Source-Based QoS Service Routing in Distributed Service Networks

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

***Abstract*— Based on the distributed and composable services model, the QoS service routing/composition problem has emerged as middleware support for multimedia applications. Different from the conventional QoS data routing, QoS service routing presents additional challenges caused by the service functionality, service dependency, resource requirement heterogeneity, and loop issues that make solutions for QoS data routing inapplicable to QoS service routing. Existing solutions for addressing this problem are either not generic enough or not integrated, so that they either become inapplicable to new environments/metrics or the computed paths are sub-optimal. This paper presents a generic and integrated approach for computing optimal service paths, and shows an aggregate performance function - $\mathcal{F}$ - that optimizes several QoS metrics at the same time. Simulations show that $\mathcal{F}$ is superior, and integrating service configuration selection with service path finding is desirable.**

keywords: QoS, service routing, service composition, overlay networks

I. INTRODUCTION

The composable services model, which allows individual services to be composed to perform more complex tasks, has been increasingly accepted as a way of achieving flexibility in software applications [1], [2]. Assuming services are widely distributed in networks, applications need special middleware support that solves the QoS service routing problem, which is to find a path such that it satisfies the service and resource requirements and that its performance in terms of given metrics is optimized.

QoS service routing needs special attention because it differs from the conventional QoS (data) routing in several aspects: (1) *service functionality and service dependency* - While data routing is solely based on network connectivities, service routing depends on, in addition to network connectivities, service functionality of the network nodes and dependency relations among services. Later on, we will refer to these two issues as service requirements. (2) *resource heterogeneity* - While in QoS data routing, resource requirement throughout a single data path is homogeneous, in QoS service routing, resource requirement throughout a service path is heterogeneous, because different services may have different requirements on machine capacity and I/O bandwidths. (3) *loop formation* - While data paths should be always loop-free, there is no similar restriction in service paths because it is perfectly sound for a single proxy to be visited recurrently for different services. As a consequence of these issues, solutions for QoS data routing become inapplicable to QoS service routing.

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

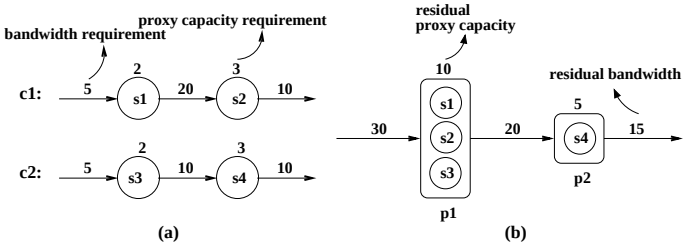

Fig. 1. (a) Two service configurations - c_1 and c_2; (b) Part of a proxy network composed of p_1 and p_2.

While solutions such as [3], [4] exist for QoS service routing, they present certain limitations. In [3], the authors apply an extended Dijkstra's algorithm on top of the proxy topology to compute the best service-to-proxy mapping in terms of their resource safety function, under the assumptions that all services are installed on all proxies (thus service functionality is not a concern) and that a service request has linear dependencies. However, the solution becomes unsuitable for different routing metrics or different assumptions (e.g., if not all services are available on all proxies, or if service requests have non-linear dependencies). In [4], a two-step solution was presented: first, a resource-shortest service configuration (the one with minimum aggregate resource requirement) is selected from the service graph, then for this linear service configuration, a service path is sought in a distributed fashion. However, decoupling service configuration selection from service path finding may yield sub-optimal solutions. Assuming the two service configurations - c_1 and c_2 - in Figure 1(a): using the "minimum aggregate resource requirement" selection criterion, c_2 is the preferred configuration. However, if we check the real service functionality of the proxies (Figure 1(b)), we see choosing c_1 would be actually more advantageous, because CPU bandwidths are usually much larger than those of wide area network links (ratios in magnitude of 100 can be assumed). Moreover, in [4], after a resource-shortest configuration has been selected, the distributed service path finding process adopts a greedy approach (each hop individually chooses the best-performing neighbor as its next hop) which, cannot guarantee the overall service path is optimal.

This paper presents a generic and integrated approach for computing optimal service paths, and shows an aggregate per-

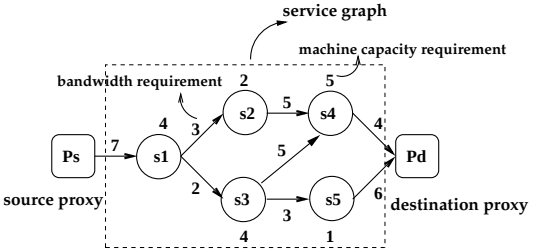

Fig. 2. Example service request with three viable confi gurations: $s_1 \rightarrow s_2 \rightarrow s_4$, $s_1 \rightarrow s_3 \rightarrow s_4$, and $s_1 \rightarrow s_3 \rightarrow s_5$.

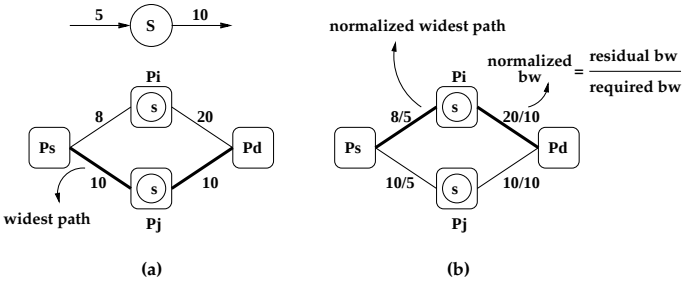

Fig. 3. (a) A genuine widest path; (b) a normalized widest path.

formance function - $\mathcal{F}$ - that tries to optimize several QoS metrics at the same time. We adopt a source-based approach, so that service paths can be computed quickly in a single node based on the currently maintained service and QoS states. The remainder of this paper will be structured as the following. In Section II, we describe the assumptions to be made throughout this paper. In Section III, we present our aggregate routing function - $\mathcal{F}$ - which is composed by several metrics that we consider most relevant to QoS service routing in a proxy overlay network. Section IV describes our solution to the QoS service routing problem. Section V presents some performance results measured from simulation tests. Section VI gives some conclusions.

II. Assumptions

An individual service component is associated with an input data QoS - Q_{in}, and an output data QoS - Q_{out}, where Q_{in} and Q_{out} are QoS vectors of multiple application-level QoS parameters such as image size, image resolution, video frame rate. Each service s has its resource usage function defined as $r_s : Q_{in} \times Q_{out} \rightarrow R$, that computes the amount of resources needed to deliver an output QoS Q_{out} when Q_{in} is the input QoS. When two services, s_i and s_j, are to be composed, then the output quality of s_i should be equal to the input quality of s_j. The notation "$s_i \rightarrow s_j$" is used to indicate that service s_i is followed by service s_j. A *service request* (shown in Figure 2) consists of a pair of source proxy p_s and destination proxy p_d, and a service graph (SG) together with the resource requirements.

As long as the underlying network does not partition, a proxy overlay network can be considered fully connected. However, due to the routing information measurement/maintenance cost issue, application-level networks are usually configured into partial graphs [5], [6]. We leave the topology issue open, as the solution is applicable to both full and partial topologies. We assume service installation on proxies is fixed. That is, we assume services are *not* dynamically downloadable (active services). The reason that we do not consider active services is that to get them widely accepted is not easy, because most system administrators may not allow dynamic installations of software in their systems due to security concerns.

III. Service Routing Performance Metrics

A. Individual Performance Metrics

Performance metrics can be classified into three categories: additive, concave, and multiplicative. A concrete service path will be denoted as: $sp = \langle p_s, s_1/p_1, \ldots, s_n/p_n, p_d \rangle$, where p_s and p_d are source and destination proxies, respectively, and s_i/p_j means that service s_i is mapped onto proxy p_j.

Delay and hop count (additive): Delay d of the service path sp - $d(sp)$ is the time required for the data to get through service path sp, which includes transmission delay and service execution delay; i.e., $d(sp) = \sum_{i=0}^{n} d(p_i, p_{i+1}) = \sum_{i=0}^{n} trans(p_i, p_{i+1}) + \sum_{i=1}^{n} exec(s_i/p_i)$. We consider hop count at the proxy granularity, and the number of proxy hops is the number of times the service path needs to switch proxies (if a proxy is visited twice for two different services, then it is counted twice). The goal of service routing is to find a path such that the aggregate delay and/or the total number of hops are minimized.

Bandwidth and proxy capacity (concave): While in data routing, bandwidth requirement in a single data path is homogeneous, and bandwidth optimization is achieved by seeking the *widest* path [7], in service routing, due to the heterogeneous resource requirement issue, selecting the *widest*-path in absolute value is no longer appropriate. For instance, in Figure 3(a), using the widest-path criterion, service s would be routed through proxy p_j, leaving the residual bandwidth from p_j to p_d zero. While the main objective of the widest-path selection in data routing is to balance traffic on the Internet links, we see that traffic balance is not achieved with the genuine widest-path selection in service routing. To achieve traffic balancing, the residual bandwidth needs to be *normalized* based on the bandwidth requirement. We define the normalized bandwidth as the ratio of residual bandwidth to required bandwidth: $bw_{norm} = \frac{bw_{res}}{bw_{req}}$ (similar to normalized proxy capacity). After the normalization process, the widest-path selection criterion can again help to achieve better traffic balance. Using the normalized widest-path criterion, s should be routed through p_i (shown in Figure 3(b)). With traffic and proxy load balancing in mind, the goal of service routing is to find a path so that the bottleneck normalized bandwidth and the bottleneck normalized proxy capacity are widest. (Methods for measuring end-to-end available bandwidths can be found in [8], [9].)

Proxy volatility (multiplicative): Another metric relevant to QoS service routing is proxy volatility - probability of a proxy being down. Given that proxies may vary greatly in this respect, the objective is to find a service path whose aggregate volatility is the lowest, so that the transmission will most likely be successful. Let $v(p_i)$ denote the volatility of proxy p_i, then $v(sp) = 1 - \prod_{i=1}^{n}(1 - v(p_i))$. Note that each proxy p_i in sp is counted once for sp's volatility, even if p_i is visited more than

once for different services. Minimizing $v(sp)$ amounts to maximizing $\prod_{i=1}^{n}(1 - v(p_i))$ (the probability of successful transmission), which follows the multiplicative composition rule.

B. Aggregate Performance Metric - $\mathcal{F}$

Many multimedia applications require multiple performance metrics, instead of just one, to be optimized at the same time. A common approach to achieving multiple-metric optimization is to define a function and generate a single metric from multiple parameters. Let: (1) p_{i-1} and p_i denote two proxies onto which two consecutive services are mapped; (2) $d(p_{i-1}, p_i)$, $h(p_{i-1}, p_i)$, $bw_{norm}(p_{i-1}, p_i)$ denote, respectively, the delay, hop count, and normalized bandwidth between nodes p_{i-1} and p_i; (3) and $v(p_i)$ and $pc_{norm}(p_i)$ denote, respectively, the volatility and normalized proxy capacity of p_i. We define our aggregate performance function as follows:

$$\mathcal{F}(p_{i-1}, p_i) = \frac{d(p_{i-1}, p_i) * h(p_{i-1}, p_i) * v(p_i)}{\alpha * bw_{norm}(p_{i-1}, p_i) + (1 - \alpha) * pc_{norm}(p_i)}$$

The reasoning behind the function $\mathcal{F}$ is: (1) $delay$, hop, and $volatility$ are all metrics that we want to minimize, thus they are put at the upper part of the fraction, and since they are incomparable to each other, the most meaningful operation among them is multiplication; (2) normalized bandwidth and normalized proxy capacity are something that we want to maximize, thus they are put at the lower part of the fraction. They are comparable, because both are normalized values, thus summation can be a meaningful operation between them. The terms α ($0 \leq \alpha \leq 1$) and $(1-\alpha)$ are used to adjust the weight of each metric in the aggregate function. The soundness of this function will be confirmed through simulations in Section V. $\mathcal{F}$ is additive: $\mathcal{F}(sp) = \sum_{i=0}^{n}\mathcal{F}(p_i, p_{i+1})$ and should be minimized.

IV. QoS Service Routing Solution

In data routing, given a network topology, then a classic graph algorithm, such as the Dijkstra's algorithm and its variants, can be applied to find an optimal network path between two nodes. However, given a proxy/service topology (a graph) and a service request (another graph), none of the existing graph algorithms can be applied directly to compute an optimal service path between two nodes, due to the service requirements. The goal of computing an optimal service path can be accomplished by first completing a *mapping* process, which takes the proxy/service topology and the service request, and maps them into a Directed Acyclic Graph (service DAG). In such a service DAG, any path that goes from the source to the sink node satisfies the service functionality and dependency requirements, thus reducing the complexity of the service routing problem greatly. Once obtained the service DAG with the correspondent node and link resource values, algorithms similar to those for QoS data routing can be applied, albeit with a more complex resource checking process, to compute an optimal QoS service path in terms of a given performance metric m.

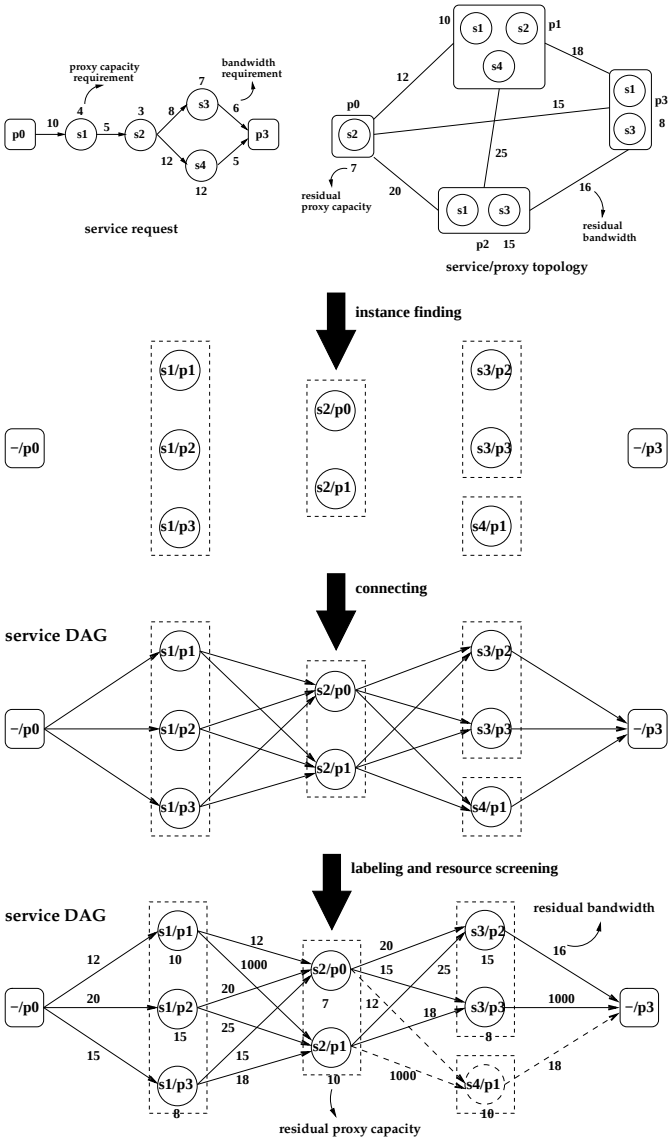

Fig. 4. Mapping process.

A. Mapping

The mapping process takes two pieces of information, proxy/service topology and service request, and maps them into a service DAG as shown in Figure 4. Detailed procedures of mapping are as follows (refer also to Figure 4):

1) *instance finding* - Find, for each requested service, instances of it in the proxy overlay. For example, s_1 in the service request has instances in three different proxies, p_1, p_2, and p_3. These instances are labeled as $s1/p1$, s_1/p_2, and s_1/p_3.
2) *connecting* - Assuming the underlying physical network does not partition, create link from one node n_i to another node n_j in the service DAG if, in the service request, the correspondent node of n_i has a directed link to that of n_j.
3) *labeling and resource screening* - Label all related nodes/links with measured performance values (such as delay, hop count, residual bandwidth, residual proxy capacity, and proxy volatility), and screen out those nodes and links (represented in dashed circles and dashed lines)

whose residual proxy capacity and residual bandwidth are less than those required by the services. For clarity purposes, we only label the residual bandwidths and residual proxy capacities in Figure 4. As mentioned in Section III, bandwidth could be CPU-memory bandwidth or network bandwidth depending on whether or not two consecutive services are mapped onto a single proxy. In Figure 4, we label CPU bandwidths as 1000.

B. Routing Computation

Once the service DAG has been obtained, if it is not because of the *loop* problem, then conventional graph algorithms, such as the Dijkstra's algorithm or DAG-Shortest-Paths algorithm, can be directly adopted to compute paths that optimize certain routing metric. Due to the loop problem, the resource screening step done in the mapping process does not guarantee that a path computed by using a conventional graph algorithm such as the Dijkstra's algorithm, would contain sufficient resources. Taking Figure 5 as an example: if the final computed service path is $\langle \ldots, s_1/p_1, s_2/p_1, \ldots \rangle$, then just knowing that p_1 had enough resources at the mapping stage is not sufficient, because resource checking was only done individually; at s_1/p_1, it was checked only whether or not the resources were sufficient to serve s_1, and at s_2/p_1, it was checked only whether or not the resources were sufficient to serve s_2. However, in practice, either or both services can be mapped onto p_1, and the residual resources may fail to meet the requirements in the latter case.

This indicates that the resource screening at the mapping stage is not sufficient. We address the problem by adding a backtracking resource checking process in a traditional shortest-paths algorithm such as the DAG-Shortest-Paths algorithm. Taking $\mathcal{F}$ as the routing metric, and u and v as two connecting nodes in the service DAG, we only perform $relax(u, v, \mathcal{F})$ after verifying that v's *updated* residual proxy capacity and (u, v)'s *updated* residual bandwidth are both non-negative. The node v's updated proxy capacity is calculated by backtracking to u and all u's predecessors, and subtracting from v's current proxy capacity, the amount of proxy resources that has been consumed by u and u's predecessors. Similar procedure is done for bandwidth checking. The extended DAG-Shortest Paths algorithm, which adds one line (line 4) into the original algorithm, is presented below (note that $pred(u)$ denotes the predecessor node of u):

DAG-SHORTEST-PATHS*(G, $\mathcal{F}$, s)
(1) INITIALIZE-SINGLE-SOURCE(G, s)
(2) for each vertex u taken in topologically sorted order
(3) do for each vertex $v \in Adj[u]$
(4) **if POSITIVE-RESOURCE-CHECKING** **(**$\langle \ldots, pred(pred(u)), pred(u), u, v \rangle$**)**
(5) do $RELAX(u, v, delay)$

bool **POSITIVE-RESOURCE-CHECKING(**$\langle n_1, n_2, \ldots, n_{k-1}, n_k \rangle$**)**
if $pc_{res}(n_k) - \sum_{i=1}^{k} pc_{req}(n_i | proxy(n_i) = proxy(n_k) \geq 0)$
and if $bw_{res}(n_{k-1}, n_k) - \sum_{i=1}^{k-1} bw_{req}(n_i, n_{i+1} | proxy(n_i)$
$= proxy(n_{k-1}), proxy(n_{i+1}) = proxy(n_k)) \geq 0$
then return *true*
else return false

C. Complexity Analysis of the Solution

Mapping: Let N_p denote the number of proxies in the proxy overlay, let N_n and N_l denote, respectively, the numbers of nodes and links in the service graph (SG), and let K denote the maximum number of instances per service in the overlay network (for sparse per service distribution, $K \ll N_p$). Thus, the number of nodes in the service DAG - V - can be written as $V = O(N_n * K)$, and the number of links in the service DAG - E - can be written as $E = O(N_l * K^2)$[1].

Routing Computation Using DAG-Shortest-Paths* Algorithm: The complexity of applying the DAG-Shortest-Paths* algorithm on top of the service DAG, whose number of nodes is V and whose number of links is E, is dominated by the *for* loops. Let L denote the size of longest service configuration in SG, then the complexity of the back-tracking resource checking process inside the inner *for* loop is $O(L)$, (the resource checking can back track up to L nodes/links). The total complexity of DAG-Shortest-Paths* is $O((V + E) * L)$.

V. Performance Evaluation

We implemented different QoS service routing approaches in *ns-2*, and will perform simulation tests to: (1) measure the performances of $\mathcal{F}$ in terms of its individual metrics; (2) compare the performances of the integrated approach against those of the two-phase approach in [4]. We assume a fully connected proxy topology, where each proxy monitors its own node and link conditions actively, and reports the results to other proxies in the system periodically. We consider an overlay topology of 20 proxies, where each proxy is assigned a random amount of capacity, and each link is assigned a random amount of bandwidth. Each proxy has a set of locally available services, and has certain volatility associated with it. We roughly assume the ratio of CPU bandwidth to network bandwidth to be 100.

A. Performances of $\mathcal{F}$

In this study, we concentrate on evaluating the performance of our aggregate metric $\mathcal{F}$ in aspects of its individual metrics: delay, hop count, normalized bandwidth, normalized proxy capacity, and proxy volatility, by comparing it with performances of two cases: *best* and *random*. In the best case, the path computation only seeks to optimize one of the single metrics (such as delay or hop count). In the random case, a service path is chosen randomly. Note that in all cases, the found paths always satisfy the service and resource requirements, the difference lies in their optimization metrics (in the random case, optimization

[1] The complexity of labeling the resource values of the nodes and links depends on how the proxy topology is structured. If the topology is considered a fully connected graph, then performance values of, for instance, bandwidth and delay between two neighboring services, are directly measured and obtained. In this case, labeling a single link or node's performance value takes constant time. However, in case the proxy topology is not fully connected (e.g., a mesh), then the delay between two neighboring services should be the aggregate delay of all links that make up the shortest path between the two nodes, and the bandwidth between two neighboring services should be the bottleneck bandwidth of all links on the shortest path. Both values can be derived using algorithms such as Dijkstra, Bellman-Ford, or Floyd-Warshal, whose performance is no larger than $O(N_p^3)$. Note that aggregate delays or bottleneck bandwidths may be computed once and cached for future service path computations. Updates are only necessary if the delay or bandwidth of certain links have been changed since the last report of state.

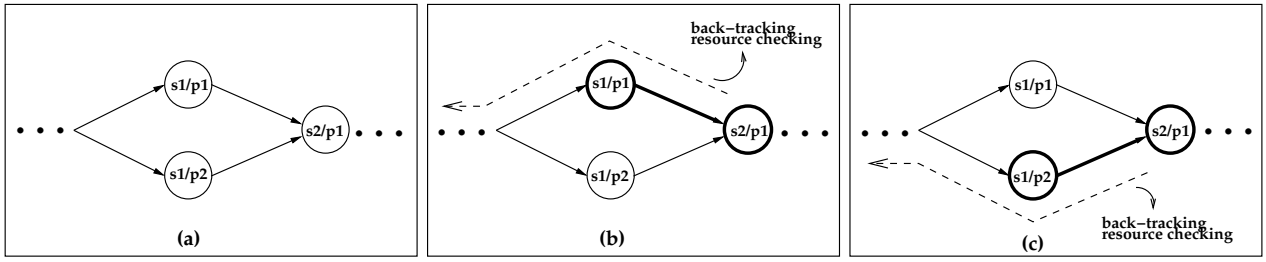

Fig. 5. The back-tracking resource checking process in each relaxation.

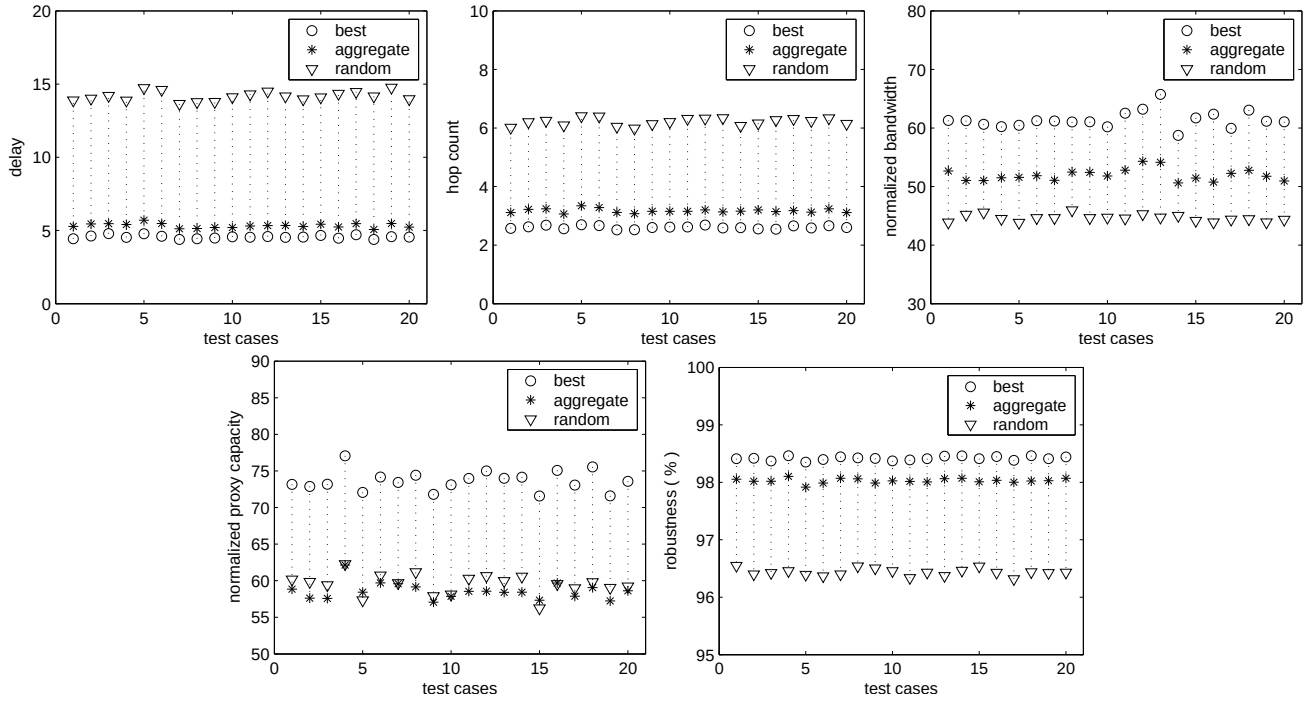

Fig. 6. Performance values of $\mathcal{F}$ compared with those of the best and random cases.

metric is none). We run 20 test cases; with each test case consisting of 200 randomly generated client requests. All 20 test cases have the same environment settings, e.g., in terms of initial proxy capacities and bandwidths. Service requests arrive randomly with a maximum inter-arrival time of 60s, and each established service path may remain active for up to 30 minutes.

Figure 6 shows the performances of $\mathcal{F}$ in terms of individual metrics; the values for each test case are averaged over the results obtained for 200 client requests. We see that $\mathcal{F}$'s individual performances range between *best* and *random*. In most cases, they are closer to *best* than *random*, which indicate that $\mathcal{F}$ is a sound aggregate function. Only in the figure of "normalized capacity" is $\mathcal{F}$'s performance comparable to *random*. This is so because, by nature, $\mathcal{F}$ tends to map consecutive services onto a single proxy in order to reduce hop count and network bandwidth usage. Since not all metrics can be optimized at the same time, the merit comes at a price of compromising proxy capacity balancing in these cases. We say that proxy capacity balancing is compromised in these cases because when two consecutive services have chances of mapping onto a single proxy ($h = 0$), then the value of $\mathcal{F}$ becomes zero, independent of how large or how small the value of pc_{norm} is at the lower fraction of $\mathcal{F}$.

B. Integrated Approach vs Two-Phase Approach

In Section I, it was clear that decoupling resource configuration selection from service path finding may yield sub-optimal paths. In this study, we concentrate on showing the performance differences between the integrated approach and the two-phase approach. In our simulation, each service graph may contain somewhere between three and eight service configurations. In the *two-phase* approach, we first select a "resource-shortest" path from the service graph according to [4], and then compute a service path that optimizes $\mathcal{F}$. Note that in [4], after a resource-shortest path is selected, service paths will be computed distributedly by using a greedy approach (each hop selects the best next hop greedily), but in our test, after a resource-shortest path has been selected, we compute an overall optimal path, because we are only interested in studying the effect of separating configuration selection from service path finding. Figure 7(a) shows the performance results of the two approaches (the x-axis shows the number of the test cases, and the y-axis shows the $\mathcal{F}$ value averaged over 200 client requests). The integrated approach yields better overall performance than

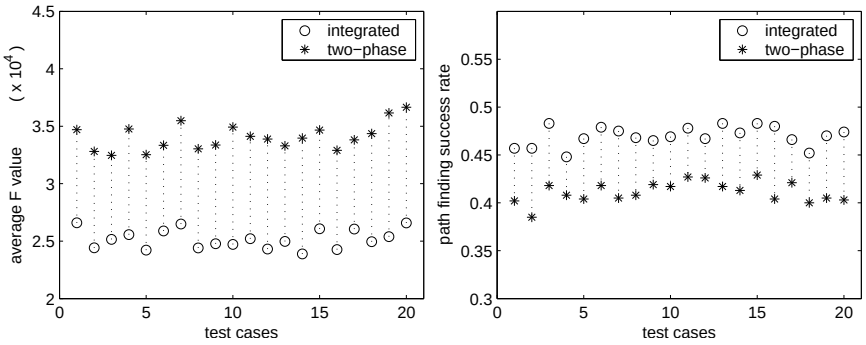

Fig. 7. (a) Comparison of performance values of $\mathcal{F}$ between the integrated approach and the separate approach. (b) Success rates of *integrated approach* vs success rates of *two-phase approach*

the separate approach. Note that in order to make the comparisons fair, in all of the simulation tests conducted so far, we made the available resources at proxies and networks sufficiently large, so that all path findings are successful.

We also compare the path finding success rates of two approaches in faces of resource scarcity. We set lower amounts of resources in the proxies and networks, and let service paths remain active for longer periods of time, so that resources may become used up at some times. We also ran 20 test cases, each consisting of 1000 client requests. Figure 7(b) shows that integrated approach clearly yields better path finding success rates. These simulation results indicate that service configuration selection should not be decoupled from service path finding whenever possible. We say whenever possible because in certain situations, the merit may not worth the associated cost. Tradeoffs need to be carefully evaluated. For instance, in the case of distributed routing, there's tradeoff between the breadth of flooding and path efficiency. Since our approach adopts source routing, integrating configuration selection with service path finding is doable with no additional cost.

VI. Conclusions

In this paper, we have presented an integrated, source-based approach for computing QoS service paths. The main advantages of the source-based approach is that service path finding can be done quickly, without flooding the network. The contribution of this paper is three-fold: (1) the paper presented a generic approach to solving the QoS service routing problem through an additional mapping process[2]; (2) it presented a sound performance aggregate function that tries to optimize several important routing metrics at the same time; and (3) it showed that service configuration selection and service path finding should be an integrated process.

[2] In [10], the authors also devised a service routing method based on mapped topologies. In their approach, for each service in the request, a new layer of the network topology is created, and linked with the layer above it. A shortest-path algorithm is then performed from the source node at layer 0 to the destination node at layer l (assuming l is the number of requested services to be in the path). This mapping may generate huge graphs, because each layer is a copy of the complete network graph. In networks of fair sizes, where it's reasonable to assume the number of instances per service is far less than the number of proxy nodes (i.e., per service distribution is generally sparse), our mapping method would generate much smaller graphs. Also, in [10], no QoS issues were considered.

References

[1] 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.

[2] 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.

[3] D. Xu, K. Nahrstedt, "Finding Service Paths in a Media Service Proxy Network (extended version, available at http://www.cs.purdue.edu/homes/dxu/pubs.html)," in *Proc. of SPIE/ACM Multimedia Computing and Networking Conference (MMCN'02)*, San Jose, CA, Jan 2002.

[4] 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.

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

[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] Z. Wang and J. Crowcroft, "QoS Routing for Supporting Resource Reservation," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, Sep 1996.

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

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

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