Large-Scale Service Overlay Networking with Distance-Based Clustering

Jingwen Jin and Klara Nahrstedt*

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

Abstract. The problem of service routing (or dynamic service composition) has recently emerged as a consequence of the distributed composable services model residing in middleware layer(s). However, existing solutions are mostly suitable for small- or medium-scale service overlay networks, as service routing is performed over flat overlay topologies such as a mesh. Due to their increasing routing information maintenance costs, these flat (single-level) topology solutions cannot cope with large-scale service overlay networking. For better scalability, in this paper, we provide a hierarchical service routing framework, which comprises three parts. In the first part, we organize the overlay network nodes into clusters based on their Internet distances. We then construct a hierarchically fully connected (HFC) topology based on the clustering result. In such a topology, nodes within a cluster are considered fully connected, and the clusters themselves are also fully connected by their border nodes. In the second part, a hierarchical state information distribution protocol will be provided so that each node in the system maintains full state of the nodes in its own cluster and aggregate state of other clusters in the system. In the third part, we present how service paths can be computed hierarchically in a divide-and-conquer fashion. Through simulation tests, we demonstrate that while achieving much better scalability, our framework provides also as good and efficient service paths as single-level mesh solutions.

Keywords: service routing, dynamic service composition, clustering, topology aggregation, hierarchical routing

1 Introduction

The concept of overlay networking has recently been brought up for supporting various kinds of middleware and application services. However, most of the existing work, such as [1,2,3,4,5,6], targets at small- or medium-scale overlay networks, as the constructed overlay topologies are mostly meshes of single level.

* This work was supported by the DARPA grant, F30602-97-2-0121, NSF grants CCR-9988199 and EIA 99-72884 EQ, and NASA grant NAG2-1406. Jingwen Jin is also supported by CAPES/Brazil.

These single-level, flat network organizations do not scale, because of their increasing routing information maintenance cost. To achieve scalability, in this paper, we introduce a middleware framework for hierarchical service routing that reduces routing information maintenance cost by means of topology abstraction and state information aggregation. In correspondence to these mechanisms, service routing will also be done hierarchically. Assuming that each node in the system maintains partial global state (full state of the nodes in its own cluster and aggregate state of other clusters in the system), hierarchical routing shares advantages of source routing and distributed routing [7].

Although hierarchical (data) routing at the network layer has been extensively studied in the literature [8,9,10], hierarchical service routing in distributed, overlay service networks (middleware and application layer) presents several additional challenges that need special efforts. (1) While at the network layer, clusters are defined manually by humans based on certain properties of the network (e.g., location, administrative domain, or connectivity), an overlay service network has a virtual, fully-connected topology[1]. Hence the first problem is *how to identify virtual clusters and how to define connectivity between nodes and clusters.* (2) In a clustered service overlay network, the second problem is *how service routing information should be aggregated and distributed.* (3) Compared to data routing, service routing exhibits two new constraints: service functionality and service dependency, that make traditional data routing solutions inapplicable [11]. While solutions such as [11,12,5] exist for service routing over flat, single-level topologies, *how can a single-level topology solution be extended to solve the hierarchical service routing problem*?

We address these challenges as follows. For the first problem, as delay (proximity) will be used as a performance metric when seeking service paths, we will use proximity as a metric for clustering services. Since services are located in proxies, the distance between a pair of services can be represented by the distance between the proxies in which the services are located. Thus the problem of clustering services by their proximity amounts to the problem of clustering proxies by their proximity. To achieve this goal, a combination of mechanisms (for distance map obtainment and clustering) will be adopted in this paper, so that the clustered overlay proxy network is congruent with the underlying physical network. Once the clusters (of proxies) have been detected, we will create the topology in such a way that nodes within a single cluster are fully connected, and the clusters themselves are also fully connected by their border nodes. Therefore, any pair of intra-cluster nodes can communicate to each other directly, and any pair of inter-cluster nodes can communicate to each other via their border nodes. We name such a topology an HFC (Hierarchically Fully-Connected) topology. Figure 1 depicts an example of a bi-level HFC topology. This topology design choice stems from the following observation. Service routing exhibits some different features (e.g., service functionality and service dependency) than data routing that make the use of partial mesh topologies, which is suitable for

[1] An overlay service network creates a virtual, fully-connected topology with nodes, called proxies, carrying value-added middleware or application services.

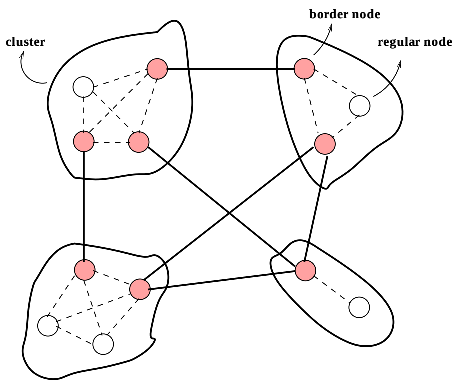

Fig. 1. An example of HFC topology: nodes within each cluster are fully connected, and clusters are fully connected by their border nodes (in shadow).

data routing, not as suitable. This is so because, in data routing, nodes simply participate as relays, and pass data as is to their neighboring nodes; however, in service routing, as the data traverses, nodes may be required to process it differently using different services. Since how services are to be composed (the service dependency issue) is mostly resolved at the runtime, two runtime-defined neighboring services may appear to be several nodes apart in a statically configured partial mesh. In other words, meshes configured statically do not reflect well service dependency needs that are resolved at execution time. In fact, highest efficiency in service paths can be only achieved if the overlay network is considered fully connected (assuming the underlying physical network does not partition). We thereby try to make the hierarchical topology as fully connected as possible. While large networks cannot afford full topologies, after having done proximity-based clustering, small groups of nearby nodes will afford to be fully connected. In a bi-level HFC hierarchy, two nodes (thus also the services installed on them) are at most two nodes away. By limiting the number of hops between any pair of services will potentially increase service path efficiencies.

For the second problem, because service routing involves functionality as the basic requirement, and certain performance metrics for optimization purposes, we will need to maintain two pieces of information: service functionality (or capability) of the nodes and performance status of single or aggregated nodes/links, by using a hierarchical state information distribution protocol. Topology aggregation with QoS parameters has been studied in [9,13]. We need to further solve the service capability information (SCI) aggregation problem. Assuming that each service can be uniquely named, and a single proxy's SCI is represented as a set of service names, we can aggregate SCI of a group of proxies as the union of their individual SCI sets. State information will be further discussed in Section 4. In this paper, we will only consider delay as our service path performance metric, QoS parameters such as bandwidth, machine capacity, and machine volatility [11], will not be considered at this point.

For the third problem, we will adopt the solution developed in our previous work [11] for intra-cluster service routing, and use a modified version for inter-cluster service routing. The resulting solution will perform hierarchical service routing in a top-down, divide-and-conquer fashion, in a sense that a single node with partial global state of the system first resolves the inter-cluster service routing problem, and then let certain proxies inside those clusters along the path to find intra-cluster service routes.

The rest of the paper is structured as follows. In Section 2, we describe the assumptions made throughout this paper. Section 3 presents the details about HFC-topology construction. Section 4 provides a hierarchical state information distribution protocol so that when stabilized, each node in the overlay network topology has a partial global state of the system. In Section 5, we show how hierarchical service path finding can be performed. Section 6 is devoted to performance study/comparison of the presented solutions in several aspects, against other topology organization methods. Section 7 concludes the paper with some future directions.

2 Assumptions

2.1 Service Model

This research is based on the composable services model [14,15,16,17], assuming services can be composed together to perform more complex tasks. Such a model allows dynamic customization of services, and is useful, for example, in multimedia application delivery or in Web applications where customizations are specially desirable to overcome various kinds of Internet heterogeneities or to cater to end-users' special needs. As two application instances that may make use of the composable services model:

- An MPEG video stream may undergo a series of transformations for customization: (1) be watermarked for copyright protection; (2) be converted from MPEG to H.261 to reduce bandwidth requirement; (3) be incorporated with a background music, under user's request; (4) be compressed, again, for less bandwidth requirement.
- A Web document may (under user's request): (1) be translated to another language; (2) be merged with another document residing on a certain machine; (3) be formatted.

Service composition has to follow certain dependency relations due to either operational constraints[2] or input/output constraints[3]. We use the notation $s_i \rightarrow s_j$ to denote that service s_i is to be followed by service s_j. A service request is

[2] For copyright protection, watermarking may need to be applied before any other service operations.

[3] For example, if two transcoding services, *MPEG2JPEG* and *JPEG2H261* are to be composed, then the former should be applied before the latter due to the data-type constraints of input/output.

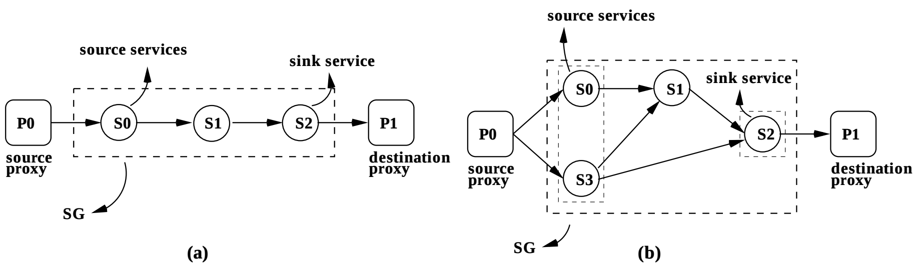

Fig. 2. Two example service requests (source proxy + SG + destination proxy). (a) service request with linear SG: $s_0 \rightarrow s_1 \rightarrow s_2$; (b) service request with non-linear SG which contains three possible configurations: $s_0 \rightarrow s_1 \rightarrow s_2$, $s_3 \rightarrow s_1 \rightarrow s_2$, and $s_3 \rightarrow s_2$.

to find a service path, between a pair of source and destination nodes, such that it satisfies a linear or non-linear service dependency graph (service graph or SG for short), as depicted in Figure 2. While a linear service graph contains only one single service configuration, a non-linear service graph may contain multiple feasible service configurations, because a path that leads from any source service to any sink service is said to satisfy the request (a feasible configuration).

2.2 Overlay Networks

In order to carry out the customizations, services need to be available at certain locations between the multimedia server and client. Since it is not scalable performing all customizations at the end points (server and client), proxies have been introduced to facilitate the deployment of applications[18]. We thus assume in this paper that composable services are installed on these media proxies. We do *not* assume active services or any type of dynamic downloading of services; i.e., in this work, services are statically installed on proxies. The reason that active services are not assumed here is that we believe in the current Internet, they are still hard to be widely deployed, as most system administrators may not allow dynamic installations of software in their systems due to security concerns. The no-active-services assumption generally implies that proxy nodes become different in terms of functional capabilities.

Given a distributed, composable service overlay network, applications demand support from middleware so as to automate processes of locating and composing services. The service path finding problem is, given a service request (source proxy + SG + destination proxy), to find an efficient mapping between services and proxies, so that the end-to-end path is efficient in terms of delay or other QoS metrics [11]. In this paper, we will only consider delay as a performance metric when performing service path finding. A concrete service path may have the form: $sp = \langle -/p_0, s_1/p_1, \ldots, s_n/p_n, -/p_{n+1} \rangle$, where p_0 and p_{n+1} are source and destination proxies, respectively, and s_i/p_j means that service s_i is mapped onto proxy p_j (note that $-/p_i$ means no service is mapped onto p_i, i.e., p_i acts as a message relay.).

3 HFC Topology Construction

As long as the underlying physical network does not partition, a set of overlay nodes can be considered fully connected. However, under such an overlay topology consideration, routing information distribution cost becomes unfeasibly high for a large overlay network, because each proxy would have $n - 1$ neighbors. A more scalable way is to consider the overlay network as a mesh [1,4,5,6]. However, static mesh configurations do not take the service dependency issue, which is mostly resolved at execution time, into consideration. The general consequence of this is two neighboring services will likely appear to be several nodes apart, so that in order to reach one service from another, several intermediary proxy nodes will need to participate in the service path as message relays. Therefore, mesh topologies would incur longer service paths than the fully connected topology. Besides, although in terms of state information maintenance scalability, mesh is better than the fully connected topology, without topology abstraction and state information aggregation, the improvement is still not sufficient for larger-scale overlay networks.

A common approach to achieving scalability is to organize large networks into clusters/groups, so that topology abstraction becomes possible to reduce the size of global state [19,9,8]. Our HFC topology has the following basic properties:

1. **distance-based clustering:** Nodes are clustered by their proximity in the Internet.
2. **connectivity:** All intra-cluster nodes are fully connected among themselves, and all clusters are fully connected by their border nodes. Later on, we will call links crossing two clusters *external links*, and all other links within single clusters *internal links*.
3. **border node selections:** The border nodes between two clusters are selected to be the two closest nodes belonging to the two clusters.
4. **visibility:** Each cluster is visible by its border nodes from outside.

An HFC topology makes service routing in a large-scale overlay network feasible and efficient because of the following reasons. First, clustering allows topology abstraction so as to reduce the state information maintenance overhead. Second, clustering based on the proxies' proximity makes small groups of closely located proxies afford to be fully connected to best cater to runtime-defined service dependency needs. Third, the border node selection rule maximizes routing efficiency between clusters due to geometric properties. Also due to geometric properties, for clusters of reasonable sizes, it's very unlikely that a single node will be selected to be border nodes to all other clusters in the system, which improves load balancing on border nodes. Lastly, when dealing with hierarchical topologies, the most common way of topology aggregation is to represent a group of nodes as a single logical node [19]. Such a representation is simplest, but also introduces too much imprecision [20]. In our framework, we will make all border nodes of a cluster (several nodes instead of a single one) represent a group. Such a visibility feature will help find efficient service paths with better precisions. This issue will become clearer in Section 5.

The construction of an HFC topology follows three major steps: distance map obtainment, clustering, and selection of border nodes.

3.1 Distance Map Obtainment

As end-to-end latency is an important application-level measurable metric that gives a good reflection of the underlying physical network [21,22], we will use this metric to represent Internet distances. Suppose n is the number of nodes in the overlay network, if we are to get a complete distance map for the clustering purpose, then $O(n^2)$ end-to-end measurements are needed, and $O(n^2)$ entries will be in the map, which is an expensive operation. However, in [22], the researchers made two interesting observations, that will allow us to reduce the complexity of the work significantly. Through real-world Internet distance measurements, the authors in [22] observed that: (1) Internet hosts can be mapped into a k dimensional coordinate space such that the geometric distance between every pair of nodes more or less reflects the network distance (round-trip propagation and transmission delay) between the nodes; (2) without directly measuring the distance between a pair of nodes x and y, this distance information can be calculated (predicted) if we know the distances between x and a set of common landmark nodes L, and the distances between y and L. Therefore, to obtain the complete distance map of n overlay proxies, we do the following:

1. Set up a small group of m landmarks - L, and let each of them measure the distances between itself and all other landmarks. To minimize the effect of Internet noises, we take the minimum value of several measurements.
2. Map the obtained distance map of L into a k-dimensional geometric space S with minimum error. This is a function minimization problem solvable by mathematical methods such as [23].
3. Each proxy obtains the coordinates of L, and measures the distances between itself and L in order to be able to derive its own coordinates relative to L's positions, again through some function minimization method [23]. To minimize Internet noises, the minimum of several measurements should be taken.

Using such an approach, a complete distance map of n proxies can be obtained by using $O(m^2 + nm)$ measurements and will only have $O(kn)$ of entries ($n \gg m$ and $n \gg k$), compared to $O(n^2)$ measurements and $O(n^2)$ entries in a genuine approach. Note that the goal of setting up the landmarks is to provide to regular proxies some reference points in the geometric space - S; the landmarks will not participate in any other activities.

3.2 Clustering by Graph Theory

We assume a particular proxy - P - is elected to perform the clustering operations once all of the proxies have reported their own coordinates to it. Clustering is a big and old research area spanning several fields. Many clustering methods exist

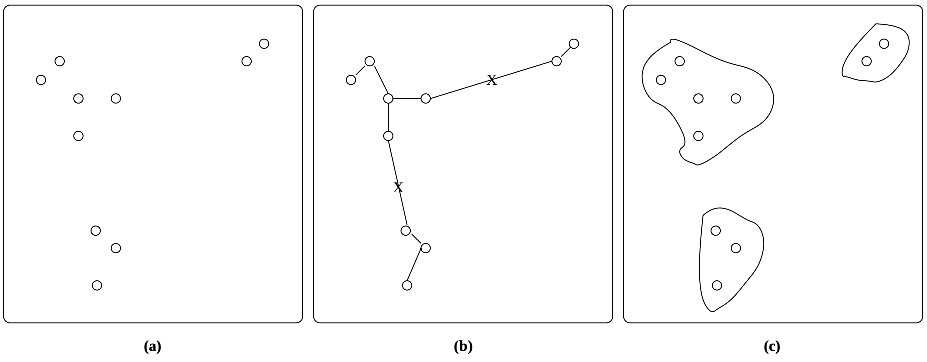

Fig. 3. (a) A set of n nodes in space S; (2) an MST connecting the nodes (inconsistent edges are marked with X; (3) clusters detected by removing the inconsistent edges in (b).

in the literature for different objectives, and most of them are guided by certain laws or principles [24]. Among them, we cite the famous Gestalt principle of grouping by *proximity*. Based on this principle, Zahn [25] demonstrated how the minimum spanning tree (MST) can be used to detect clusters. We adopt this method to detect proxy clusters as follows.

1. Construct the MST for the set of n nodes in S.
2. Identify inconsistent edges in the MST.
3. Remove the inconsistent edges to form connected components and call them clusters.

The crucial step in the algorithm is the definition of inconsistency. We consider an edge to be inconsistent when its length is significantly larger than the average of nearby edge lengths [25]. Let a denote the length of the link being examined - l, and let T_l and T_r denote the left and right sub-trees connected by l, whose average length of links is denoted by b. We say that l is inconsistent if $a/b > k$, where k is a selected number, e.g., 2, 3, ...

In this work, we concentrate on clustering an overlay network of size n using one computation. However, in the real world, we should allow proxies to join and leave dynamically. While we can let future proxies join clusters of their nearest neighbors, multiple joins and leaves may deteriorate the quality of clustering. Hence some kind of re-structuring mechanism needs to be devised. We leave this as our future work.

3.3 Selection of Border Proxies

We also let P carry out the border proxies selection operation. As stated, in the HFC topology, clusters are fully connected by their border proxies. Thus, between every pair of clusters, a pair of border proxies is selected. For maximum routing efficiency, the two border proxies between a pair of clusters are

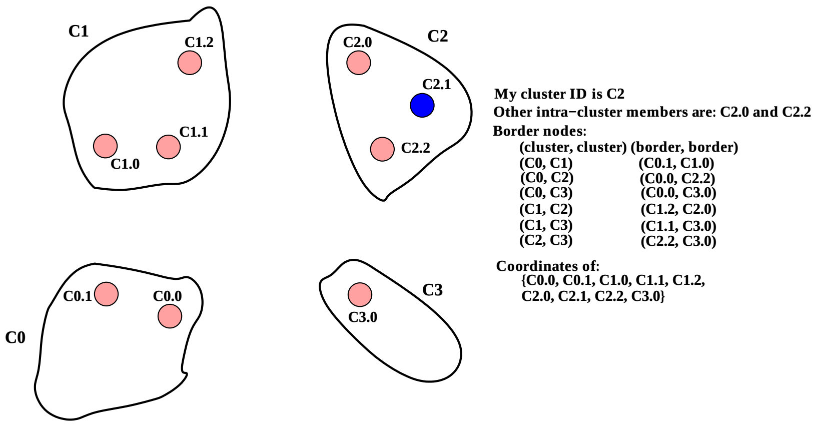

Fig. 4. Information learned by $C_{2.1}$ from P.

selected to be the pair of nearest proxies belonging to the two clusters. Let $X = \{x_1, x_2, \ldots, x_m\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$ denote two clusters, and let $[x_b, y_b]$ denote their border proxies, where $x_b \in X$ and $y_b \in Y$, then for all other pairs of proxies $[x_i, y_j]$ (such that $x_i \neq x_b$, $y_j \neq y_b$, $x_i \in X$, and $y_j \in Y$), $distance(x_i, y_j) \geq distance(x_b, y_b)$.

Once the proxy P is done with clustering and border node selection, it will distribute the relevant topology information to each proxy in the system. In particular, each proxy in the system will learn the following from P: (1) its own cluster's ID and membership information, i.e., who are other members belonging to this cluster; (2) the cluster IDs in the system and their border proxies; (3) coordinates of all members within the cluster and coordinates of all border proxies in the system. Figure 4 depicts the information learned by $C_{2.1}$ from P.

4 Service Routing Information Distribution

As stated in Section 1, service routing needs two pieces of information: distance and service functionality/capability. Since each proxy has already the relevant coordinates information that allows itself to derive distances between any intra-cluster nodes and between any pair of clusters (represented by the corresponding border nodes), only the service capability information needs to be further distributed. Each proxy will maintain two Service Capability Tables, one for all proxies in its own cluster - SCT_P, and the other for all clusters in the system - SCT_C. The following protocol will be adopted for distribution and maintenance of the nodes' service capability information.

1. **Local State:** Every proxy p_i in the system periodically distributes its local service capability information through a *local state* message[4] to all of the

[4] In a *local state* message, p_i lists the names of services installed on p_i.

proxies within its own cluster. A proxy p_j that receives a *local state* message will update its SCT_P.

2. **Aggregate State:** Each border proxy p_b periodically aggregates the service capability information of its own cluster and distributes it through an *aggregate state* message[5] to the neighboring border nodes in other clusters. A border node p'_b that receives such a packet updates its own SCT_C, and is responsible for forwarding it to other proxies of its own cluster. Any proxy that receives a forwarded *aggregate state* packet simply updates its own SCT_C.

5 Hierarchical Service Path Finding

In [11], we provided a generic, global-view-based approach to finding optimal service paths in flat topologies. Due to the service functionality and service dependency restrictions in service routing, no existing solutions for data routing (e.g., Dijkstra's Algorithm) can be directly applied to solve the service routing problem. In [11], we showed how to map the service topology and service request information into a directed acyclic graph (which we call service DAG), to make a classical shortest-paths algorithm applicable for computation of optimal service paths. The main objective of the mapping phase was to get rid of the complexities caused by the two restrictions stated above, so that any path that leads from the source node of the service DAG to the sink node of the service DAG would be a viable service path, and applying a shortest-path algorithm will return us a shortest service path in terms of a given metric.

Since in the HFC framework, the topology and state information has been abstracted at certain point, we will not be able to find a concrete service path in one single step. Instead, we will have to perform service path finding hierarchically, first at the cluster level and then at the proxy level. With only an abstract state of other clusters in the system, no single proxy is able to find a concrete service path solely on its own, unless all requested services can be satisfied in the local cluster. The general idea of hierarchical service routing is to first let some proxy (e.g., the destination proxy specified in the request) look for a cluster-level service path, so that to which cluster each service is mapped to is defined. Later, the proxy can let certain nodes of those particular clusters decide which specific in-cluster proxies will be the concrete providers of those services and combine their decisions to obtain the final service path. Figure 5 depicts a picture of hierarchical service routing at high level. We will call such a service path finding mechanism *divide-and-conquer*. Different than in [11], which finds optimal solutions in terms of a given performance metric, now we can no longer guarantee that the resulting service paths are optimal. That is, although we sought to optimize distance independently at two levels of service routing, the overall service path may not be optimal due to topology abstractions.

[5] In an *aggregate state* message, p_b lists the names of all services available in its entire cluster. Assume $S_1, S_2, \ldots, S_m$ are the sets of services available at proxies $p_1, p_2, \ldots, p_m$ in a certain cluster C_i. Then the aggregate service set S of this cluster is the union of all S_i's; i.e., $S = S_1 \cup S_2 \cup \ldots \cup S_m$.

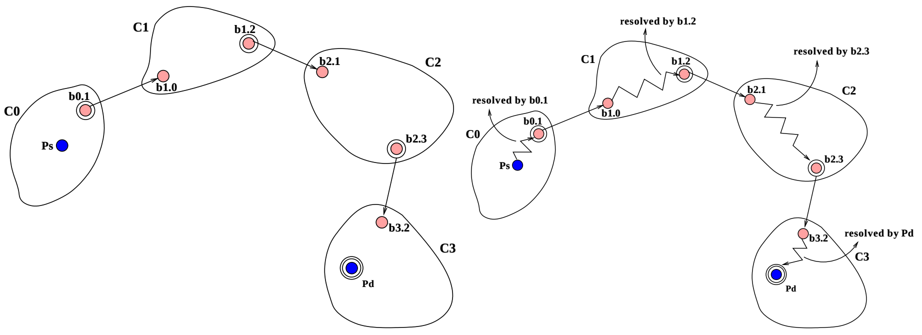

Fig. 5. Hierarchical routing at high level: (a) A single node - p_d - first computes an inter-cluster service path connecting certain border nodes; (b) certain nodes within each cluster individually compute intra-cluster service paths and return their answers to p_d to form the final service path.

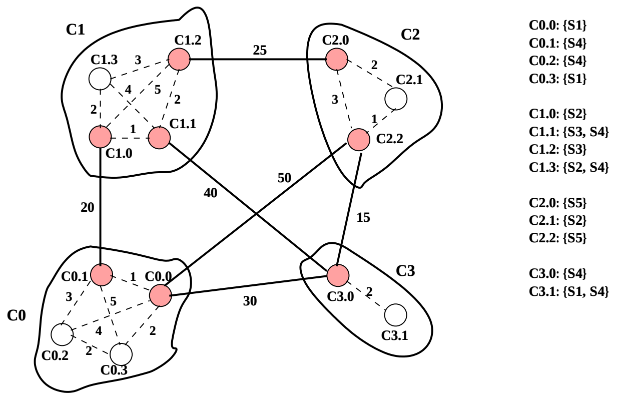

Fig. 6. The service topology (network topology + services) of the example.

In parallel to describing the general procedures of hierarchical service routing below, we provide a small example for better illustration. The topology, as well as the detailed service capability information, are shown in Figure 6. There are four clusters, C_0, C_1, C_2, and C_3, whose elements (proxies) are labeled as $C_{0.0}$, $C_{1.2}$, . . ., according to how they are clustered, to ease our reading. Services available at each proxy are listed at the right side of the figure.

5.1 Inter-cluster Service Path Finding

Without loss of generality, we assume a service request, which comprises a source proxy, a service graph, and a destination proxy, and issued by a client, is sent to the destination proxy, and the output of this destination proxy will feed into the client's input.

1. **map:** Based on the aggregate state information maintained at p_d, find instances of the requested services in all clusters in the system by looking up p_d's SCT_C, and construct a service DAG as described in [11].
   Example: Based on the aggregate global state perceived by C2.1 (Figure 7(a)), map the service request (Figure 7(b)) into a service DAG(Figure 7(c)) by finding instances of each service in SCT_C. The two end nodes of such a service DAG are, respectively, the IDs of the clusters in which the source and destination proxies fall; p_d knows its own cluster's ID, and can query the source proxy for the source proxy's cluster ID. Note that the distance labeled on each link between a pair of clusters - C_i and C_j, is the distance between the border proxies of two clusters. The distance is zero if $C_i = C_j$. Although for simplicity the example only considers linear service graph, the solution can be easily extended to also consider non-linear service graphs, as shown in [11].
2. **apply shortest-paths algorithm:** On top of the service DAG, a shortest-path algorithm such as Dijkstra's algorithm, or DAG-shortest-paths algorithm can be applied to compute a shortest path. We will call the resulting path a CSP (Cluster-Level Service Path). Such a CSP is a service path comprised by possibly several external clusters, whose fine-resolution states are not known at p_d.
   Example: Although simply applying a classical shortest-paths algorithm (e.g., DAG-shortest-paths algorithm) on top of the service DAG would result in a cluster-level shortest service path, whose total distance is the sum of the lengths of external border links making up the path, we modify the DAG-shortest-paths algorithm in such a way that selection of a shortest path also takes into account internal distances as much as possible. If, in the service topology of Figure 7, there are two cluster-level service paths (*path 1:* $C_0 \to C_1 \to C_2$ and *path 2:* $C_0 \to C_3 \to C_2$) that both satisfy the given request, just judging from the external links, the proxy $C_{2.1}$ would see no difference between the two, because both paths have their total external link lengths of 45. However, if $C_{2.1}$ considers the internal distances between the border nodes as well, the latter might be a preferred path because: (1) in *path 1*, the service path will leave C_0 from $C_{0.1}$, enter C_1 from $C_{1.0}$, leave C_1 from $C_{1.2}$, enter C_2 from $C_{2.0}$, and finally reach the destination - $C_{2.1}$, the total distance will be no less than $20 + 5 + 25 + 2 = 52$; (2) in *path 2*, the service path will leave C_0 from $C_{0.0}$, enter C_3 from $C_{3.0}$, leave C_3 from $C_{3.0}$, enter C_2 from $C_{2.2}$, and finally reach the destination - $C_{2.1}$, the total distance will be no less than $30 + 15 + 1 = 46$. At this point, we have no way to know how long the intra-cluster service paths will be, but since the lower-bound distance of path 1 is higher than that of path 2, there's no reason for us not to prefer the latter to the former. In order to take the internal distances into account, we need to add a back-tracking procedure before performing the regular *relax* procedures in a classical shortest-paths algorithm. Details of such back-tracking verification are omitted from this paper. Readers can find similar mechanism used in [11]. The shortest service path is shown in bold lines in Figure 7(c).

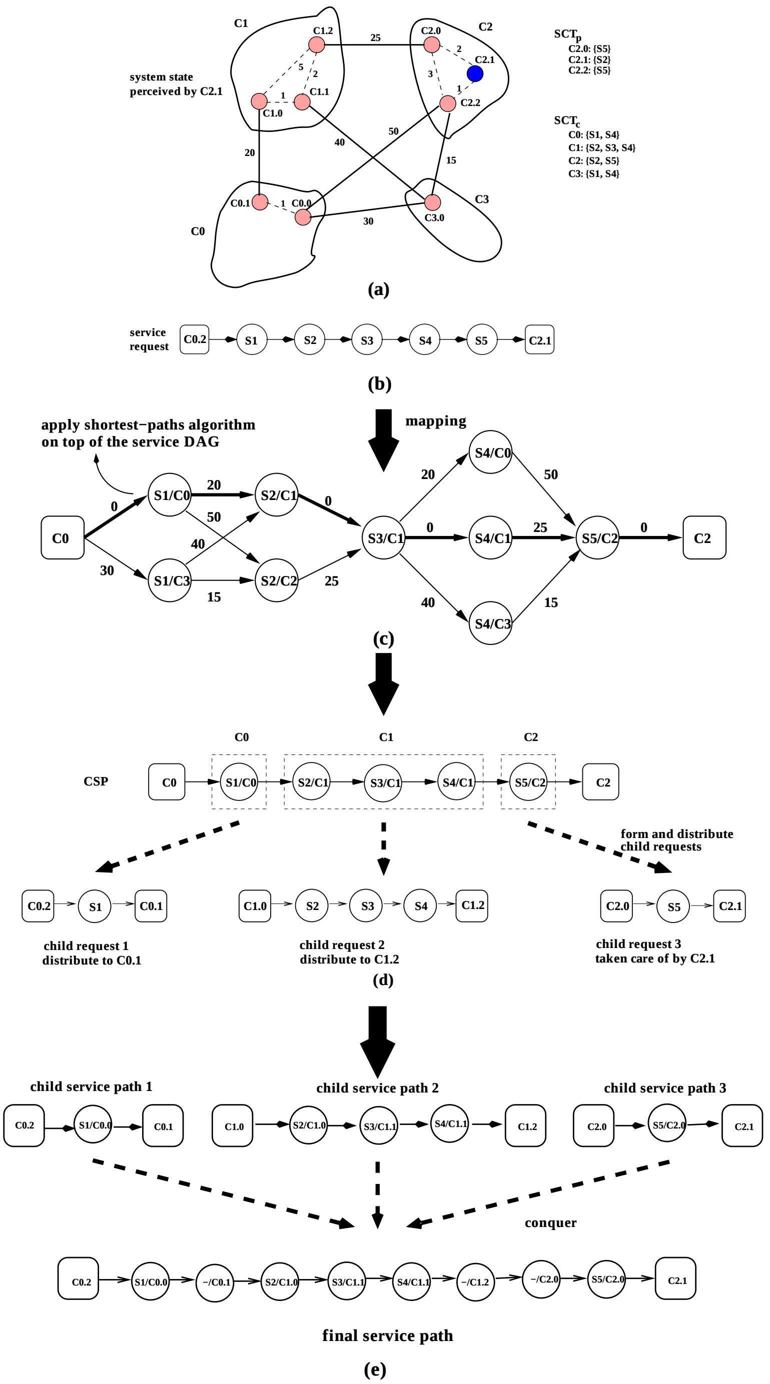

Fig. 7. Inter-cluster service routing (steps performed by the destination proxy $C_{2.1}$ of the service request).

3. **distribute child service requests (divide):** Dissect the original service request into smaller portions, and distribute these child requests to the corresponding clusters.
   Example: Once getting a shortest cluster-level service path, the destination proxy $C_{2.1}$ is responsible for dissecting the original request into pieces. Starting from the source node in the CSP, a new child request is formed for a consecutive set of nodes if these consecutive nodes are all mapped into the same cluster. For each child request, the selection of source proxy and destination proxy is done as follows. Let $\langle C_0, C_1, \ldots, C_n \rangle$ denote the sequence of clusters making up the CSP, and let the notation $b_{i,j}$ denote the border node inside cluster C_i connecting to cluster C_j, then in general, the source proxy of a child request i is $b_{i,i-1}$, and the destination node is $b_{i,i+1}$. If it is the first child request ($i = 0$), the source proxy is that indicated in the original service request; and if it is the last child request ($i = n$), the destination node is that indicated in the original service request. Figure 7(d) shows the dissected child service requests. The first two child requests are to be sent to $C_{0.1}$ and $C_{1.2}$, respectively, and the third one will be taken care of by $C_{2.1}$ itself.
4. **compose child service paths (conquer):** Wait for results of those child service requests to arrive, and compose all child service paths into a single one. This is the final service path.
   Example: The destination proxy $C_{2.1}$ waits for all child service paths[6] to arrive, and combine them as shown in Figure 7(e).

5.2 Intra-cluster Service Path Finding

Since at the cluster level, proxy p_d only knows in which cluster each requested service should be located, it will rely on some in-cluster proxies to further decide which proxy, specifically, will be the concrete provider of each service. A proxy p_x, upon receiving a child service request that consists only of services satisfiable by p_x's cluster, will compute an optimal intra-cluster service path by using the solution described in [11]. The basic procedure consists of two steps: mapping and applying shortest-paths algorithm on top of the obtained service DAG. Different from the mapping step done for inter-cluster service routing above, where each service is mapped into a cluster by looking up SCT_C, now each service will be mapped onto concrete proxies by looking up p_x's SCT_P. After completing the computation of a shortest child service path, p_x sends the result (child service path) back to p_d for it to be composed with others.

Example: Computations of intra-cluster service paths are shown in Figure 8.

6 Performance Studies

In this section, we conduct simulation tests to study the performances of our hierarchical framework. The simulations are done by using the well-known simulator *ns2*, and our Internet topologies are generated following the *transit-stub*

[6] Computations of child service paths are shown in Section 5.2.

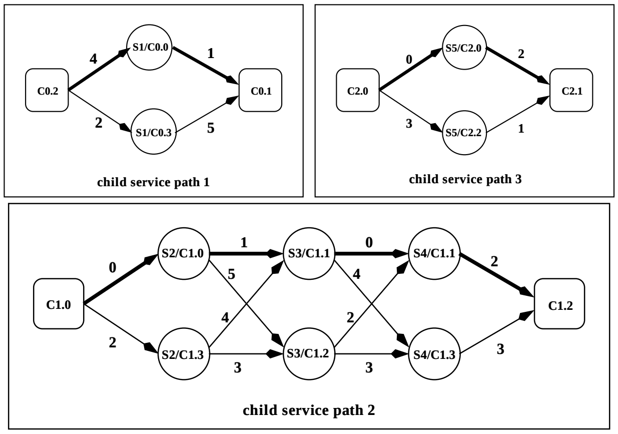

Fig. 8. Computations of child service paths. Shortest paths are indicated with bold lines.

model [26]. We will measure performances of the HFC framework in two aspects: state information maintenance overhead and service path efficiency.

6.1 State Information Maintenance Overhead

The biggest advantage of hierarchical routing is that state information maintenance overhead is reduced through topology abstraction. We will study the performance of the HFC topology by comparing it to that of single-level topologies. Since state in service routing includes two pieces of information: distance and service capability, we will quantify their overheads separately. Overhead is quantified in number of *node-states*; if a single node p keeps n node-states for particular state (either distance or service capability), it means that p maintains that many entries in its corresponding state table where each entry can be either for a single node or for a cluster, depending on situations.

Coordinates-Related Overhead: In this work, since we used coordinates-based distance map in the HFC framework, we will also assume this for single-level topology service routing. In a single-level topology, each proxy is required to keep coordinates of all proxies in the system. Therefore, assuming n is the size of the overlay network, the per proxy coordinates-related overhead is n node-states. However, in an HFC topology, each proxy is only required to keep the coordinates of its local cluster nodes and the coordinates of all border nodes in the system. We set up overlay topologies of different sizes: 250, 500, 750, and 1000 proxies, on top of physical topologies generated by using the TS model [26]. Each overlay topology is tested on top of 10 different physical topologies. Figure 9(a) shows the results averaged over 10 tests.

Service-Capability-Related Overhead: In a single-level topology, each proxy again, needs to maintain service capability information for all proxies (n nodes)

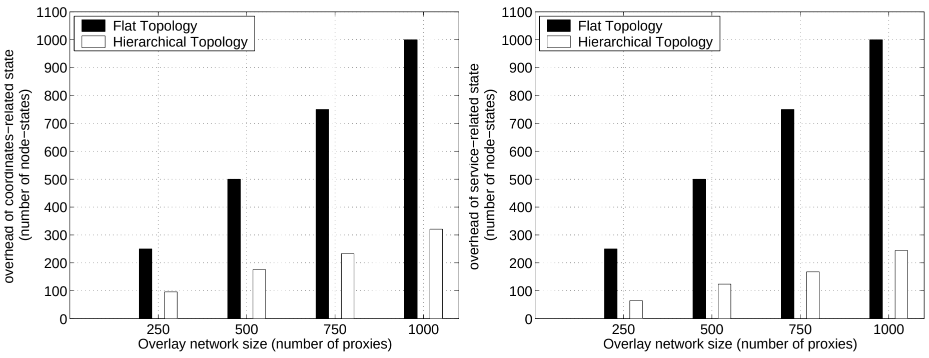

Fig. 9. (a) Number of coordinates-related node-states kept at a single proxy; (b) number of service-related node-states maintained at a single proxy.

in the system. However, in the HFC framework, the number of such nodes perceived at a single proxy is the sum of the number of nodes in each proxy's own cluster plus the number of clusters in the system. Figure 9(b) shows results of the simulation tests with the same setups as above.

While in flat topologies, both overheads would increase linearly with constant one, the increases are much slower in the hierarchical case (with dramatically smaller constants), meaning the HFC framework scales much better than flat topologies. Although theoretically in the worst case, hierarchical organizations may not produce advantages, for example, when most of the nodes fall into one cluster, such undesirable phenomena did not happen in our simulation tests, and we think that extremely unbalanced network node distribution would not happen in practice either. We intend to analyze the node distribution issue further in our future study. Also note that in all of the simulations of this paper, coordinate spaces of two dimensions are used. It would be also interesting, in the future, to quantify the precisions of the distance maps obtained by using coordinate spaces of different dimensions, and see their impact on clustering.

6.2 Service Path Efficiency

The goal of service routing is to find *efficient* service paths. Since in this paper, we only consider distance as the routing metric, we say that for a single service request, a shorter path is more efficient that a longer one. We will compare path efficiencies achieved by our hierarchical service routing framework (HFC with topology abstraction) against those of regular mesh-based solutions. At the same time, we will also quantify the performance losses solely caused by topology aggregation. Hence we will compare path efficiencies achieved by our HFC framework against those achieved by HFC without topology abstraction. To be fair, performances will be compared in the same simulation environments. We wrote programs to generate random simulated environments for overlay topolo-

Table 1. Simulation test environments.

physical topology	landmarks	proxies	clients	services/proxy	service req. length
300	10	250	40	4-10	4-10
600	10	500	90	4-10	4-10
900	10	750	140	4-10	4-10
1200	10	1000	120	4-10	4-10

gies with 250, 500, 750, and 1000 proxies. Table 1 shows the settings. We conduct two sets of tests for different purposes.

We first compare path efficiencies of hierarchical service routing against those of regular meshes. A regular mesh is constructed with the following rules: each proxy creates links to its 1-4 nearest neighbors, and 1-2 randomly chosen, farther located neighbors (to make the topology connected). In a single-level (mesh) topology solution, each node maintains global state of the system. Thus, a single node is able to find an optimal service path by applying the methods described in [11]. The first two bars in each 3-bar set of Figure 10 show the average service path lengths obtained for the two tests. Each point in the figure corresponds to the average result of up to 5 runs (on top of 5 different physical topologies), with 1000 client requests per each run. We see that despite the distance information imprecision introduced in the HFC framework, the performance of the HFC framework is still comparable to (actually slightly better than) single-level mesh solutions. This is the case because in the HFC topology, the number of hops between two overlay nodes (or neighboring services) is constrained to no more than two. As predicted, this feature potentially increases service path efficiencies.

The biggest disadvantage of hierarchical routing is that path efficiencies may get compromised due to routing information imprecision. Thus, it is interesting to examine the performance degradation caused solely by topology aggregation in hierarchical service routing (i.e., performance compromise of doing hierarchical service routing). For this purpose, we compare performances achieved by HFC topologies in two cases. In case one, we perform the topology aggregation as described throughout this paper. In the second case, we do *not* perform any topology abstraction or state information aggregation on top of the HFC topology. Therefore, each proxy will have full state of the whole system. The last two bars in each 3-bar set of Figure 10 give us a comparison between the two; the differences between the two show the performance deterioration caused by the distance imprecision in hierarchical service routing.

7 Conclusions

In this paper, we have presented a hierarchical service routing framework that solves the three challenges described in Section 1, and studied its performances through simulations. Despite the distance imprecision introduced in hierarchical service routing, our framework achieved path efficiencies as good as regular single-level mesh solutions, while also achieving scalability.

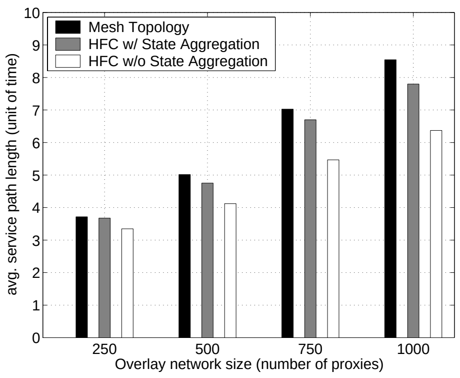

Fig. 10. A comparison of service path lengths.

Service routing is still a new area. To the best of our knowledge, this is the only work so far dealing with hierarchical *service* routing. We expect this research area to be further explored in the future. Some of the improvements that we can think of for our HFC framework can be dynamic membership and QoS. The framework so far does the clustering operations all at once. However, in the real world, we should allow proxies to join and leave dynamically. While we can let future proxies join clusters of their nearest neighbors, multiple joins and leaves may deteriorate the quality of clustering. Thus some kind of re-structuring mechanism needs to be devised. Also, many applications are QoS demanding. How to embed QoS (e.g., network bandwidth, machine load, machine volatility) into hierarchical service topologies, and properly aggregate those pieces of information into meaningful service routing state, are important issues.

Acknowledgments

This material is based upon work supported by DARPA (under Award No. F30602-97-2-0121), NSF (under Awards No. CCR-9988199 and EIA 99-72884 EQ), NASA (under Award No. NAG2-1406), and CAPES/Brazil. 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 also like to thank the anonymous reviewers for their helpful comments.

References

1. 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.
2. 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.

3. Jingwen Jin and Klara Nahrstedt. mc-SPF: An Application-Level Multicast Service Path Finding Protocol for Multimedia Applications. In *Proc. of IEEE International Conference on Multimedia and Expo (ICME2002)*, Lausanne, Switzerland, Aug 2002.
4. D. Xu, K. Nahrstedt,. Finding Service Paths in a Media Service Proxy Network. In *Proc. of SPIE/ACM Multimedia Computing and Networking Conference (MMCN'02)*, San Jose, CA, Jan 2002.
5. Xiaohui Gu, Klara Nahrstedt, Rong N. Chang, Christopher Ward. QoS-Assured Service Composition in Managed Service Overlay Networks. In *Proc. of The IEEE 23rd International Conference on Distributed Computing Systems (ICDCS 2003)*, Providence, Rhode Island, May 2003.
6. Jingwen Jin and Klara Nahrstedt. On Construction of Service Multicast Trees. In *Proc. of IEEE International Conference on Communications (ICC2003)*, Anchorage, Alaska, May 2003.
7. S. Chen, K. Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network Magazine*, 12(6):64–79, 1998.
8. King-Shan Lui, Klara Nahrstedt, Shigang Chen. Hierarchical QoS Routing in Delay-Bandwidth Sensitive Networks. In *Proc. of IEEE Conference on Local Computer Networks (LCN 2000)*, Tampa, FL, Nov 2000.
9. Turgay Korkmaz and Marwan Krunz. Source-Oriented Topology Aggregation with Multiple QoS Parameters in Hierarchical Networks. *ACM Transactions on Modeling and Computer Simulation*, 10(4):295–325, Nov 2000.
10. F. Hao and E. W. Zegura. On Scalable QoS Routing: Performance Evaluation of Topology Aggregation. In *Proc. of IEEE INFOCOM*, Tel Aviv, Israel, Mar 2000.
11. 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.
12. Sumi Choi, Jonathan Turner, and Tilman Wolf. Configuring Sessions in Programmable Networks. In *Proc. of IEEE INFOCOM*, Anchorage, Alaska, Apr 2001.
13. King-Shan Lui, Klara Nahrstedt. Topology Aggregation and Routing in Bandwidth-Delay Sensitive Networks. In *Proc. of IEEE Globecom 2000*, San Franscisco, CA, Nov-Dec 2000.
14. F. Kon, R. Campbell, M. D. Mickunas, K. Nahrstedt, and F. J. Ballesteros. 2K: A Distributed Operating System for Dynamic Heterogeneous Environments. In *Proc. of the 9th IEEE International Symposium on High Performance Distributed Computing*, Pittsburgh, Aug 2000.
15. 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.
16. Xiaodong Fu, Weisong Shi, Anatoly Akkerman, and Vijay Karamcheti. CANS: Composable, Adaptive Network Services Infrastructure. In *Proc. of Third USENIX Symposium on Internet Technologies and Systems*, San Francisco, CA, Mar 2001.
17. 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.

18. A. Fox, S. Gribble, Y. Chawathe, and E. Brewer. Adapting to Network and Client Variation Using Infrastructural Proxies: Lessons and Perspectives. *IEEE Personal Communications*, Aug 1998.
19. The ATM Forum. *Private Network-Network Interface Specification Version 1.0 (PNNI 1.0)*, Mar 1996.
20. B. Awerbuch, Y. Du, B. Khan, and Y. Shavitt. Routing Through Teranode Networks with Topology Aggregation. In *Proc. of IEEE ISCC*, Athens, Greece, Jun 1998.
21. Sylvia Ratnasamy, Mark Handley, Richard Karp, Scott Shenker. Topologically-Aware Overlay Construction and Server Selection. In *Proc. of IEEE INFOCOM*, New York, NY, Jun 2002.
22. T. S. Eugene Ng, Hui Zhang. Predicting Internet Network Distance with Coordinates-Based Approaches. In *Proc. of IEEE INFOCOM*, New York, NY, Jun 2002.
23. J. A. Nelder and R. Mead. A Simplex Method for Function Minimization. *Computer Journal*, 7, 1965.
24. Erich Rome. Simulating Perceptual Clustering by Gestalt Principles. In *Proc. of 25th Workshop of the Austrian Association for Pattern Recognition*, Berchtesgaden, Germany, Jun 2001.
25. C.T. Zahn. Graph-Theoretical Methos for Detecting And Describing Gestalt Clusters. *IEEE Transactions on Computers*, C 20, 1971.
26. E. Zegura, K. Calvert, S. Bhattacharjee. How to Model an Internetwork. In *Proc. of IEEE INCOFOM*, Apr 1996.