop-free, in service routing, data path loops are allowed, because a single network node is allowed to be visited multiple times if it is capable of serving multiple services in the request. We will see that this looping creates complexities in multicast tree management and failure recovery. We define *service neighbor* of a service s_i as s_i's proceeding service in service graphs. For instance, if $SG_1 = s_1 \rightarrow s_2 \rightarrow s_3$ and $SG_2 = s_1 \rightarrow s_4$, then s_1's service neighbor can be either s_2 or s_4, depending on which service graph is in use. We also define *next service hop* of a node x to be an instance of x's next logical service in the request. Thus, if $sp = (p_s \rightarrow s_1/p_\alpha \rightarrow s_2/p_\beta \rightarrow s_3/p_\gamma \rightarrow \ldots \rightarrow p_d)$, then p_s's next service hop is s_1/p_α, and s_1/p_α's next service hop is s_2/p_β and so forth.

III. Related Work

A. Service-Added Unicast Routing

Service-added unicast routing has been investigated extensively in different domains (Web [12], peer-to-peer networks [13], and company networks) and in different levels of network (physical network level [14] and overlay network level [12], [15]). Depending on the size of the network addressed, solutions to service routing fall into scalable and non-scalable ones.

In [12], [14], [16], global routing state (including QoS and service availability information) of the network is maintained at a single network node, so that computation of service paths can be performed in a centralized manner. Such an approach does not scale, considering the associated state maintenance overhead increases quickly with the network size. Better scalability can be achieved by introducing hierarchies into the network, so

that topology abstraction and state information aggregation become viable to significantly reduce the state maintenance overhead. A hierarchical solution was developed in [15]. Alternatively, scalable service routing can take a distributed approach by having the network nodes maintaining state information of a limited neighborhood and then having routing decisions made in a hop-by-hop manner. The work in [13] describes a hop-by-hop approach whose routing decision is based on local heuristics.

B. Service-Added Multicast Routing

Service-added multicast routing has been studied in [17], [18]. In our previous work [17], two algorithms for building service trees have been devised and their performances compared. The construction and the management of service trees take a source-based approach, which is simple and allows service trees to be computed quickly, and usually performance optimizations are better achieved. However, due to the rapidly increasing routing state maintenance overhead with the network size, scalability is constrained. Service tree construction in [18] is slightly more scalable, by having a DHT infrastructure maintain the global routing state as well as the service tree information. This approach imposes extra burden onto the DHT infrastructure. Furthermore, their landmark-vector-based clustering is not precise in predicting Internet distances among participating nodes. From the descriptions in [18], it is not clear how their approach maintains a service tree distributively.

C. Routing in Overlay Networks

Overlay network routing can be performed either on top of structured topologies [19], [17] or on top of unstructured topologies [20], [12]. The former approach views the overlay network topology as a partial mesh, so that the same routing protocols designed for the physical network, such as OSPF, MOSPF, and DVMRP, can be directly employed at the overlay layer. In the latter approach, hosts are considered fully connected, and for each application, a special topology, e.g., a multicast tree, is built and maintained.

IV. Design Overview

We describe the most salient features of our solution framework below:

- **Hop-by-hop routing based on resource conditions and geometric location information:** Without maintaining full state information, network nodes will jointly compute service paths in a hop-by-hop manner. Routing takes resource conditions as well as geometric location information of the Internet hosts into consideration.
- **Foreground topology and background topology:** To minimize delays, a service path or tree should be built without involving relay nodes; that is, network nodes that do not contribute special functionality to the application should be by-passed. Therefore, service paths/trees are built on top of an unstructured topology. However, we maintain a separate structured topology for background communications (i.e., for control messages).
- **Incremental multicast tree construction:** The problem of obtaining an optimal (Steiner) multicast tree in traditional routing has been known as NP-complete, and adding the requirement for service functional correctness makes the problem even more complicated. Traditional multicast tree construction has adopted heuristic solutions that usually branches out from an existing node to cover one member at a time, until all members have been included in the tree. Such a heuristics-based incremental solution also naturally supports the dynamic membership feature required by many applications. Our design is based on the same idea: the service multicast tree is built incrementally based on the unicast service routing solution. Moreover, we seek to optimize resource sharing by integrating data multicast into service multicast, thus providing a hybrid multicasting solution.
- **Local adaptation and local recovery:** Periodically, on-tree nodes monitor their local performances and trigger local adaptation or recovery when necessary. It is expected that the local adaptations together would help increase global tree optimality.

Below we present foundations that support our solution. More detailed descriptions are followed in later sections.

A. Service Discovery with Geometric Location Awareness

A service discovery system's task is to return service instances' locations, typically the IP addresses of the hosts in which instances are resided. However, with only the IP address information, it is hard to estimate how far away service instances are located from each other, thus making hop-by-hop routing decisions also hard if communication delay is a concern. We address this weakness by associating each Internet host with geometric coordinates (which will be retrieved by enhanced service discovery engines) and using it to estimate Internet distances (communication delays) between hosts.

The relative geometric coordinates of nodes in a large network can be efficiently obtained by the Global Network Positioning (GNP) approach [21]. In such an approach, a small set of landmark nodes L is first established and the distances among each other measured. The measured distance information is then mapped into a geometric space S of k dimensions such that the geometric distance between each pair of nodes best approximates the real-world measured distance. A regular host h can calculate its own coordinates in S after measuring its own distances to L. Through real-world measurements, the authors in [21] demonstrated that geometric distances obtained using their GNP method approximates the corresponding physical distances. As will be clear later, the added geometric location information in the service discovery system will serve us as guidance for finding more delay-efficient service paths/trees.

B. Topology Setup and Routing State Obtainment

To maximize routing efficiencies at the overlay layer, we do not set network topology constraints. That is, for data delivery, the initial network is a fully connected, unstructured topology, and a service path/tree is built for each application scenario. Routing state is measured on-demand; i.e., upon receiving a

request for routing, a network node initiates certain probing activities to learn about the resource conditions of its associated neighbor as well as the link conditions in between[4].

However, while service paths/trees are built on top of an unstructured overlay topology, we maintain another structured mesh topology for general control message communication. Note that the tree and the mesh are employed for different purposes: the former is used for content distribution and the latter is used for control messages.

For communication efficiency, we connect the overlay network nodes (for control message purposes) into a Delaunay triangulation [24], because Delaunay triangulation is a spanner graph that possesses some nice properties[5] and method of incremental construction of Delaunay triangulation network has been derived [24].

By using such a geometric topology, control messages can be routed by using an on-line, local routing method, such as the greedy approach or compass routing approach [25]. A local routing method has the advantage of not requiring global information of the topology in order to route from source to destination. For example, in compass routing, all information available at any point of routing is: the coordinates of the destination, the current position, and the directions of the outgoing links from the current node. Compass routing takes the following approach: starting at source, the current node chooses the outgoing link with the closest slope to that of the line segment connecting the current node to the destination. It has been proven that Delaunay triangulation D supports compass routing, which means that for every pair of nodes in D, compass routing always produces a path from source to destination [25]. Note that the adoption of Delaunay triangulation for overlay network topology and compass routing for control messages is only a choice in our design, for simplicity and efficiency purposes. Alternative network topologies and routing methods may be used as well.

C. Routing Approaches

We adopt a hop-by-hop approach to computing service paths based on routing states obtained by on-demand resource probing as well as the geometric location information of the service instances. Generally speaking, starting from the source node, we gradually add to the path those instances of required services as we route toward the destination.

The source may first discover the locations of all requested services' instances by invoking a geometric-location-enhanced service discovery system. After that, a service path can be resolved in a hop-by-hop manner as follows. Each hop sends QoS probe messages to all instances of its service neighbor, and then among the instances that satisfy resource requirements, the current hop will select the one that has largest amount of available resource and that is *on the way to destination*.

In this paper, our major focus will be on the less investigated, more challenging *QoS service multicast routing* problem, whose usefulness has been illustrated in Figure 1(b), and whose importance is undubious due to resource constraints in the physical world. While source-based (pure) service multicast has been proposed and studied in our previous work [17] for small service networks, we now consider the problem in a larger scale where centralized planning is unsatisfactory, for it becomes infeasible for a single network node to maintain full state information of the whole network. For better scalability, we devise a fully distributed approach for service multicast. By distributed, we mean not only service path/tree construction, but also multicast group management as well as tree maintenance (including adaptation and failure recovery), will be performed distributively.

Moreover, we propose to further optimize resource usages by integrating data multicast into service multicast, thus providing a combined multicast delivery mode which we call *hybrid multicast*.

V. Service Unicast Routing

Hop-by-hop QoS routing can be classified into two categories: single-path routing (SPR) and multiple-path routing (MPR). In SPR, one single path is probed for QoS, while in MPR, multiple candidate paths are probed, and then among the candidate paths, the best one is selected [26]. Usually MPR is done by multiplicating probe messages at outgoing links as the probing proceeds. To control probing overhead, special rules or mechanisms have to be adopted to constrain the number of probes multiplicated at outgoing links. MPR may find better paths than SPR, but at the cost of more message overhead. To minimize the overhead spent on probing, we adopt an SPR approach in this paper. However, we set guidance for SPR so that the probed path is likely to be a good one.

In QoS (data) routing, starting from one end, the shortest network path towards the other end is usually probed for QoS. If, at certain point, insufficiency of resources is detected, the probe will detour to other neighboring links/nodes [27]. In data routing there is always the shortest network path (maintained by, e.g., the distance vector or link state protocol) that serves as guidance for hop-by-hop QoS path finding so that the computed QoS-satisfied path is not unnecessarily long. However, in service routing, due to the unexpected functional dependency relations among services, no similar shortest *service* paths can be easily maintained as to allow a node to quickly lookup for the next service hop along the shortest service path to destination.

A. Local-Heuristics-Based (LHB) Approach

Existing SPR-based hop-by-hop unicast QoS service routing approach [13] works as follows: starting from the source, the current node selects, among many probed service neighbors, the one whose aggregate value of available bandwidth, machine resources and machine's up time is optimum. We name this approach *Local-Heuristics-Based* approach, because routing decisions are based on heuristics obtained within one hop of distance. The local heuristics alone, however, would only potentially optimize the path's overall concave or multiplicative metrics (e.g., the path's bottleneck bandwidth or robustness) and may help balance the network and machine loads, but does not pose any constraint on the length of the overall service path,

[4]Methods for measuring end-to-end available bandwidths can be found in [22], [23].

[5]A path found within a Delaunay triangulation has length bound by a constant times the straight-line distance between the endpoints of the path.

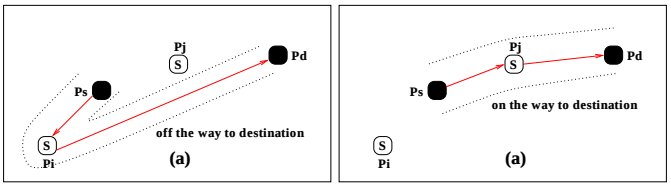

Fig. 4. (a) Based on LHB, p_s may choose a next hop that is not on the way to destination; (b) For making the routing aware of the hosts' geometric locations, p_s chooses a next hop that is on the way to destination.

which is an additive metric that requires special planning. Without planning of any sort for optimizing path lengths, service paths computed hop-by-hop by adopting local heuristics tend to be long, and inevitably consume more network resources.

A simple example is illustrated in Figure 4(a). Suppose we want to find a good instance of s between p_s and p_d, and suppose p_s detects that both instances of s (s/p_i and s/p_j) are equally good in terms of network bandwidth and machine capacity, for being unaware of the relative location of the two service instances, p_s may choose s/p_i, which is off the way to destination.

B. LHB Enhanced with Geometric Location Guidance

The weakness of *LHB* can be remedied if we enhance the service discovery system and let it return also the geometric location information of the queried service instances. By doing so, we can let the current node select the one, among the instances that satisfy all resource requirements, that lies on the shortest service path (estimated by the hosts' geometric locations) from current node to destination. This is illustrated in Figure 4(b): p_s chooses s/p_j because p_j lies on the shortest path from p_s to p_d; in other words, s/p_j is "on the way to destination".

Whether or not a service node lies on the way to destination can be computed as shown in the following example. For simplicity, the example focuses on optimizing overall path length by means of geometric location guidance, and we call this approach *Geometric Location Guided* (GLG). In Figure 5, we want to find a path between the source p_s and the destination p_d, with $SG = s_1 \rightarrow s_2 \rightarrow s_3$. Before starting the hop-by-hop routing, p_s invokes an enhanced service discovery system to learn about the locations (including IP address and geometric location) of candidate instances of all services in SG. In Figure 5(a), knowing p_1 and p_2 are hosts in which s_1 resides, p_s probes available end-to-end bandwidth and delay to p_1 and to p_2, as well as available machine capacities of p_1 and p_2. Based on the probing results, p_s derives the correspondent overlay map of service instances - a DAG (Directed Acyclic Graph) where nodes represent service instances and links represent dependency relations among the instances. First-hop nodes and links that do not meet resource requirements or are failed are excluded (represented in the figures in dashed circles or lines). At p_s, suppose both instances have sufficient resources, p_s then applies a shortest paths algorithm [28] on top of the DAG to obtain a shortest service path (shown in bold lines). The first hop along the shortest path will be chosen as our next hop, as it is on the way to future service instances and the destination (Figure 5(a')). In Figure 5(b), once at p_1, p_1 probes the resource conditions of three instances of next service in the request - s_2/p_3, s_2/p_4, and s_2/p_5. Figure 5(b') shows how p_1 chooses the most delay-efficient and QoS-satisfied next service hop. Note that in this case, the probed bandwidth between p_1 and p_3 does not meet the requirement, thus the correspondent link is deleted (shown in dashed line) from the service DAG. Such a hop-by-hop process continues until all of the services in the request have been resolved.

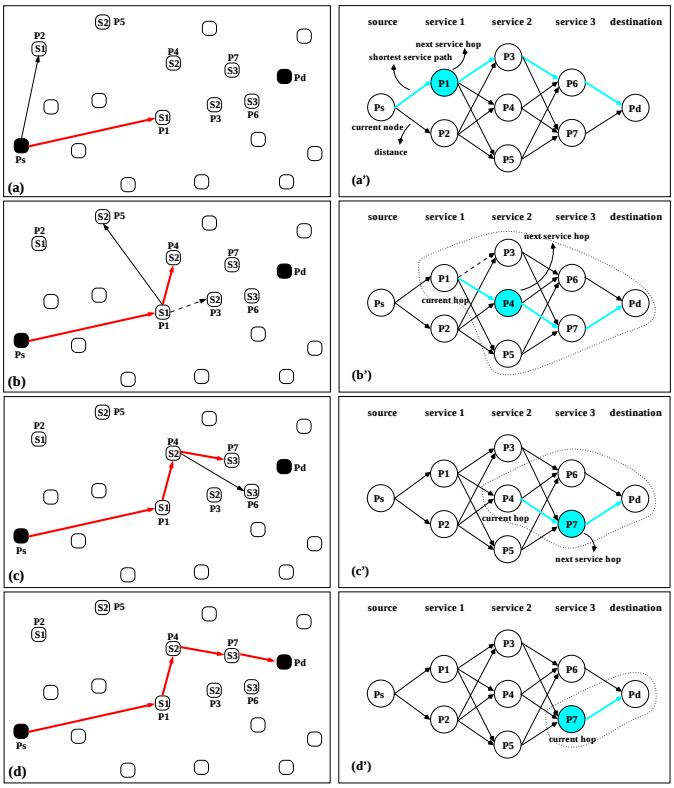

Fig. 5. Finding a QoS-satisfied and potentially shortest service path hop-by-hop from p_s to p_d that satisfies the service graph $s_1 \rightarrow s_2 \rightarrow s_3$.

Combining LHB and GLG, selection of next hop can follow one of the following approaches: (a) **LHB-GLG:** applies a shortest paths algorithm [28] on top of the DAG to obtain the shortest service paths (identifies the next service hops that potentially lead to shortest service paths), and then among the potential next hops that lie on the shortest paths select the one that is best in terms of available resources; (b) **GLG-LHB:** among the potential next hops that are best in terms of resources, select the one that potentially leads to shortest path. Approach LHB-GLG actually resembles the *widest-shortest* approach, and approach GLG-LHB resembles the *shortest-widest* approach in traditional routing. An additional advantage of LHB-GLG is that it would also reduce probing overhead, as only the hops along the shortest paths are probed for resource conditions.

C. Routing Backtracking

SPR-based hop-by-hop routing may end up in an unsuccessful state even if there exists a qualified path. For improved success rate, routing should backtrack to other unexplored branches if the current probe yields a dead end (e.g., when resource conditions of the candidate node-branches are

not satisfactory; or when probes yield no responses because of node/link failures). In this paper, we consider back-tracking to immediately-previous node if the current node/link yields unsatisfactory performance quality.

VI. Service Multicast

When a multimedia stream is delivered to a group of users that demand different transformational rules on the stream, then instead of having the stream transformed and delivered through multiple independent service paths, we should explore resource sharing, by construct a single service multicast tree for transformation and delivery purposes. To support the dynamic membership feature of many multimedia applications, we take an incremental approach to building service multicast trees. The incremental approach builds the tree by covering one member at a time to naturally cope with the dynamic membership feature of many applications.

A. Graftable Node

A key issue in multicast tree building is to find a point of attachment (*graftable on-tree node*) for the new joining member. In traditional data multicast, every on-tree node can be such a point because the original data from the root get forwarded as is by all on-tree nodes. The problem is only on how to find a good one. In the Protocol Independent Multicast (PIM) protocol, a newly joining member m's request is forwarded towards the source along the shortest path, and the first on-tree node n hit by the request becomes the *graftable on-tree node* for m.

Unlike the conventional data multicast, where every on-tree node functionally qualifies as a graftable node for all other group members, in service multicast, not all on-tree nodes functionally qualify as graftable nodes for other joining members. In fact, due to the functionality issues, an on-tree node n only qualifies as a graftable node for a member m (whose service request is r) if n's up-tree service path (the service path from the root to n) is a prefix of r. Let $sp = (p_s \rightarrow s_1/p_\alpha \rightarrow s_2/p_\beta \rightarrow s_3/p_\gamma \rightarrow s_4/p_\delta \ldots \rightarrow p_{d1})$ denote a service path, and let $r = (p_s, s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_5 \rightarrow \ldots, p_{d2})$ denote a service request, then several nodes (p_s, s_1/p_α, s_2/p_β, and s_3/p_γ) in sp qualify as functionally graftable service node for r. To maximize service sharing, we use the *longest match* (prefix) [17] criterion when selecting a graftable service node. We call the graftable service node selected by the longest prefix criterion the *best functionally graftable service node*. In this case, s_3/p_γ is the best functionally graftable service node, because $s_1 \rightarrow s_2 \rightarrow s_3$ is the longest prefix of sp and r and its last service - s_3 - is mapped onto p_γ.

B. Incremental Service Tree Construction

Construction of our service multicast tree will take the following procedures. Each member joining the multicast group would send its request r towards the source through the structured overlay network topology (in our case the Delaunay triangulation) by using compass routing. For each overlay node n_i that is hit by the request, it is verified if n_i is an on-tree node. If it is not, then n_i simply forwards the original request to the next hop (computed by compass routing) towards the source, and if it is, it tries to match r with the local copy of functional service tree T_f (management of T_f will be discussed further later) to identify the best functionally graftable service node n. The current node n_i then forwards the request to n if $n \neq n_i$. With a prefix of r satisfied at point n, n calculates the suffix of r, and starts a hop-by-hop routing process (by using a unicast service routing solution described in Section V) towards destination m for the suffix of r.

C. Tree Management

We now briefly describe the tree management issue. In data multicast, routers express their join/leave interests through IGMP (Internet Group Management Protocol) and, since all routers have one single function - to forward data as is, they basically need to be only aware of their children in the multicast tree. However, the same information is insufficient in service multicast due to additional service functionality constraints. In service multicast, in order to be able to identify graftable service nodes for new requests, an on-tree node must know the functional tree information of the multicast group. This implies that whenever the functional aspect of the service tree has been modified, tree state needs to be updated in all current on-tree proxy nodes by broadcasting adequate control messages within the multicast group. Note that although a single proxy may appear in multiple positions in a functional service tree, only one copy of the tree needs to be maintained *per physical node*.

D. An Example

Figure 6 depicts an example of how a service multicast tree is built and managed. In Figure 6(a), assume p_{d1} is the first group member. After p_{d1} has joined, the on-tree proxy nodes p_s, p_1, p_4, p_7, and p_{d1} will obtain a copy of the functional service tree - T_f - depicted on the right side of Figure 6(a). When p_{d2} joins, a service request $r_2 = (p_s, s_1 \rightarrow s_2 \rightarrow s_4, p_{d2})$ is sent from p_{d2} towards the source by using compass routing. The request hits an on-tree node p_1 before it reaches p_s. Since p_1 has a copy of T_f, it finds that p_4 is the best functionally graftable node for the current request, thus forwarding the request to p_4. In Figure 6(b), a service branch is established hop-by-hop from the graftable node p_4 to p_{d2}. Since the graftable node p_4 has already satisfied a prefix of r_2, only the correspondent suffix needs to be satisfied by the new service branch from p_4 to p_{d2}.

After finishing the join operation, p_{d2} broadcasts adequate message to on-tree nodes so that they incorporate the new functional branch into the old T_f. The functional service tree T_f maintained by all on-tree nodes will thus become that on the right-side figure of Figure 6(b). Note that T_f only needs to be updated if the service tree has been modified *functionally*. As an example, if a third join request has the form $r_3 = (p_s, s_1 \rightarrow s_2 \rightarrow s_3, p_{d3})$, then p_{d3} can get attached to p_7 without functionally changing the service tree. Therefore no updates are needed.

It is easy to see that service multicast definitely helps to save machine resources because each service in the functional service tree gets executed only once. It should also reduce network bandwidth consumption compared to service unicast, as in most

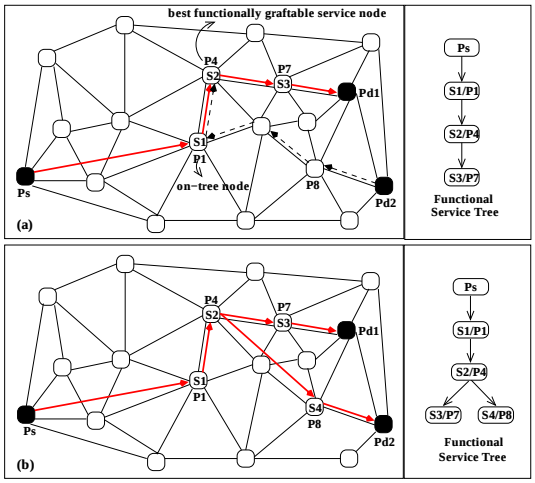

Fig. 6. (a) A service request message is sent from the newly joining member p_{d2} towards the source by using compass routing. The request hit an on-tree node p_1 before it reaches p_s. Since every on-tree node maintains T_f, p_1 found that p_4 is the best graftable node for the current request, thus forwarding the request to p_4. (b) A service branch satisfying the suffix of the original request is established hop-by-hop from the graftable node p_4 to p_{d2}.

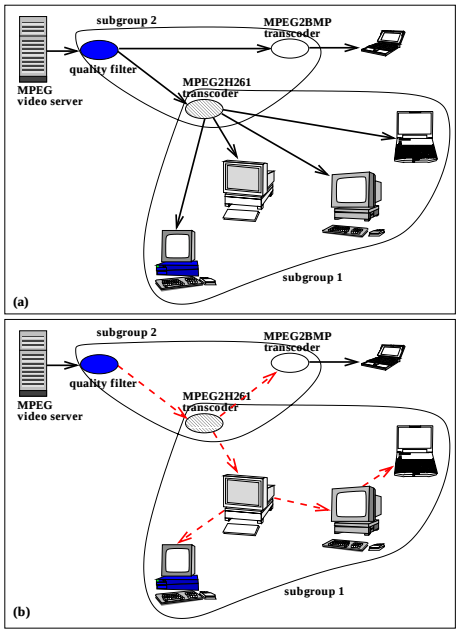

Fig. 7. (a) Pure service multicasting; (b) hybrid multicasting (service multicasting + data multicasting).

cases, we can expect the length of a service branch satisfying only the suffix of the request to be shorter than an individually built service path that needs to satisfy the whole request.

VII. Hybrid Multicast

In pure service multicast, each service branch gets directly attached to its best functionally graftable node. However, in doing so, bandwidth usage may not have been optimized. An example is illustrated in Figure 7(a): the proxy providing the MPEG2H261 transcoding service needs to send four separate copies of transformed data to its downstream nodes. Likewise, the node of quality filter will send two separate copies of filtered data to the downstream nodes. The scenario illustrates that data delivery in those sub-groups are sub-optimal. First, it is expensive to do so, because bandwidths need to be separately allocated. Second, after a node's (e.g., the one offering MPEG2H261) outbound network bandwidth usage reaches its limitation, then no new service branches can be created starting from that point.

We address these weaknesses by further employing data multicast in the local sub-groups. Although IP-layer multicast would be a solution, in this research, we will only exploit data multicast at the application layer because, different from the IP-layer multicast, application-layer multicasting does not require support from the infrastructural level. Our target is, taking Figure 7(a) as an example, to build a hybrid multicasting scenario that explores, in addition to service multicast, data multicast in the subgroups 1 and 2, as shown in Figure 7(b). In addition to boosting the overall cost efficiency of the service tree, exploring data multicast would also increase possibility of finding successful service branches when resources are scarce.

A. Tree Management

To realize such a hybrid multicast scenario, we make each on-tree (physical) proxy and (logical) service node to keep two trees respectively: the global functional service tree (T_f) and the local data distribution tree (T_d). Since two types of tree exist in the hybrid multicast case, we will call nodes on the functional tree T_f *on-functional-tree nodes* to explicitly mean they are nodes providing specific functionalities, rather than nodes that only perform relaying of data. The same as in service multicast, each on-functional-tree *proxy* will keep an updated T_f, which is the functional service tree of the whole multicast group. In addition to T_f, each on-tree *service node* n also keeps a T_d, whose root is itself, and whose lower-level members are its children in T_f (T_d should also maintain the location information of its nodes, for some purpose that will be clear soon). While T_f is global and its maintenance is still to enable on-functional-tree nodes to individually search for functionally graftable nodes for other joining requests, T_d is local and is maintained for exploiting benefits of data multicast in subgroups.

B. Parent Switching Protocol

When a new service branch b gets attached to a graftable node n, initially, n's T_d will have b's first node (say n') attached to itself. However, as n is aware of the geometric locations of its T_d's nodes, it will be able to identify which nodes are closer to n' than itself. If there is any such node, then n will initiate a *parent switching protocol*, so that at the end, n' gets attached to a closer parent with sufficient network bandwidth. Note that the parent switching protocol is only for switching parent in the local data distribution tree, it does not affect the global functional service tree.

The *parent switching protocol* works as follows. First, n sends n' a list of nearby nodes in an increasing order of distance. Upon receiving the list, n' starts to probe the bandwidth conditions from itself to the listed nodes one by one in the increasing order of distance. Once it finds a node whose outbound bandwidth to n' is sufficient for supporting the data stream, n'

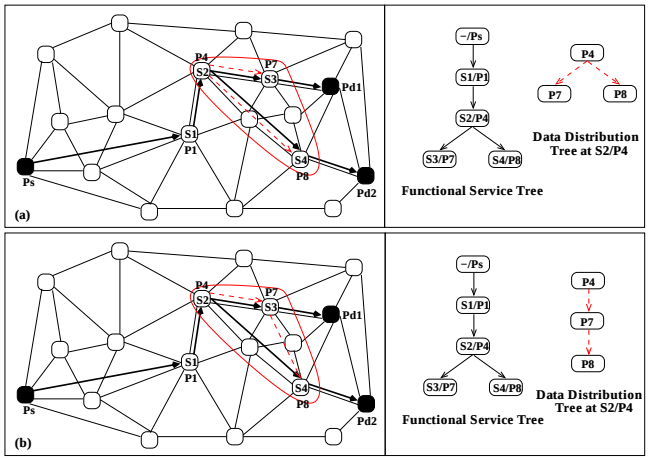

Fig. 8. Exploring data multicast in a service multicast scenario: (a) a new service branch's first node, p_8, is initially directly attached to the graftable service node p_4 (p_4 as p_8' parent in the local data distribution tree); (b) p_8 gets parent-switched to p_7 in the data distribution tree.

sends a request of *parent switching* to n, so that n will update n''s parent in its T_d. Different from T_f, which is maintained by every *on-functional-tree proxy*, a separate T_d needs to be maintained by every *on-functional-tree service node*. This means that if a single proxy offers different services in the multicast group, then it needs to keep multiple data trees.

C. An Example

Figure 8 depicts what the global functional service tree and the local data distribution tree would look like in the scenarios. In Figure 8(a), right after P_{d1} and P_{d2} have successfully joined the multicast group, the functional service tree kept by all on-tree service nodes and the data distribution tree at s_2/p_4 are shown on the right side of Figure 8(a). Subsequently, inside the subgroup (circled), the *parent switching protocol* will take place. Suppose p_7 is closer to p_8 than p_4, and suppose from p_7 to p_8 there is sufficient bandwidth to support the data stream, then p_8 will ask p_4 to switch parent, after which p_4's data distribution tree becomes the one shown on the right side of Figure 8(b).

With data multicasting in all subgroups, it can be expected that end-to-end service paths may become longer than in pure service multicast. However, such individual performance degradations would be justified by overall network bandwidth savings.

VIII. Routing Maintenance

Maintenance of paths/trees is called for due to network, traffic, and group dynamics. During the lifetime of a service path/tree, the on-path or on-tree nodes and links may have varying resource conditions, or may even fail completely. In parallel, new members may join the multicast group, and old members may leave, causing the tree structure to become "distorted" and its performance to degrade. Therefore, for the continuous operation of the application at a good QoS level, adaptation (e.g., service multicast tree rearrangement) and failure recovery are mechanisms that need to be incorporated in the service routing framework.

A. Tree Rearrangement

The natural consequence of constructing a multicast tree incrementally is that over time, as new branches are added and existing branches are pruned, the tree structure may become sub-optimal. To maintain a good tree structure, selections of service instances have to take currently covered members into account, which means that previously selected service nodes may need to be relocated.

A centralized tree rearrangement approach for traditional multicasting has been studied in [29], [30]. However, a centralized approach is inappropriate in our case in which the service multicast tree is constructed and managed in distributed manners. We adopt a distributed approach by distributing the task of tree rearrangement (including performance monitoring and rearrangement itself) to all on-tree nodes. The basic idea is as follows: on-tree nodes monitor their regional performances[6] and trigger local tree rearrangement if necessary. We expect that the local rearrangements together would contribute to global tree improvement.

Although theoretically tree rearrangement can be performed every time a join or leave has occurred, in practice, the disturbance caused by excessive changes may be intolerable to the ongoing multicast sessions, as packets are constantly in flight within the tree. Replacement of a node with large number of downstream members may cause large disturbance (as all downstream members may perceive some data loss). On the other hand, the change will also benefit all downstream members (i.e., utility is large). Considering tradeoffs between performance gain and disruption of services, certain threshold needs to be maintained to suppress those adaptation operations whose performance gains are not significant enough.

Let x be an on-tree service node providing service s, and y be a candidate service node that is capable of providing s but is not on-tree. If y replaces x, we define the potential performance improvement as $\gamma = \frac{c(x)-c(y)}{c(x)}$[7]. Disturbance can be measured as packet loss rate (at fine granularity) or the number of downstream members that perceive data loss (at coarse granularity), and utility can be defined as the fraction of benefited members. We denote the disturbance caused by replacement of x as θ, and the utility associated with the replacement as μ. We therefore define the real benefit β of replacing x by y as a function of performance gain, disturbance, and utility: $\beta = \gamma * \frac{\theta}{\mu}$, and replacement only takes place if β is larger than threshold.

Once a performance monitor has detected that a replacement would yield a performance improvement β that is higher than the threshold value, the current service node x will hand over its roll to y. It is important that parent and child service nodes do not perform handovers simultaneously, since they use each other as a reference point in the detection phase. To guarantee this property, before handing over, the current service node needs to synchronize with its parent and child nodes (basically blocking them from doing concurrent handovers).

We better illustrate the idea with a simple example in Figure

[6] We define a *region* in a multicast tree to be the neighborhood of an on-tree node, including its upstream and downstream nodes as well as the links connecting those.

[7] The term $c(x)$ is defined as the sum of x's neighboring link costs ($c(x)$), which are measured as delays in this paper.

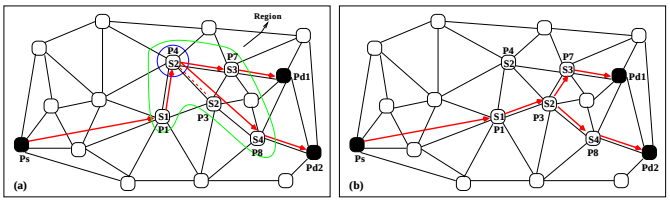

Fig. 9. Tree rearrangement: (a) All on-tree nodes monitor their local performances. For instance, p_4 monitors its local performance and tries to compare itself with an alternative candidate service instance at p_3. (b) p_4 hands over its roll to p_3 because the replacement would yield better local performance.

14. The circled service node represents a performance monitor that monitors the regional performance by periodically checking if there are other candidates (found by invoking a service discovery system) with better performances that can replace itself. In the example, p_3 is a node also serving s_2. Node p_4 then asks p_3 to probe the available machine resource as well as the delay and available bandwidths between p_3's potential parent and children if p_3 were to replace p_4. Suppose p_3 satisfy all resource requirements and also has the benefit β improved by some percentage larger then the threshold, then p_4 will hand over its roll to p_3.

B. Failure Recovery

Traditional failure recovery mechanisms used in routing fall into two approaches: *protection-based approach* and *restoration-based approach* [31]. In the protection-based approach, dedicated protection mechanisms, such as backup paths, are employed to cope with failures on the primary path. In the restoration-based approach, on detection of a failure, an attempt is made to reroute the path around the faulty nodes and links.

The protection-based approach has been adopted in [32], [13] for recovering single service paths. However, this approach is not suitable in multicast scenarios because of two reasons: (1) it is prohibitively expensive, in terms of resource allocations, to maintain one or more backup trees for the primary tree; (2) the dynamic membership feature causes the primary tree to change over time, thus there would be too much of overhead to keep the backup trees up-to-date. For these reasons, the restoration-based approach is more suitable for multicasting.

In this paper, we consider only hardware failures, and assume the fail-stop failure model in the sense that failures are detectable (e.g., by use of timeout). Different from traditional routing, in which failure of a node or link means only a single failure on the path or tree, in service-added routing, failure of a single physical node or link may trigger failures of several spots in the service path or tree. This is so because a single network node may be contributing several services in the path or tree.

Before discussing failure recovery, we first need to devise a failure detection mechanism. While the use of heartbeat messages is a common mechanism for failure detection, there are additional challenges caused by the fact that one physical node can serve multiple (consecutive or non-consecutive) component services. Due to the dependency complexities, we need to further derive the *physical node dependency graph* for detection (this will be clearer as we show an example later). Each on-tree network node then periodically sends heartbeat messages to its *physical* parent and, if the parent does not respond within a specified time, then the current node will infer that the parent has failed. Upon the detection, the current node n tries to find out its closest live ancestor, and asks it to initiate a hop-by-hop routing process towards n to locate suitable instances for the failed services in between.

An example is shown in Figure 10. Figure 10(a) depicts a functional service tree together with the group members. As stated, each physical node monitors the liveness of its physical parent. The physical node dependency graph that shows the monitoring relations is shown on the right side of Figure 10(a).

Failures of some nodes (e.g., p_2 and p_3) are simpler to deal with, while failures of certain other nodes (e.g., p_1) yield more complex situations. For example, if p_2 fails (detectable by member 1), then member 1 will try to locate a live ancestor closest to itself (in this case p_1) and afterwards p_1 initiates a hop-by-hop routing process towards member 1 to recuperate service s_3. Failure of p_1 is more complex to deal with, as the node participates in multiple positions and branches of the tree. The failure itself may be detected by three nodes: p_2, p_3, and member 2, which report to their closest live ancestors - the root and the node p_3 in this case.

We discuss only recoveries initiated by the root, as this is a complex case involving parallel failures of multiple branches. While the root may recover one branch at a time simply based on the arrival time of the requests, certain overheads incurred by recovery synchronization need to be considered. If: (1) p_2*'s request precedes* p_3*'s* - once p_2's request has been satisfied, p_3's request can be ignored (because p_3's request is part of p_2's request); (2) p_3*'s request precedes* p_2*'s* - p_2's recovery request can only be initiated after p_3's request has been satisfied, because the recovered service node (say s_1 is mapped to p_1') s_1/p_1' will serve as a reference point for part of p_3's request. Furthermore, p_3's recovery request will be initiated by p_1' instead of the root. From this example, we see that different recovery orders will affect the overall recovery time due to delays associated with communication and synchronization.

Optimization of the recovery ordering is hard to achieve because node R (the closest ancestor that is responsible for initiating recovery operations) is unable to predict the total recovery time. To overcome this problem, we employ a heuristic of *minimum recovery dependencies (MRD)* to try to minimize the overall recovery time. Assuming nodes report failures independently: upon receiving failure report from one node, node R is able to deduce, from the functional tree information, if other branches would be affected by the failure. In the example of Figure 10, if p_3's failure report arrived at the root first, the root is able to deduce that the failure also would affect p_2. Using the MRD heuristic, the root can initiate recovery action for p_2 even without receiving p_2's failure report, because p_3's request will get naturally satisfied after p_2's request gets satisfied (thus reducing recovery dependencies).

An alternative approach for dealing with failure reporting and recovery works as follows: since p_3 (upon detecting failures of p_1) is able to deduce that another node - p_2 - will also eventually detect the same failure, p_3 may just adopt a *lazy failure reporting* mechanism by backing off its report indefinitely. By doing so, the number of total error reports will be reduced. However,

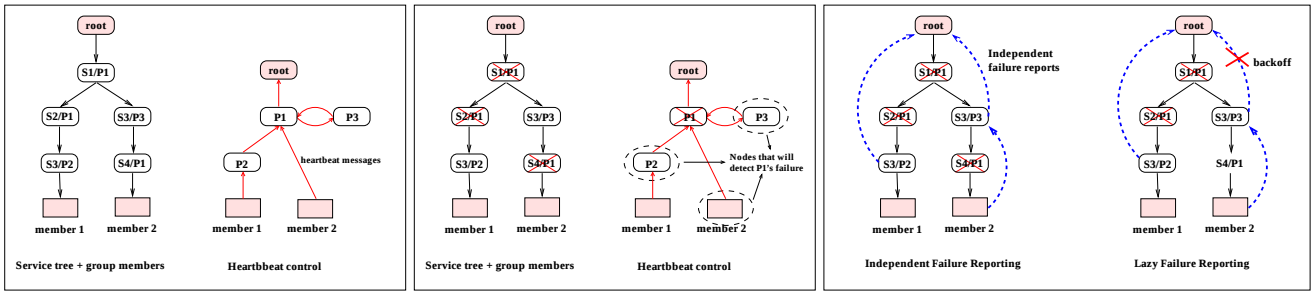

Fig. 10. Detection of failure: (a) *left:* a service tree plus the membership information; *right:* the graph that represents the monitoring relations among the nodes. (b) *left:* the failure of p_1 triggers failures of multiple service nodes in the tree; *right* The failures are individually detected by p_2, p_3, and member 2; (c) *left:* independent failure reporting; *right:* lazy failure reporting that back-off failure reporting if possible.

this may increase the total recovery time as p_2 may only be able to detect the same node failure later. In order to follow the heuristic of *minimizing recovery dependencies*, only node p_3 should back off failure reporting; node p_2 should always report immediately in this case.

IX. Performance Study

We implemented the service routing framework (including service unicast, pure service multicast, and hybrid multicast delivery modes) in the well-known network simulator *ns-2*. This section is devoted to performance studies of the proposed approaches.

A. Evaluation Methodology

Our physical Internet topologies are generated by the *transit-stub* model [33], by using the GT-ITM Topology Generator software. A number of physical nodes are randomly chosen as proxy nodes, whose service capability and machine capacity are assigned by certain functions. The end-to-end available bandwidth from an overlay proxy node a to another overlay proxy node b is the bottleneck bandwidth of the shortest physical path from a to b. Among the physical network nodes, a small set of them (10 nodes) are chosen to be the landmark nodes - L, based on which the proxies can derive their coordinates in the geometric space defined by L[21]. We use geometric space of 5 dimensions in our simulations; calculation of geometric coordinates is done by using the software available at http://www-2.cs.cmu.edu/ẽugeneng/research/gnp/. Construction of the Delaunay triangulation overlay mesh for control message purposes is aided by the Qhull software developed by the Geometry Center at University of Minnesota (http://www.geom.umn.edu/software/qhull).

We use the following performance metrics for the evaluations of routing approaches (performance metrics of other issues such as adaptation and failure recovery are described later in the result sections):

- *Host Utilization*: is the ratio of amount of machine resources in use to the machine's total amount of resources. In simulations, we represent a machine's computing capacity as a single numerical value, although in reality, it should be a resource vector of multiple parameters (e.g., memory, cpu).
- *Link Utilization*: is the ratio of used bandwidth to the total bandwidth of the physical network links that measures how much the physical links are loaded.
- *Service Path Length*: is the sum of individual virtual link lengths that make up the service path, where the virtual link lengths are represented as end-to-end delays.
- *Delay* $\times$ *Bandwidth Product*: The purpose of this metric is to measure the volume that the streaming data occupies in the network. For example, if the streaming data requires 2MB of bandwidth on a physical link whose single trip delay is 10ms, then the volume of data is said to be 20MB*ms.
- *Path Finding Success Rate*: is the rate of finding service paths successfully. Service path finding failures may occur when resources are scarce, or when there is no instance of the required service(s). However, in our following tests, there will be always at least one instance of each service in the system, thus failures can only be caused by resource scarcity.

B. Performances of Different Service Unicast Approaches

In this section, we measure performances of the service unicast approaches (*GLG*, *LHB*, *GLG-LHB*, and *LHB-GLG*) described in Section V. We further run a hop-by-hop approach that is based on random-walk (*RANDOM*), to serve as a base case to all of the approaches in study.

The simulation settings for the test are as follows. The physical network contains 600 nodes, and among them, 10 are selected as landmarks and 500 as proxies. We randomly generated 5000 requests between randomly selected pairs of proxies. We compare the performances under two different resource settings: one with sufficient resources to admit all service requests, and the other with insufficient resources, in which case late join requests may get rejected because of resource scarcity.

Sufficient-resource settings: In sufficient-resource settings, since all service requests get successfully admitted, the performance metrics of interest are *host utilization*, *link utilization*, *service path length*, and *delay* $\times$ *bandwidth product*. The comparative results of several service unicast routing approaches are shown in Figure 11[8]. As has been predicted, since *GLG*

[8]For visibility, link utilization and proxy utilizations are plotted as a transformed *inverse cumulative distribution function*, also known as *inverse survival function*. Service path length is plotted as an *inverse cumulative distribution function*.

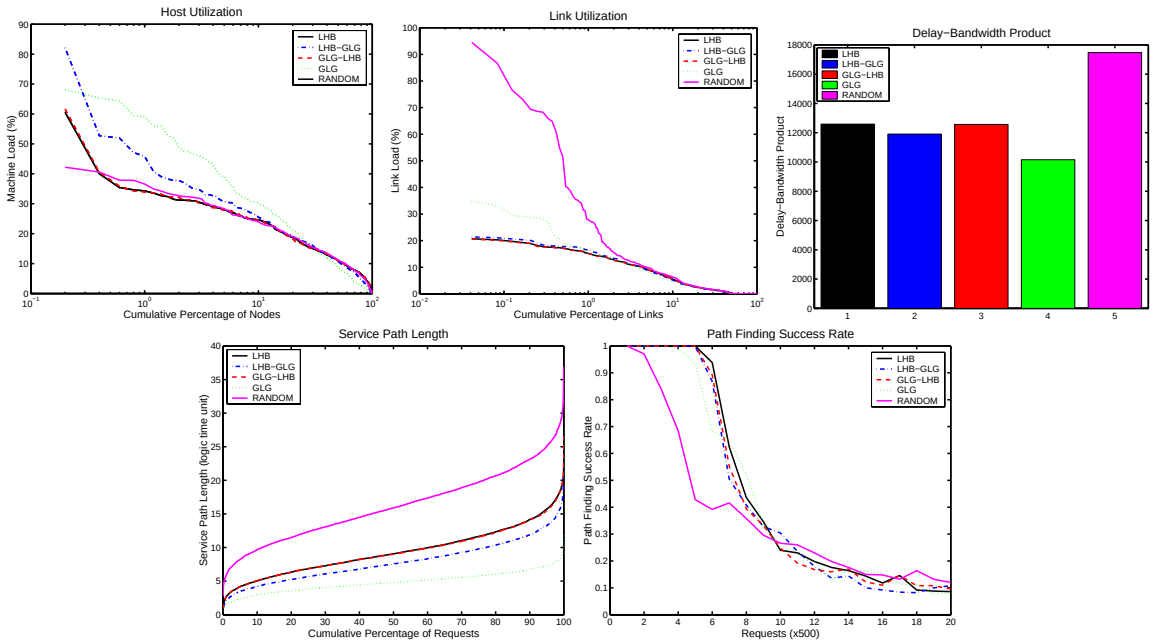

Fig. 11. Comparisons of the service unicast approaches in terms of: (a) *host utilization*; (b) *physical link utilization*; (c) *delay-bandwidth product*; (d) *service path length*; and (e) *gradual path finding success rate in the backtracking-off mode*.

genuinely seeks shortest QoS-satisfied service paths, load balancing on hosts and links is poor. This is indicated by the fact that the *GLG* curves are steep. *LHB* does in fact help to keep a more balanced network and machine load, as the next service hop is the one that maximizes an aggregate function of available bandwidth and machine capacity. On the other hand, *LHB* performs poorly in terms of *delay bandwidth product* (Figure 11 (c)) and *service path length* (Figure 11 (d)), because service paths computed by *LHB* are long, and therefore demand more network resources. However, in these respects *GLG* performs best, because service paths computed by this approach tend to be short, and as such, require less network resources. *GLG-LHB*'s performances are quite close to those of *LHB*, and *LHB-GLG* has good performances overall.

Insufficient-resource settings: After certain resources get exhausted, a join request may be denied. The performance metric of interest in such an insufficient-resources scenario is *path finding success rate* which, in some way, indicates how well load balancing is achieved. Figure 11(e) shows the path finding success rates of the different service unicast approaches with back-tracking turned off. As has been expected, since *GLG* does not take load balancing into consideration, certain resources may become exhausted more quickly than other approaches that consider load balancing, and as a consequence, *path finding success rate* was lowest in *GLG*. In the backtracking-off mode, LHB and LHB-GLG have achieved similar aggregate success rates. However, when when we turn on backtracking, the aggregate success rate of LHB-GLG surpasses that of LHB by 4.9%. This is because LHB-GLG has incurred less network resource consumption.

From the above performance analyses, we see that none of the approaches performs best in all aspects. *GLG*'s performances in terms of *service path lengths* and *delay-bandwidth product* are significantly superior to others', but is worst in *path finding success rates*. *LHB* is one of the best in finding service paths successfully, but incurs longer service paths than others and as a consequence, tends to require more network resources. From all approaches, *LHB-GLG* seems to have best balanced these contradictory factors, as it incurs relatively short service paths while maintaining a high path finding success rate.

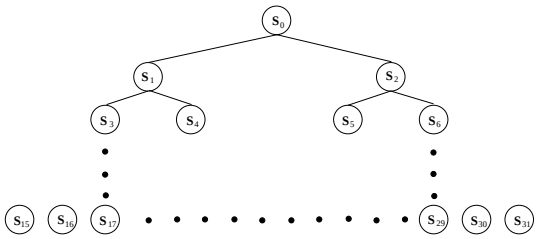

Fig. 12. Binary functional service tree.

C. *Service Unicast* vs *Pure Service Multicast* vs *Hybrid Multicast*

In this section, we study the performance benefits of employing pure service multicast and hybrid multicast. Since *LHB-GLG* is a service unicast approach that strikes best balance among the performance metrics, we employ *LHB-GLG* as the building block for incrementally constructing a multicast tree. Simulations are run for multicast group sizes of 256, where service requests are randomly selected from a binary functional service tree as shown in Figure 12; a service request is a logical path from the root to a random leaf.

For these comparisons, we set up sufficient-resource environments. As we can see from Figure 13 (a), there is not too much difference, in terms of host utilization, between pure service multicast and hybrid multicast. This was expected because local data multicast would not further diminish the number of

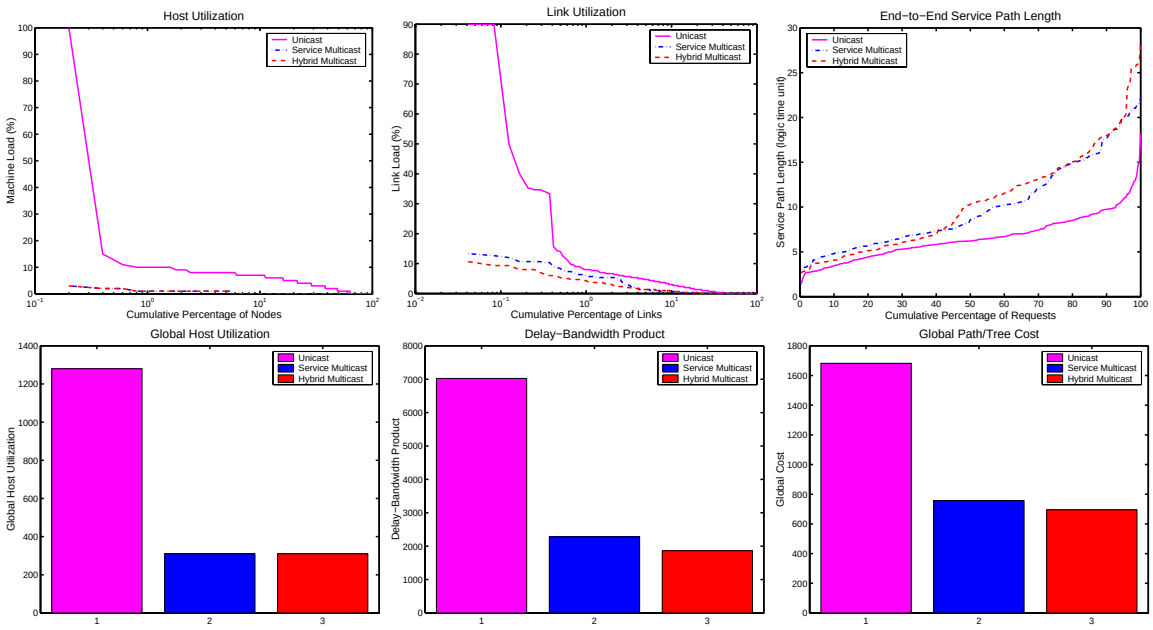

Fig. 13. Comparisons of: (a) *host utilization*; (b) *physical link utilization*; (c) *end-to-end service path length*; (d) global host utilization; (e) *delay bandwidth product* and; (f) *global service path/tree cost* among the different delivery modes: *service unicast*, *pure service multicast*, and *hybrid multicast*.

service executions. Figure 13 (b) shows that hybrid multicast yields much lower link utilization than pure service multicast. Not surprisingly, the two multicast cases yield tremendous delay bandwidth product savings compared to unicast (Figure 13 (e)). Compared to service unicast, service multicast incurs longer end-to-end service paths in all cases, and hybrid multicast incurs longer paths than service multicast in most of cases (Figure 13 (c)). However, the longer end-to-end service paths in hybrid multicast are justified by lower global tree costs (Figure 13 (f)) due to service path sharing.

D. Tree Rearrangement

The local tree rearrangement operations together contribute to a global tree quality improvement. As described in Section VIII-A, we have set thresholds to suppress those tree rearrangement activities that only yield small performance gains. We therefore study the relations between threshold and global performance. For simplicity, it is assumed that the effects of θ and μ in local benefit $\beta = \gamma * \frac{\theta}{\mu}$ cancel off.

Figure 14 depicts the total tree costs (in logical units) after adaptations versus local adaptation thresholds with different service distribution probabilities[9]. The experiment settings were as follows: similar to the settings described in Section IX-C, we used group sizes of 256, whose service requests are drawn from a pool of binary functional tree shown in Figure 12. We first run the service multicast tree construction program to incrementally build a service multicast tree. After that, the local performance monitors are turned on, and local rearrangements are triggered if the potential performance improvement is larger than the local threshold values. The total tree cost is measured after all local rearrangement operations have been stabilized. At low threshold values, tree rearrangements are triggered more often, thus leading to better global performances. At higher threshold values, only those rearrangement operations that yield larger performance gains are triggered. Therefore, leading to lower global performance gain (higher final global tree cost). We can also see differences of global tree costs for different service distribution probabilities: sparser service distribution yields higher tree costs and denser service distribution yields lower tree costs overall, which are in match with our intuitions.

[9] Service distribution probability x means that each component service is randomly distributed at $x\%$ network nodes.

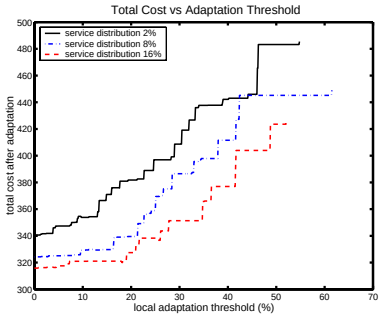

Fig. 14. Total tree cost after adaptations *vs* local adaptation threshold with different service distribution probabilities.

E. Failure Recovery

We compare performances of three failure reporting and recovery approaches as described in Section VIII-B: *independent reporting and independent recovery* (IRIR), *independent reporting and heuristic-based recovery* (IRHR), and *lazy reporting and heuristic-based recovery* (LRHR). The approaches are evaluated under three metrics: number of failure reports, global tree costs after recoveries, and time needed for recovery. The experiment was conducted as follows: still using multicast group size of 256 and binary functional tree as before, after the service multicast tree has been built, we randomly

failed on-tree network nodes one by one. The results shown in Table I are based on 3 runs, each run with 20 node failures. The numerical values have been normalized based on the results of IRIR (base case). We can see that the lazy reporting mechanism helps to reduce the number of failure reports significantly, and the heuristic-based recovery mechanism helps to maintain better-cost service multicast trees after recoveries. Between IRHR and LRHR, there is tradeoff: while IRHR incurs better recovery time, it yields larger number of failure reports than LRHR. This is so because by having the network nodes independently reporting failures to a live ancestor, the ancestor is likely to be aware of the failure sooner, and thus can start recovery operations sooner. On the other hand, by adopting the lazy reporting mechanism (LRHR), failures are likely to be noticed later, because certain failure reports are suppressed because the detecting node assumes that other nodes will eventually detect and report the same.

metrics	IRIR	IRHR	LRHR
failure reports	100	100	74.2
global cost after recovery	100	87.5	87.5
recovery time	100	85.1	93.6

TABLE I

PERFORMANCE COMPARISONS AMONG IRIR, IRHR, AND LRHR.

X. CONCLUSIONS

In this paper, we have seen the challenges and complexities introduced by additional functional requirements into QoS routing, and have provided efficient, robust, adaptive, and scalable solutions for the service-added QoS routing problems in two different domains: unicast and multicast. We further provided a hybrid multicasting solution (by integrating traditional data multicast into service multicast) to achieve best resource sharing.

The component service technology has been widely advocated in the past few years, and has spawned quite a few new research problems, including service discovery, service composition/orchestration, security, service deployment. Service composition (both in unicast and multicast domains) in this paper rely on existing service discovery agents, and assumes that component services have been pre-distributed. In our future work, we plan to study the problem of automatically deploying component services (e.g., based on their access patterns) to fully automate component services from deployment to usage.

REFERENCES

[1] Rakesh Mohan, John R. Smith and Chung-Sheng Li, "Adapting Multimedia Internet Content for Universal Access," *IEEE Transactions on Multimedia*, Mar 1999.

[2] Surendar Chandra, Carla Schlatter Ellis, and Amin Vahdat, "Application-Level Differentiated Multimedia Web Services Using Quality Aware Transcoding," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 12, Dec 2000.

[3] 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," *Special Issue of Computer Networks on Pervasive Computing*, 2001.

[4] A. Ivan, J. Harman, M. Allen, and V. Karamcheti, "Partitionable Services: A Framework for Seamlessly Adapting Distributed Applications to Heterogeneous Environments," in *Proc. of IEEE International Conference on High Performance Distributed Computing (HPDC)*, Edinburgh, Scotland, Jul 2002.

[5] Shankar R. Ponnekanti and Armando Fox, "SWORD: A Developer Toolkit for Web Service Composition," in *the Eleventh World Wide Web Conference (Web Engineering Track)*, Honolulu, Hawaii, May 2002.

[6] Stefan Tai, Rania Khalaf, and Thomas Mikalsen, "Composition of Coordinated Web Services," in *Proc. of ACM/IFIP/USENIX International Middleware Conference (Middleware 2004)*, Toronto, Canada, Oct 2004.

[7] Duangdao Wichadakul, *Q-Compiler: Meta-Data QoS-Aware Programming and Compilation Framework*, Ph.D. thesis, Computer Science Department, University of Illinois at Urbana Champaign, Jan. 2003.

[8] Erik Christensen, Francisco Curbera, Greg Meredith, Sanjiva Weerawarana, "Web Services Description Language (WSDL) 1.1," Mar 2001.

[9] UDDI.ORG, "Universal Description, Discovery and Integration (UDDI Technical White Paper)," Sep 2000.

[10] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, Hari Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," in *ACM SIGCOMM*, San Diego, California, Aug 2001.

[11] Sylvia Ratnasamy, Pau Francis, Mark Handley, Richard Karp, Scott Shenker, "A Scalable Content-Addressable Network," in *Proc. of ACM SIGCOMM*, San Diego, CA, Aug 2001.

[12] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, Quan Z. Sheng, "Quality Driven Web Services Composition," in *The Twelfth International World Wide Web Conference*, Budapest, Hungary, May 2003.

[13] Xiaohui Gu, Klara Nahrstedt, "A Scalable QoS-Aware Service Aggregation Model for Peer-to-Peer Computing Grids," in *Proc. of High Performance Distributed Computing*, Edinburgh, Scotland, Jul 2002.

[14] Sumi Choi, Jonathan Turner, and Tilman Wolf, "Configuring Sessions in Programmable Networks," in *Proc. of IEEE INFOCOM*, Anchorage, Alaska, Apr 2001.

[15] Jingwen Jin and Klara Nahrstedt, "Large-Scale Service Overlay Networking with Distance-Based Clustering," in *Proc. of ACM/IFIP/USENIX International Middleware Conference (Middleware2003)*, Rio de Janeiro, Brazil, Jun 2003.

[16] Jingwen Jin and Klara Nahrstedt, "Source-Based QoS Service Routing in Distributed Service Networks," in *Proc. of IEEE International Conference on Communications 2004 (ICC2004)*, Paris, France, Jun 2004.

[17] Jingwen Jin and Klara Nahrstedt, "On Construction of Service Multicast Trees," in *Proc. of IEEE International Conference on Communications (ICC2003)*, Anchorage, Alaska, May 2003.

[18] Zhichen Xu, Chunqiang Tang, Sujata Banerjee, Sung-Ju Lee, "RITA: Receiver Initiated Just-in-Time Tree Adaptation for Rich Media Distribution," in *13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV03)*, Monterey, CA, Jun 2003.

[19] Y. Chu, S. G. Rao and H. Zhang, "A Case For End System Multicast," in *Proc. of ACM SIGMETRICS*, Santa Clara, CA, Jun 2000, pp. 1–12.

[20] Paul Francis, "Yoid: Extending the Internet Multicast Architecture," Apr 2000.

[21] T. S. Eugene Ng, Hui Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," in *Proc. of IEEE INFOCOM*, New York, NY, Jun 2002.

[22] B. Melander, M. Bjorkman, and P. Gunningberg, "A New End-to-End Probing and Analysis Method for Estimating Bandwidth Bottlenecks," in *Proc. of Global Internet Symposium*, 2000.

[23] Manish Jain, Constantinos Dovrolis, "End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput," in *Proc. of ACM SIGCOMM*, Pittsburgh, PA, Aug 2002.

[24] J. Liebeherr, and M. Nahas, "Application-Layer Multicast with Delaunay Triangulations," in *Proc. of Sixth Global Internet Symposium (IEEE Globecom 2001)*, San Antonio, Texas, Nov 2001.

[25] Evangelos Kranakis, Harvinder Singh, Jorge Urrutia, "Compass Routing on Geometric Networks," in *Proc. of the 11th Canadian Conference on Computational Geometry*, Vancouver, CA.

[26] S. Chen, K. Nahrstedt, "An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions," *IEEE Network Magazine*, vol. 12, no. 6, pp. 64–79, 1998.

[27] Shigang Chen, Klara Nahrstedt, Yuval Shavitt, "A QoS-Aware Multicast Routing Protocol," *IEEE Journal on Special Areas in Communication*, vol. 18, no. 12, pp. 2580–2592, Dec 2000.

[28] Jingwen Jin, Klara Nahrstedt, "QoS Service Routing for Supporting Multimedia Applications," Tech. Rep. UIUCDCS-R-2002-2303/UILU-ENG-2002-1746, Department of Computer Science, University of Illinois at Urbana-Champaign, USA, Nov 2002.

[29] A. Chakrabarti and G. Manimaran, "A Case for Scalable Multicast Tree Migration," in *Proc. of IEEE Globecom*, San Antonio, USA, Nov 2001.
[30] R. Sriram, G. Manimaran and C. Siva Ram Murthy, "A Rearrangeable Algorithm for the Construction of Delay-Constrained Dynamic Multicast Trees," *IEEE/ACM Transactions on Networking*, vol. 7, no. 4, pp. 514–529, 1999.
[31] Aaron Striegel and G. Manimaran, "A Survey of QoS Multicasting Issues," *IEEE Communications Magazine*, vol. 40, no. 6, Jun 2002.
[32] B. Raman and R. H. Katz, "An Architecture for Highly Available Wide-Area Service Composition," *Computer Communications Journal (special issue on "Recent Advances in Communication Networking")*, May 2003.
[33] E. Zegura, K. Calvert, S. Bhattacharjee, "How to Model an Internetwork," in *Proc. of IEEE INCOFOM*, Apr 1996.