On Exploring Performance Optimizations in Web Service Composition

Jingwen Jin and Klara Nahrstedt

Department of Computer Science
University of Illinois at Urbana-Champaign, USA
{jjin1, klara}@cs.uiuc.edu

Abstract. The importance of service composition has been widely recognized in the Internet research community due to its high flexibility in allowing development of customized applications from primitive services in a plug-and-play manner. Although much research in defining architectures, choreography languages and etc, has been conducted, little attention has been paid to composite services' runtime performance-related aspects (e.g., network bandwidths, path delay, machine resources), which are of great importance to wide-area applications, especially those that are resource-consuming. Service composition in the wide area actually creates a new type of routing problem which we call *QoS service routing.* We study this problem in large networks and provide distributed and scalable routing solutions with various optimization goals. Most importantly, we propose ways to reduce redundancies in data delivery and service execution through explorations of different types of multicast (service multicast and data multicast) in one-to-many application scenarios.
keywords: service composition, QoS, multicast, application-level routing, overlay networks

1 Introduction

The Internet has long been recognized as an environment with heterogeneity everywhere and in every aspect, and this heterogeneity problem has been further exacerbated with the increasing popularity of small devices using wireless connections 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 etc) communicating over networks of different 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 created for such purposes [1, 2]. However, given the range of diversity involved in the Internet, developing monolithic transformational services to bridge all conceivable end-to-end heterogeneities would be some task that requires tremendous amount of effort, if not totally impossible.

Fortunately, the *component service* model, which allows complex services to be dynamically and rapidly aggregated from primitive ones, has been proposed

and started to be adopted in the Internet (e.g., the Web and peer-to-peer networks) for service flexibility and reusability [3–5]. This new, flexible service model has triggered many interesting and useful Internet applications. Imagine a mobile phone user that wants to retrieve the content of a Web document written in Latin and hear it through speech 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*.

At the service deployment time, for the sake of robustness, each service needs to be replicated in multiple network locations (i.e., have multiple instances). Service composition should happen at the runtime, and it is desirable to select service instances based on current network and machine conditions, so that the service path not only meets service functional requirements, but also satisfies certain performance requirements (e.g., ensuring that there is sufficient network bandwidth between the output and input of every pair of consecutive components). Since service composition includes a broad range of issues (e.g., architecture, language standard), we create and use the terminology *service routing* for focus on functional correctness and performance aspects involved during the run time of service composition. We will assume Web services are deployed at *proxies* (be them regular caching proxies or dedicated application-specific proxies).

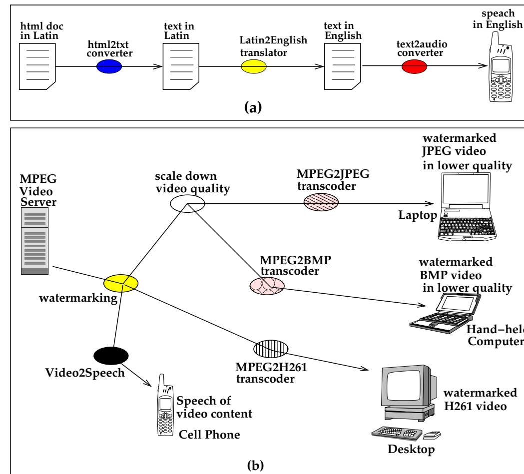

Fig. 1. Two Web scenarios that make use of composite services: (a) A mobile phone user retrieves a Web document written in Latin and hears it through speech in English; (b) news video from CNN or Yahoo server is customized within a service network according to end users' network and machine capacities. When there are multiple end users interested in receiving the same source data, service multicast can be employed for resource optimization purposes.

Interesting composite services can be also useful in one-to-many application scenarios. Imagine the Web news 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 have end-to-end service paths individually built, such a unicast delivery model may incur waste of bandwidths (due to redundancies in data delivery) and machine resources (due to redundancies in service execution). We propose to build a single service tree, rather than multiple independent service paths, through which the data should be delivered 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. To differentiate the two delivery modes, hereafter we will use the terminologies *service unicast (routing)* and *service multicast (routing)* for service routing in one-to-one and one-to-many scenarios, respectively.

For composite services to be widely acceptable and useful, automating the service routing process at the middleware layer has become critical to enable seamless provisioning of integrated services at the application layer despite the fact that an integrate service might be actually distributed over multiple hosts in wide-area networks. Service unicast routing has been reasonably addressed in the literature [6–9]. Some of the existing work, e.g., [8, 9], adopt a global planning approach which, concerning its limited scalability, is not suitable for the current Web. Scalable routing falls into two approaches: hierarchical [6] and distributed [7], each with its own advantages and disadvantages. The routing approach to be adopted in this paper follows the latter category, because distributed routing based on on-line probing involves with more updated (thus more accurate) routing state. In the unicast context, a distributed solution based on local heuristics has been described in [7]. However, the local optimality alone often will incur long service paths. We remedy this shortcoming by using the geometric information of the network hosts as guidance to compute more delay-efficient paths.

Our major focus would 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. While source-based (pure) service multicast has been proposed and studied in [10, 11] for small service networks, in this paper, we consider the problem in the current Web scale. In such a large scale, centralized planning is certainly not a viable solution, 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. Moreover, we propose to further optimize resource usages by integrating data multicast into service multicast. We call such a combined multicast delivery mode *hybrid multicast*.

The remainder of this paper will be structured as follows. We first describe some background and related work in Section 2, followed by the foundation of our solution design in Section 3. We present our distributed solutions for service unicast, pure service multicast, and hybrid multicast in Sections 4, 5, and 6, respectively. The solutions are implemented in the well-known network

simulator *ns-2* and in Section 7 we provide some performance results. Section 8 gives some concluding remarks of this paper as well as directions for our future research.

2 Background and Related Work

To realize service composition in the Internet, many important issues need to be addressed. (1) **service description:** When a developed service component is to be deployed, it needs to be described by 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. (2) **service discovery:** 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. Scalable ways of performing service discovery have been also investigated in peer-to-peer networks [12, 13]. (3) **service request compilation:** At the application design or run time, given service specifications of two communicating ends, it needs to be further verified which service components are to be composed and in which order, i.e., to obtain a compositional service model or a *service request*[1]. Research in this area can be found in [14, 5].

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 routing*, that resides above these tasks and that can choose appropriate service instances (returned by a discovery system) for the basic components in a service request (returned by a QoS compiler), so that users at the application layer will see the application as an integrated service, rather than separate components (Figure 2).

A sample service request is shown in Figure 3. The functional part of a service request will be denoted as $r = (p_s, s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots, p_d)$, which is to find a service path between the source p_s and the destination p_d containing s_1, s_2, and s_3, in sequence. A service path will be denoted as $sp = (p_s \rightarrow s_1/p_\alpha \rightarrow s_2/p_\beta \rightarrow s_3/p_\gamma \rightarrow \ldots \rightarrow p_d)$, where s_i/p_θ means service s_i is provided by proxy p_j (mapping of a service onto a proxy). Note that different from the traditional data routing, where paths should be loop-free, in service routing, data loops are allowed, in the sense that a single network node is allowed to be visited multiple times in case it is capable of serving multiple (either consecutive or inconsecutive) services in the request. Therefore, when we refer to "a service node", it means mapping of a service onto a proxy (s_i/p_θ). 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 n to be an instance of n's preceding service in the request.

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

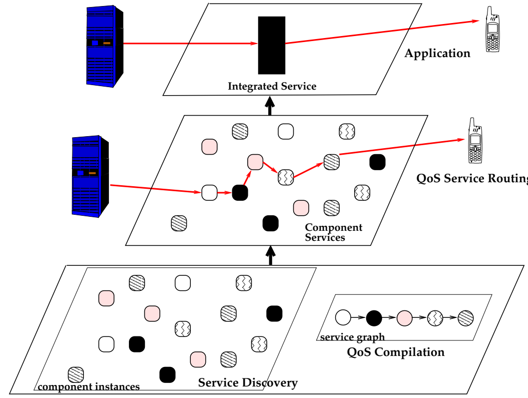

Fig. 2. The *service routing* substrate is resided between the application layer and the service discovery/QoS compilation layer to make component services transparent to the application layer.

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.

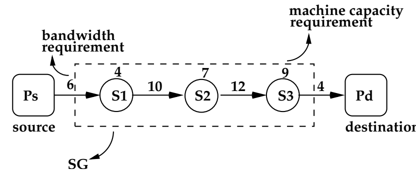

Fig. 3. A service request with linear service graph (SG): from the *source* to the *destination*, locate a QoS-satisfied path that encompasses $s_1 \rightarrow s_2 \rightarrow s_3$ in sequence.

Service unicast has been investigated extensively in different domains (e.g, Web, peer-to-peer networks, or company networks) and in different levels of the network (e.g., physical network level or overlay network level). Depending on the size of the network, computations of service routing can be performed in different ways, e.g., centralized or distributed. In [8, 9], a single network node is required to maintain the global routing state (QoS and service availability information) of the network, so that computation of service paths can be performed locally. However, such an approach does not scale because the associated state maintenance overhead increases quickly with the network size. A remedial step for increasing scalability is to introduce hierarchies into the network, so that topology abstraction and state information become possible to significantly reduce the state maintenance overhead. A hierarchical solution was developed in [6]. Alternatively, scalable service routing can take a distributed approach by having the network nodes maintaining state information of a limited neighbor-

hood. [7] describes a distributed, hop-by-hop approach whose routing decision is based on local heuristics.

Service multicast was proposed in our previous work [10], and two algorithms for building service trees have been devised and their performances compared. However, since both the construction and the maintenance of service trees take a source-based approach, the solution is suitable only for small-scale networks. A source-based approach is simple, and allows service trees to be computed quickly, and usually path/tree optimizations are better achieved. However, due to the rapidly increasing routing state maintenance overhead with the network size, scalability is constrained.

Overlay network routing can be performed either on top of structured topologies [15, 10] or on top of unstructured topologies [16, 9]. The former approach views the overlay network topology as a partial mesh, so that routing protocols (such as OSPF and MOSPF) designed for the IP layer 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.

3 Foundation of Our Solution Design

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 distributed routing decisions also hard if communication delay is a concern. We address this weakness by associating each Internet host with geometric coordinates and using it to estimate Internet distances (communication delays) between hosts. The relative geometric coordinates of a machine can be automatically assigned by the method described in [17] and, as will be clear later, the added geometric location information will serve us as guidance in finding more delay-efficient service paths/trees.

To maximize path performances at the overlay layer, in this paper we do not set network topology constraints (i.e., the initial network is a fully connected, unstructured topology), and a service path/tree is built for each application scenario. However, while service paths/trees are built on top of an unstructured overlay topology, another structured mesh topology is maintained for general control messages. 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. This design choice is similar to that of YOID [16]. For communication efficiency, we connect the overlay network nodes into a Delaunay triangulation [18], because Delaunay triangulation is a spanner graph that possesses some nice properties: a path found within a Delaunay triangulation has length bound by a constant times the straight-line distance between the endpoints of the path. By using such a geometric topology, control messages can be routed by using an on-line routing method, such as the greedy approach or compass routing approach [19].

In a large, unstructured service overlay network where service neighbors are not defined until the runtime, we do on-line probing of the service instances' resource conditions (e.g., bandwidth and machine capacity) to identify the best next service hop according to the request, instead of maintaining routing state. By distributed, we mean not only the construction of service paths/branches, but also the maintenance of multicast group tree information, will be performed distributively. In [10], the functional service tree is centrally maintained at the root. Thus every join request had to go to the root to learn its functionally graftable service nodes. Such a centralized approach introduces both a single point of failure and a bottleneck. In this research, the functional service tree will be maintained by all on-tree nodes, so that each of them can individually look for graftable service nodes for other join requests (more details later).

4 Service Unicast

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 [20]. While 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 distributed QoS path finding so that the computed QoS-satisfied path is not unnecessarily long, in service routing, due to the complex dependency relations among services, no similar shortest *service* paths can be maintained as to allow a node to quickly lookup for the *best* next service hop[2].

Lacking maintenance support, next service hop needs to be discovered at the runtime. Specifically, starting from the source, we gradually add to the path 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 service discovery system. A service path can be thus 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 best one according to its selection criteria.

However, existing solutions in unicast QoS service routing that follow the distributed approach are not satisfactory. For example, in [7], selection of next service hop is solely based on local heuristics, where the next service hop is the one whose aggregate value of available bandwidth, machine resources and machine's up time is optimum. The local heuristics alone, however, would only help balance the network and machine loads and potentially optimize the path's overall concave or multiplicative metrics (e.g., the path's bottleneck bandwidth or robustness), but would not pose any constraint on the overall service path length, which is an additive metric that requires global optimizations. As a consequence, service paths computed hop-by-hop by adopting local heuristics

[2] By *best*, we mean the QoS-satisfied service instance that leads to a shortest service path.

tend to be long, and inevitably consume more network resources. We will name this approach *local resource-amplest (LRA)* approach.

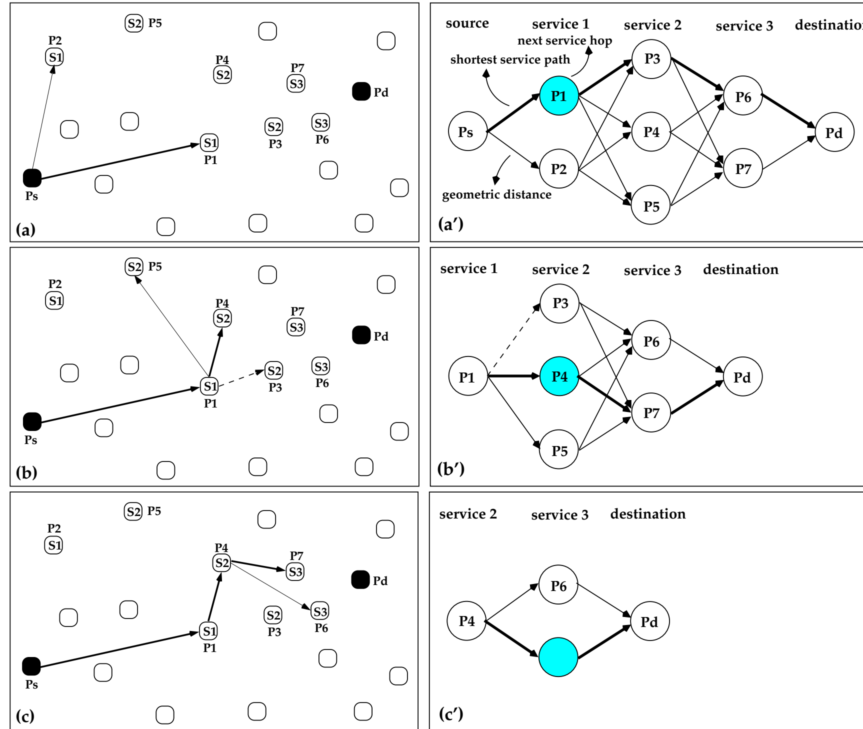

Fig. 4. Finding a QoS-satisfied and potentially shortest service path hop-by-hop from p_s to p_d that satisfies the service graph $s_1 \rightarrow s_2 \rightarrow s_3$.

The weakness of the *LRA* approach can be remedied by using the hosts' geometric location information as guidance when performing the hop-by-hop routing computation. At this point, let us temporarily ignore the load balancing issue, and concentrate on path delays. We first describe how a QoS-satisfied delay-efficient service path can be computed. In Section 3, we mentioned Internet hosts can obtain their geometric information as described in [17]. Such information can then be easily incorporated into a service discovery system, so that the discovery system is able to tell also the locations of the service instances. With this location information, we resolve the service path in a hop-by-hop manner as follows. Each hop sends QoS probe messages to all of its service neighbors, and then among the instances that satisfy all resource requirements, the current hop will select the one that potentially leads to the most delay-efficient service path as the next service hop, by doing some extra computation as shown in Figure 4.

Figure 4 depicts a case where we want to find a path, between the source p_s and the destination p_d, in which services s_1, s_2, and s_3 are to be included in sequence. In Figure 4(a), starting from the source, p_s probes resource conditions of both instances of next service in the request, s_1/p_1 and s_1/p_2. Resource conditions in this case may be available bandwidths from p_s to p_1 and from p_s to p_2, and p_1 and p_2's available machine capacity. Assuming both instances have sufficient resources, p_s chooses the one that potentially leads to a shorter service path. This can be computed as shown in Figure 4(a'): by deriving the correspondent service DAG (Directed Acyclic Graph) based on the service request and service instances' availability (returned by a discovery system), and applying a shortest paths algorithm [21] on top of it, a shortest service path (shown in bold lines) can be calculated, and after which next service hop (shown in shadow) that optimizes the overall path length is chosen. In Figure 4(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 4(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 are resolved. We name such an approach *GLG*, which stands for *geometric location guided.*

Since *LRA* and *GLG* are intended for individual optimization goals (load balancing and delay respectively), it can be predicted each one will perform poorly in terms of the non-optimized metric. For example, *LRA* would have poor performance in terms of delay, and likewise, *GLG* would perform poorly in terms of load balancing. If we are to consider both metrics at the same time, then combining *LRA* and *GLG* would be necessary. Two derivatives exist: *LRA-GLG* and *GLG-LRA*. In the first one, each hop first identifies the next service hops that potentially lead to shortest service paths, and then among them, it makes its selection based on their resource conditions. As an example, if at a network node p, p detects that both $sp_1 = (p \rightarrow s_1/p_\alpha \rightarrow \ldots \rightarrow p_d)$ and $sp_2 = (p \rightarrow s_1/p_\beta \rightarrow \ldots \rightarrow p_d)$ are two potential shortest service paths, then p may decide which one to go, either p_α or p_β, based on p_α and p_β's resource conditions. The second derivative, *LRA-GLG*, is different from *GLG-LRA* just in the order of application of two routing features.

5 Pure Service Multicast

When a multimedia stream is delivered to a group of users that require different transformational rules on the stream, then instead of having the stream transformed and delivered through multiple independent service paths, a more efficient way is to construct a service multicast tree for the transformation and delivery purposes. To support the dynamic membership feature of many multimedia applications, we take an incremental approach for service multicast tree

building, which means that one service path/branch is constructed at a time to cover the newly joining member.

A key issue in multicast tree building is the *graftable on-tree node* concept. For example, in the PIM protocol, a newly joining member m's request is forwarded towards the source. If the request hits some on-tree node n before reaching the source, then n is said to be the *graftable on-tree node* for m, and a branch starting from n and ending at m is usually constructed to cover 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, due to the functionality issues, not all on-tree nodes functionally qualify as graftable nodes for other joining members. Rather, 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 in sp qualify as functionally graftable service node for r: p_s, s_1/p_α, s_2/p_β, and s_3/p_γ. To maximize service sharing, we use the *longest match* (prefix) [10] 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 the longest prefix of sp and r is $prefix = s_1 \rightarrow s_2 \rightarrow s_3$ and *prefix*'s last service s_3 is mapped onto p_γ.

Construction of our service multicast tree will take the following procedures (an example will be shown later in the section). Each member joining the multicast group would send its request r towards the source through the organized overlay network topology (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 locally maintained functional service tree T_f (maintenance of T_f will be discussed further later) to identify the best functionally graftable service node n, and forwards the request accordingly. Between n and m, a service branch can be constructed hop-by-hop by using a unicast service routing solution described in Section 4. Note that with a prefix of r satisfied by the found graftable node, we only need to find a service branch for the suffix of r between n and m.

We now briefly describe the tree maintenance issue. In data multicast, routers express their join/leave interests through IGMP (Internet Group Management Protocol), and since a router has one single function (to forward data as is), it basically needs to be only aware of its children in the multicast tree. However, the similar does not hold in service multicast due to service functionality constraints. Rather, to enable an on-tree node to identify graftable service nodes for others, it needs to keep 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

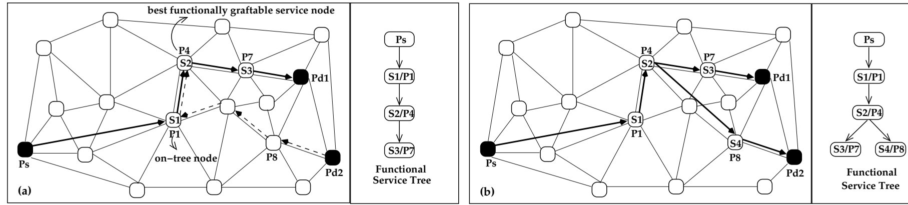

Fig. 5. (a) A service request message is sent from the newly joining member p_{d2} towards the source by using compass routing, and the request hit an on-tree node p_1 before it reaches p_s. Since every on-tree node maintains T_f, p_1 found that p_4 is the best graftable node for the current request, thus the request is forwarded to p_4; (b) a service branch is established hop-by-hop from the graftable node p_4 to p_{d2}.

adequate control messages. Although because of the possible loop issue in service routing, a single proxy may appear in multiple positions of a functional service tree, only one copy of the tree needs to be maintained *per proxy*.

Figure 5 depicts an example of how a pure service multicast tree is built and maintained. In Figure 5(a), assume p_{d1} is the first member who joins the group. After p_{d1} has joined the group, the on-tree proxy nodes p_s, p_1, p_4, p_7, and p_{d1} will maintain a functional service tree T_f depicted at the right side of Figure 5(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, and 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 functionally graftable node for the current request, thus the request is forwarded to p_4. In Figure 5(b), a service branch is established hop-by-hop from the graftable node p_4 to p_{d2}. Since the graftable node p_4 already satisfied a prefix of r_2, only the correspondent suffix needs to be satisfied by the service branch from p_4 to p_{d2}. After p_{d2} joins, the functional service tree T_f maintained by all on-tree nodes becomes that on the right-side figure of Figure 5(b). Note that T_f only needs to be updated if the functional aspects of the tree have been modified. 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, and the functional service tree remains unchanged.

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 usages compared to service unicast, as in most of the 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.

6 Hybrid Multicast

In pure service multicast, each service branch gets directly attached to its best functionally graftable node. However, in such an approach, bandwidth usage may not have been optimized. An example is illustrated in Figure 6(a): the

proxy offering 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. This may cause data delivery in those sub-groups to be sub-optimal. First, it may be 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 this point.

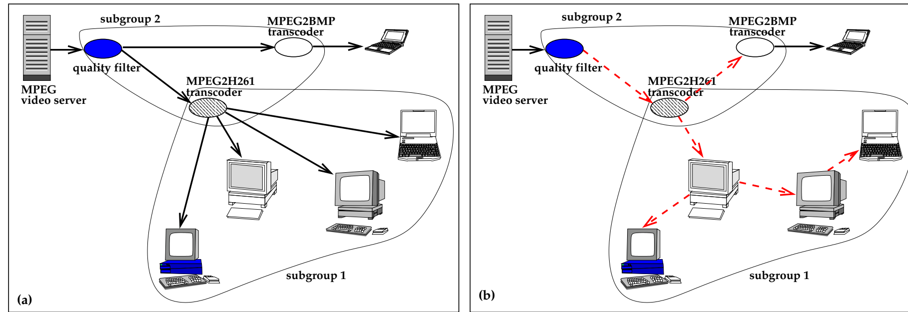

Fig. 6. (a) Pure service multicasting; (b) hybrid multicasting (service multicasting + data multicasting).

We address these weaknesses by further employing data multicast in the local sub-groups. Although IP-layer multicast would be an option, in this research, we will only exploit data multicast at the application layer because the real deployment of multicast at the IP layer has been hindered by its need of change in the infrastructural layers. Two feasible application-layer data multicast trees (for subgroups 1 and 2) can be built as shown in Figure 6(b). In addition to boosting the overall cost efficiency of the service tree, exploring data multicast would also increase the success rate in finding QoS service branches when resources are scarce.

To realize such a hybrid multicast scenario, the distributed approach requires each on-tree proxy and/or service node to keep two trees: one for the global functional service tree, and the other for local data distribution tree, which we denote as T_f and T_d respectively. 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 relay of data. The same as in pure 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.

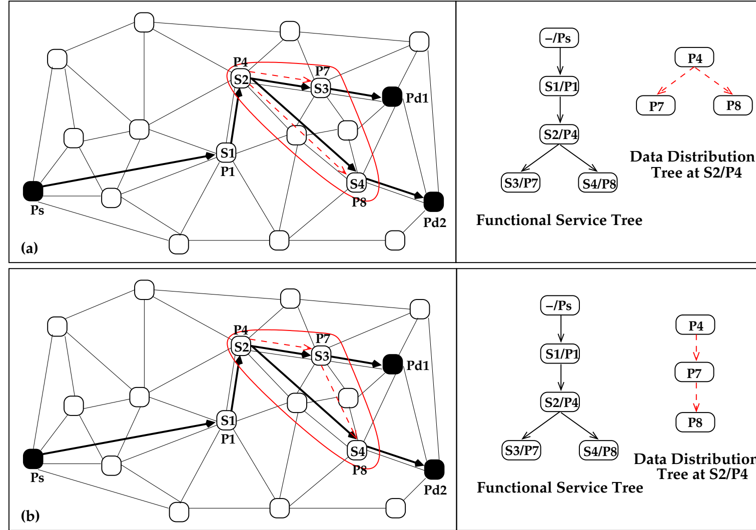

Fig. 7. Exploring data multicast in a service multicast scenario: (a) a new service branch's first node, p_8, is initially directly attached to the graftable service node p_4 (p_4 as p_8' parent in the local data distribution tree); (b) p_8 gets parent-switched to p_7 in the data distribution tree.

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 nodes which are closer in an increasing order of distance. Upon receiving the list, n' starts to probe the listed nodes for the bandwidth conditions one by one in an increasing order of distance. Once it finds a node whose outbound bandwidth to n' is sufficient for supporting the data stream, n' 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 different 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 (one per each service it offers).

Figure 7 depicts what the global functional service tree and the local data distribution tree would look like in the scenarios. In Figure 7(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 7(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 7(b).

It is clear that with the employment of local data multicast, end-to-end service paths may become longer than in pure service multicast. However, such a performance degradation is justified by savings on network bandwidths.

7 Performance Study

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

7.1 Evaluation Methodology

Our physical Internet topologies are generated by the *transit-stub* model [22], 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 randomly assigned by some 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 are chosen to be the landmark nodes - L, based on which the proxies can derive their coordinates in the geometric space defined by L[17]. We use planar geometric spaces in our simulations, and 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 in the evaluations:

- *Link Utilization*: is the ratio of used bandwidth to the total initial bandwidth of the physical network links that measures how much the physical links are loaded. The ratio may range between 0 to 1: at 0, the physical link has zero load; at 1, the link is fully loaded.
- *Proxy Utilization*: is the ratio of amount of machine resources in use to the machine's total initial 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).

- *Service Path Length*: is the sum of individual virtual link lengths that make up the service path, where the virtual link lengths are end-to-end delays.
- *Delay* $\times$ *Bandwidth Product*: The purpose of this metric is to measure the volume that the 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.

7.2 Performances of Different Service Unicast Approaches

In this section, we measure performances of the different service unicast approaches (*GLG*, *LRA*, *GLG-LRA*, and *LRA-GLG*) described in Section 4 in terms of all listed performance metrics.

The simulation settings for these tests are as follows. The physical networks contain 300 nodes, and among them, 10 are landmarks and 250 are proxies. We randomly generated 1000 service path 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, where late join requests may be rejected because of resource scarcity. For each scenario, we run two test cases.

Sufficient-resource settings: In this case, since all service requests get successfully admitted, the performance metrics of interest are *link utilization*, *proxy utilization*, *service path length*, and *delay bandwidth product*. Figure 8 (a) and (b) show the physical network link and proxy utilization of the different approaches. As has been predicted, since *GLG* genuinely seeks shortest QoS-satisfied service paths, load balancing in both respects is poor. This is indicated by the fact that the *GLG* curves are steepest. *LRA* 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, *LRA* performs poorly in terms of *service path length* (Figure 8 (c)) and *delay bandwidth product* (Figure 8 (d)), because service paths computed by *LRA* are long, and therefore demand more network resources. However, this time *GLG* performs best, because service paths computed by this approach tend to be short, and as such, require less network resources. *GLG-LRA*'s performances are quite close to those of *LRA*, and *LRA-GLG* has the best performances in these two respects.

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 8(e) shows the path finding success rates of the different service unicast approaches. As has been expected, since *GLG* does not take load balancing into consideration, certain resources

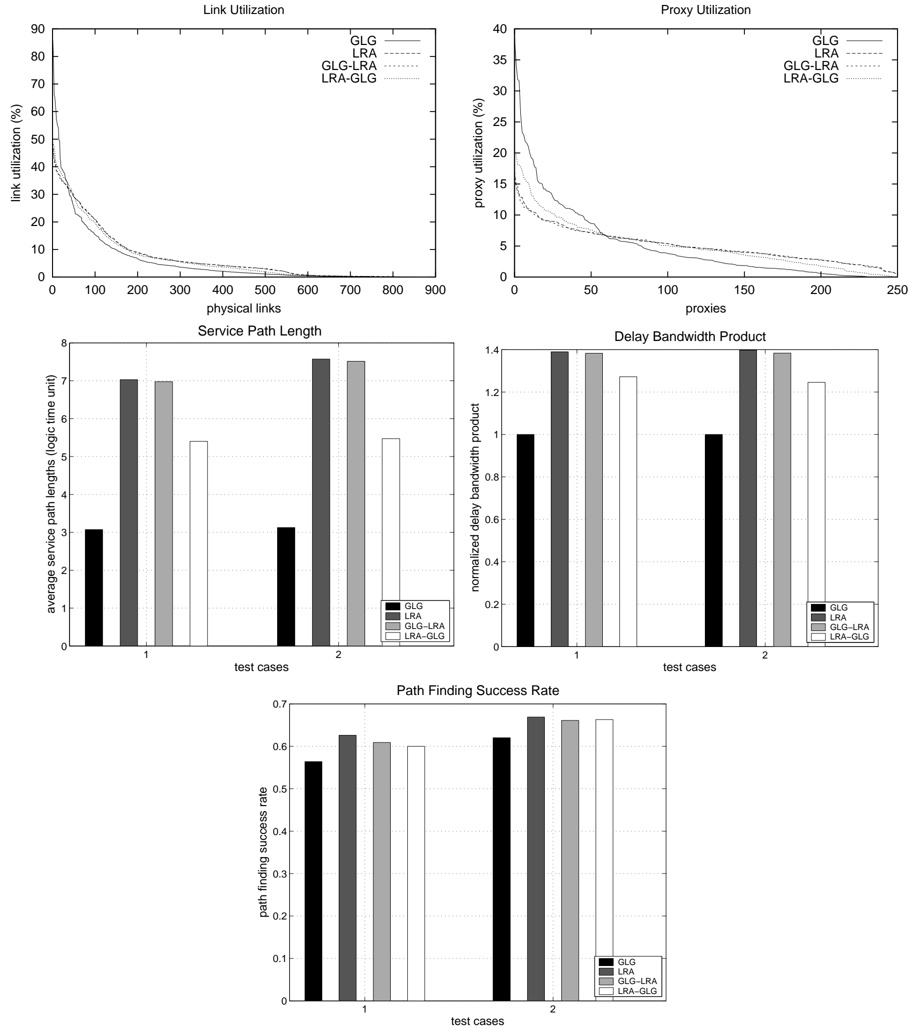

Fig. 8. Comparisons of: (a) *physical link utilization*; (b) *proxy utilization*; (c) *service path length*; (d) *delay bandwidth product*; and (e) *path finding success rate* among the different service unicast approaches.

may become exhausted more quickly than other approaches that consider load balancing, and as a consequence, *path finding success rate* was lowest in *GLG*.

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*; *LRA* 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. *LRA-GLG* seems to have best balanced these contradictory factors, as it incurs relatively short service paths while maintaining a high path finding success rate.

7.3 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 *LRA-GLG* is the best service unicast approach that balances load and optimizes path lengths at the same time, constructions of multicast tree branches adopt *LRA-GLG* when selecting service hops. The two multicast approaches, pure service multicast and hybrid multicast, will be compared against the corresponding service unicast solution, which is unicast *LRA-GLG*.

Sufficient-resource settings: Simulations are run for multicast group size of 100, where service requests are drawn from a pool of size 20. As we can see from Figure 9 (a), hybrid multicast yields better bandwidth (link) utilization than pure service multicast. However, there is not too much difference in proxy utilization between pure service multicast and hybrid multicast (Figure 9 (b)). This is expected, because local data multicast does not further diminish the number of service executions. Compared to service unicast, both types of multicast incur longer end-to-end service paths (Figure 9 (c)), but less total tree lengths (Figure 9 (d)) due to service path sharing. Not surprisingly, the two multicast cases yield tremendous delay bandwidth product savings compared to unicast (Figure 9 (e)). While it is intuitive that advantages of hybrid multicast over service multicast (and multicast over unicast) increase with the multicast group size, it would be interesting to quantify the gains in the future work.

Insufficient-resource settings: Since the major purpose of designing hybrid multicast was to make even better network bandwidth usage than pure service multicast, in this test, we only make bandwidth scarce. As Figure 9 (f) shows, service path finding success rate increases dramatically from unicast to the two cases of service multicast, and hybrid multicast over-performs pure service multicast.

8 Conclusions

In this paper, we have explored performance optimizations in Web service composition in several respects. First, we made an improvement over an existing hop-by-hop service unicast solution - *LRA* - that makes routing decisions based

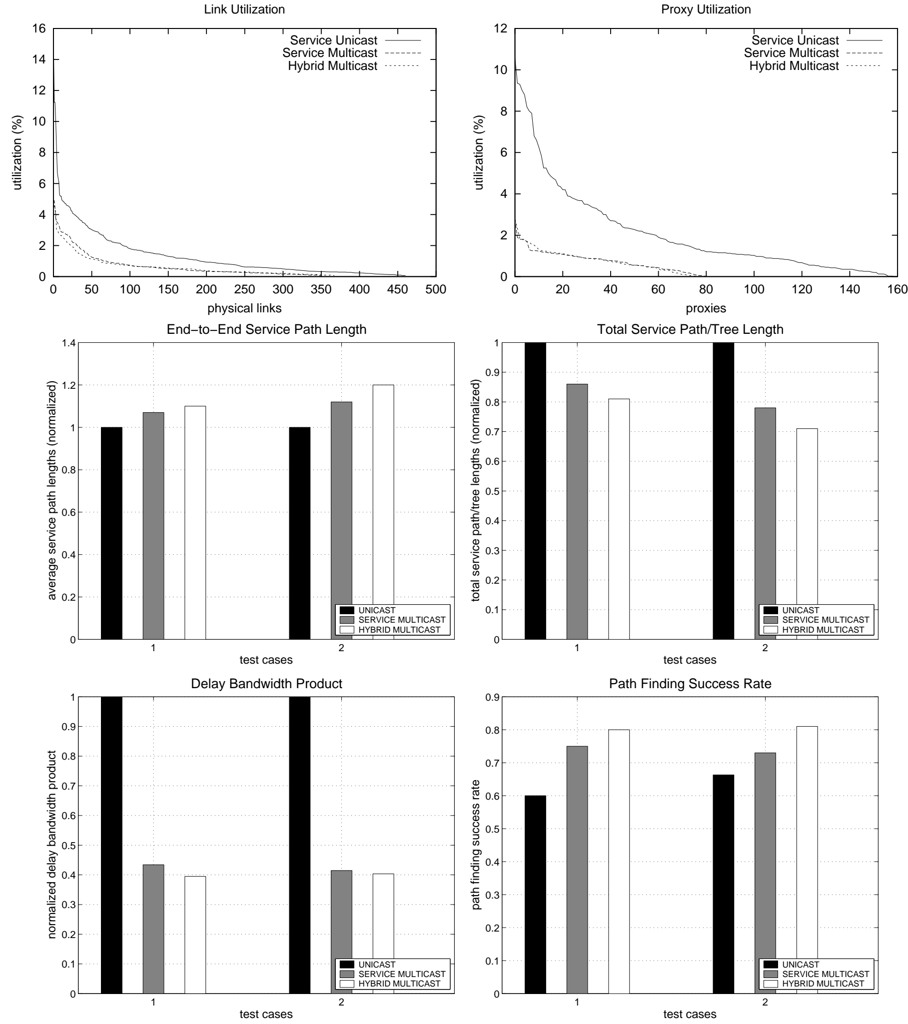

Fig. 9. Comparisons of: (a) *physical link utilization*; (b) *proxy utilization*; (c) *end-to-end service path length*; (d) *total service path/tree length*; (e) *delay bandwidth product*; and (f) *path finding success rate* among the different delivery modes: *service unicast*, *pure service multicast*, and *hybrid multicast*.

on local heuristics only, by introducing and using the geometric location information of the Internet hosts. The geometric location guidance (*GLG*) can significantly reduce service path lengths compared to *LRA*. We studied different combinations of *GLG* and *LRA* in terms of several performance aspects. The simulation performances showed that *LRA-GLG* best balances the tradeoffs. Second, the paper proposed a fully distributed approach for incrementally building service multicast trees, by identifying and solving several key differences (e.g., graftable node, tree maintenance) between service multicast and the conventional data multicast. Advantages of pure service multicast over service unicast were also verified through simulations. Third, We proposed to further explore benefits of data multicast inside service multicast scenarios, and provided a hybrid multicast solution. We showed how this can be realized, and by how much hybrid multicast can outperform pure service multicast.

Due to space limitations, failure recovery issues have been left out. Recovery operations are called for when a physical node or link fails. Since loops are allowed in service routing, failure of a single physical node may trigger failures of several points in the service path/tree. As an example, assume a single service path $sp = (p_s \rightarrow s_1/p_\alpha \rightarrow s_2/p_\beta \rightarrow s_3/p_\alpha \rightarrow \ldots \rightarrow p_d)$, if p_α fails, then two service nodes (s_1/p_α and s_3/p_α) need to be repaired. Failure recovery is even more complex in multicast scenarios; special failure detection and recovery mechanisms will be needed. We plan to address these issues in our future work.

9 Acknowledgments

This material is based upon work supported by NSF (under Awards No. CCR-9988199 and EIA 99-72884 EQ) and NASA (under Award No. NAG2-1406). Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the awarding agencies. The authors would like to thank the anonymous reviewers for their helpful comments.

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*, 18(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. Jingwen Jin and Klara Nahrstedt. Large-Scale Service Overlay Networking with Distance-Based Clustering. In *ACM/IFIP/USENIX International Middleware Conference (Middleware2003)*, Rio de Janeiro, Brazil, Jun 2003.
7. 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.
8. Sumi Choi, Jonathan Turner, and Tilman Wolf. Configuring Sessions in Programmable Networks. In *Proc. of IEEE INFOCOM*, Anchorage, Alaska, Apr 2001.
9. 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.
10. Jingwen Jin and Klara Nahrstedt. On Construction of Service Multicast Trees. In *Proc. of IEEE International Conference on Communications (ICC2003)*, Anchorage, Alaska, May 2003.
11. 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.
12. 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.
13. 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.
14. Duangdao Wichadakul. *Q-Compiler: Meta-Data QoS-Aware Programming and Compilation Framework.* PhD thesis, Computer Science Department, University of Illinois at Urbana Champaign, Jan. 2003.
15. Y. Chu, S. G. Rao and H. Zhang. A Case For End System Multicast. In *Proc. of ACM SIGMETRICS*, pages 1–12, Santa Clara, CA, Jun 2000.
16. Paul Francis. Yoid: Extending the Internet Multicast Architecture, Apr 2000.
17. T. S. Eugene Ng, Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of IEEE INFOCOM*, New York, NY, Jun 2002.
18. 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.
19. Evangelos Kranakis, Harvinder Singh, Jorge Urrutia. Compass Routing on Geometric Networks. In *Proc. of the 11th Canadian Conference on Computational Geometry*, Vancouver, CA.
20. Shigang Chen, Klara Nahrstedt, Yuval Shavitt. A QoS-Aware Multicast Routing Protocol. *IEEE Journal on Special Areas in Communication*, 18(12):2580–2592, Dec 2000.
21. Jingwen Jin, Klara Nahrstedt. QoS Service Routing for Supporting Multimedia Applications. Technical Report UIUCDCS-R-2002-2303/UILU-ENG-2002-1746, Department of Computer Science, University of Illinois at Urbana-Champaign, USA, Nov 2002.
22. E. Zegura, K. Calvert, S. Bhattacharjee. How to Model an Internetwork. In *Proc. of IEEE INCOFOM*, Apr 1996.