This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2009 proceedings

Evaluating Algorithms for Composable Service Placement in Computer Networks

Xin Huang, Sivakumar Ganapathy, and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts, Amherst, MA
{xhuang,sganapat,wolf}@ecs.umass.edu

***Abstract*—Novel network architectures with distributed service components on both routers and end-systems have been proposed to provide the necessary flexibility in the next-generation Internet. Such architectures allow services to be composed along the data path to satisfy different communication requirements. A major operational challenge in such systems is to determine where to perform the required services along the data path. This problem, which we call as the "service placement problem" is proven to be NP-complete when considering resource (e.g., link and processing capacity) constraints. In this paper, we present an evaluation of several existing and newly developed heuristic algorithms for solving the problem. We explore the quality of the achieved placement in terms of effective use of system resources, end-to-end delay, connection drop rate, and load balancing. Our results show the design trade-offs of these different algorithms.**

I. Introduction

The current Internet architecture has been vastly successful in providing simple end-to-end connectivity. However, this architecture also shows its inherent shortcomings of not being able to adapt to the increasing diversity of systems, protocols, and data communication paradigms. To address these problems, the network research community is in the process of designing new networks architectures for the next-generation Internet [1].

In the context of next-generation network architectures, we have proposed and developed the concept of a network service architecture [2]. In this architecture, the network provides processing services in the data path of packets. Connections can request a custom sequence of services to achieve the necessary communication functionality. Examples of such services are any form of packet processing in the data or control plane (e.g., VPN, firewalls, IDS, NAT, web switching, multicast, payload transcoding, encryption, etc.).

A major challenge in this network architecture (as well as in any other architecture where traffic needs to traverse nodes that provide specific functionality) is to determine a suitable routing solution. Traffic does not simply follow the shortest path, but needs to be routed across nodes where certain services are available. Considering the cost of both communication and service processing is essential when optimizing for the least-cost path. We call the problem of determining where service processing takes place and how traffic is routed between these nodes the "service placement problem." This problem is known to be NP-complete when considering resource constraints and several heuristic approaches have been proposed.

In this paper, we explore the service placement problem. We present a novel heuristic algorithm for solving the problem and evaluate this and several existing algorithms extensively. Specifically, the contributions of our paper are:

- A novel service placement algorithm (service step search) that can achieve good mapping results under resource constrains.
- A prototype implementation of service step search and two other existing service placement algorithms (randomized mapping and layered graph) on a single platform.
- A detailed comparison between all three algorithms to determine quantitative tradeoffs in terms of mapping quality and resource utilization.

Our results show that our proposed service step search algorithm performs particularly well in resource constrained networks, where as the layered graph algorithm is less computationally demanding and more suitable for networks without resource constraints.

The remainder of the paper is organized as follows. Section II discusses related work. Section III formalizes the problem statement. Mapping algorithms are introduced in Section IV. Evaluation results are presented in Section V. Section VI summarizes and concludes this paper.

II. Related Work

Configurability in the data path functionality of networks has been suggested in our own work [2] as well as on end-system protocol stacks [3]. An efficient implementation of such network architectures can be achieved with programmable routers [4], [5] or virtualized router platforms [6].

The related service placement problem has been described in the context of active networks by Choi et al. in [7]. When considering processing and link bandwidth constraints, it has been shown in [8] that the problem of finding the optimal solution for even a single connection can be reduced to the NP-complete traveling salesman problem.

A distributed solution to the (unconstrained) service placement problem has been explored in our recent work [9], [10]. Such a distributed solution could be used for inter-AS routing for service placement. However, since a scalable solution inherently requires a "limited view" of the network, a centralized solution is more desirable for intra-AS routing.

This material is based upon work supported by the National Science Foundation under Grant No. CNS-0626690.

978-1-4244-3435-0/09/$25.00 ©2009 IEEE

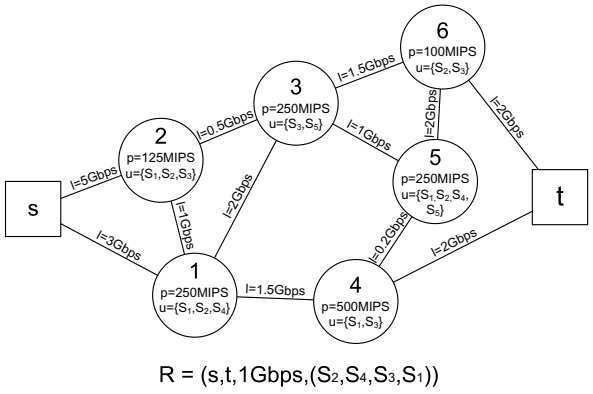


Fig. 1. An Example of the Service Placement Problem. Communication costs and processing costs are unit cost. Processing complexities are 1 instruction/byte.

III. Service Placement Problem

The service placement problem is stated as follows. The *network* is represented by a weighted graph, $G = (V, E)$, where nodes V correspond to routers and end systems and edges E correspond to links. A node v_i is labeled with the set of services that it can perform $u_i = \{S_k | \text{service } S_k \text{ is available on } v_i\}$, the processing cost $c_{i,k}$ (e.g., processing delay) of each service, and the node's total available processing capacity p_i. An edge $e_{i,j}$ that connects node v_i and v_j is labeled with a weight $d_{i,j}$ that represents the link delay (e.g., communication cost) and a capacity $l_{i,j}$ that represents the available link bandwidth. A *connection request* is represented as $R = (v_s, v_t, b, (S_{k_1}, \ldots, S_{k_m}))$, where v_s is the source node, v_t is the destination node, b is the requested connection bandwidth, and $S_{k_1}, \ldots, S_{k_m}$ is an ordered list of services that are required for this connection. For simplicity, we assume that the processing requirements for a connection are directly proportional to the requested bandwidth b. For service S_k, a complexity metric $z_{i,k}$ defines the amount of computation that is required on node v_i for processing each byte transmitted. An example of the service placement problem is shown in Figure 1.

Given a network G and a request R, we need to find a path for the connection such that the source and the destination are connected and all required services can be processed along the path. The path is defined as $P = (E^P, M^P)$ with a sequence of edges, E^P, and services mapped to processing nodes, M^P: $P = ((e_{i_1,i_2}, \ldots, e_{i_{n-1},i_n}), (S_{k_1} \rightarrow v_{j_1}, \ldots, S_{k_m} \rightarrow v_{j_m}))$, where $v_{i_1} = v_s$, $v_{i_n} = v_t$, $\{v_{j_1}, \ldots, v_{j_m}\} \subset \{v_{i_1}, \ldots, v_{i_n}\}$ and nodes $\{v_{j_1}, \ldots, v_{j_m}\}$ is traversed in sequence along the path.

The path P is *valid* if (1) all edges have sufficient link capacity (i.e., $\forall e_{x,y} \in E^P, l_{x,y} \geq (b \cdot t)$, assuming link $e_{x,y}$ appears t times in E^P), and (2) all service nodes have sufficient processing capacity (i.e., $\forall S_{k_x} \rightarrow v_{j_x} \in M^P, p_{j_x} \geq \sum_{y|S_{k_y} \rightarrow v_{j_x} \in M^P} b \cdot z_{j_x,k_y}$). In the above example, paths $P1 = ((e_{s,1}, e_{1,4}, e_{4,t}), (s_2 \rightarrow v_1, s_4 \rightarrow v_1, s_3 \rightarrow v_4, s_1 \rightarrow v_4))$, $P2 = ((e_{s,2}, e_{2,1}, e_{1,4}, e_{4,t}), (s_2 \rightarrow v_2, s_4 \rightarrow v_1, s_3 \rightarrow v_4, s_1 \rightarrow v_4))$, and $P3 = ((e_{s,2}, e_{2,1}, e_{1,3}, e_{3,5}, e_{5,6}, e_{6,t}), (s_2 \rightarrow v_2, s_4 \rightarrow v_1, s_3 \rightarrow v_3, s_1 \rightarrow v_5))$ are all valid.

To determine the quality of a path, we define the total cost $C(P)$ of accommodating connection request R as the sum of communication cost and processing cost: $C(P) = \left(\sum_{x=1}^{n-1} d_{i_x,i_{x+1}}\right) + \left(\sum_{\{(j_x,k_x)|S_{k_x} \rightarrow v_{j_x} \in M^P\}} c_{j_x,k_x}\right)$. In many cases, it is desirable to find the *optimal* connection setup. This optimality can be viewed (1) as finding the optimal (i.e., least-cost) allocation of a single connection request or (2) as finding the optimal allocation of multiple connection request. In the latter case, the optimization metric can be the overall least cost for all connections or the best system utilization, etc. In this paper, we define *optimal* as the first case. Thus, for the above example, the problem is to find the least-cost path ($P1$) among all the valid paths.

In the context of this problem statement, we make several assumptions. We assume connection requests are for constant bit rate (CBR) connections. Once link and processing resources are reserved, they are dedicated to the connection for which they were allocated. We assume that the type and order of services is known at connection setup time and does not change over the lifetime of the connection. We assume that cost for communication and cost for service processing can be represented using a single metric (e.g., delay).

Unfortunately, it has been shown in [8] that even the problem of finding the optimal solution for a single connection in a capacity-constrained network can be reduced to the traveling salesman problem, and thus is NP-complete. This means that no deterministic, polynomial-time algorithm can be found. Therefore, the only feasible solution for a practical system is to use a heuristic algorithm that can find a near-optimal solution. The following section introduces several such approaches.

IV. Mapping Algorithms

In this section, we introduce three algorithms that find approximate solutions to the service placement problem in capacity-constrained networks: (1) *randomized placement*, (2) *layered graph*, and (3) *service step search*. We have chosen these three algorithms for our discussion and evaluation because randomized placement represents the naïve approach to solving the problem, layered graph represents the current state of the art in network service placement, and service step search is our novel contribution to solve practical problems that occur in the layered graph algorithm.

A. Algorithm I: Randomized Placement

The randomized placement algorithm is a very simple approach to service placement. As indicated by the name, nodes where processing is performed are chosen randomly among the nodes that can provide the requested service. Then, the shortest path between these nodes is determined to connect the entire path. If a node does not have sufficient processing capacity or if no path with sufficient link capacity can be found to that node, the random node selection process for that service is repeated. The algorithm terminates when a valid path is found or if the random selection fails a predetermined number of times. Note that the solution is likely to be different every time the placement is repeated.

The benefit of randomized placement lies in its simplicity. Random choices for processing nodes can be computed easily and quickly. To determine the shortest path between these nodes, a simple shortest path algorithm (e.g., Dijkstra's algorithm) can be used. Another benefit of randomized placement is that randomization avoids the problem of systematically getting stuck in a non-optimal solution. The main drawback is that the randomized placement is completely oblivious to the quality of its choices. However, it has been shown that repeated randomized placement can lead to solutions that converge on the optimum [11].

The computational complexity of the randomized placement algorithm for a single request with $|s|$ services in a network with $|V|$ nodes and $|E|$ edges is $\mathcal{O}(|s|\,(|E| + |V|\log|V|))$. This is derived from $|s|$ iterations of the shortest path algorithm, which has a computational complexity of $\mathcal{O}(|E| + |V|\log|V|)$.

B. Algorithm II: Layered Graph

The layered graph algorithm attempts to find a lowest cost path by combining communication cost and processing cost into a single graph structure, called layered graph. Then, a simple shortest path algorithm is run on the structure to determine the best sequence of communication and processing steps. The construction of the layered graph is done as follows: a total of $|s| + 1$ copies of the original network are used to represent the graph layers. The top layer (layer 0) is used for communication before service 1 is performed. The next layer (layer 1) represents communication that is performed after service 1 (and before service 2) is completed. Layer 0 and 1 are connected with vertical edges on all nodes where service 1 processing is possible. The costs of these edges correspond to the processing costs of service 1 on the nodes that they connect. The layering and connecting with vertical edges is continued until all $|s| + 1$ layers are in place. Then, using Dijkstra's algorithm, the shortest path from the source node's instance in layer 0 to the destination node's instance in layer $|s|$ is determined. This resulting path provides the least cost connection from the source to the destination while ensuring that all services are performed in sequence (due to vertical edges). To obtain the final path in the original network, all nodes and edges are projected onto a single instance of the network. Vertical edges in the path correspond to service placements and horizontal edges are used to connect the path. The layered graph for the example problem is shown in Figure 2. This algorithm was described by Choi et al. in [8].

The benefit of the layered graph algorithm lies in its ability to find the least-cost path while considering all service requirements. While this appears to be an ideal solution, we explain in Section IV-D that it shows limitations when considering link and processing capacity constraints. Thus, it may not be able to find a suitable path for a connection request at all.

The computational complexity of the layered graph algorithm is simply that of Dijkstra's algorithm on the layered graph. Since it contains $|s| + 1$ times as many nodes and edges as the original graph, the complexity is $\mathcal{O}(|s||E| + |s||V| + |s||V|\log(|s||V|))$.

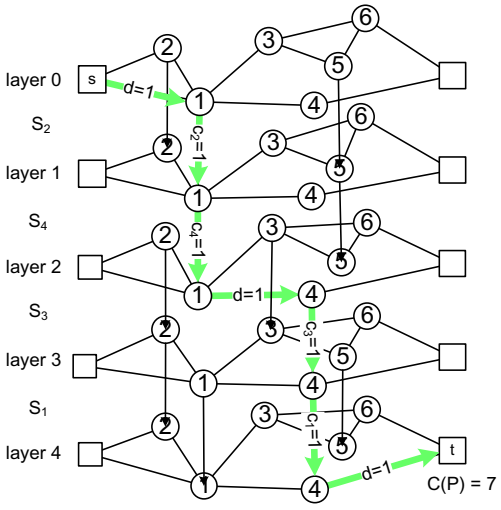


Fig. 2. Layered Graph Algorithm Solution for the Problem in Fig. 1.

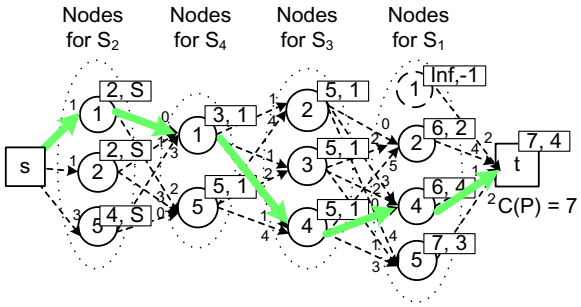


Fig. 3. Service Step Search Algorithm Solution for the Problem in Fig. 1.

C. Algorithm III: Service Step Search

We introduce a new approach called service step search to address some of the shortcomings of the layered graph algorithm. The idea behind service step search is to build a service step search graph that is structured according to the sequence of services that are requested. In the service step search graph, all nodes that can perform a particular service are placed into one level. Nodes in one level are fully connected to all nodes in the next level (i.e., the next service). The weights of these edges correspond to the cost of the shortest path between the corresponding nodes. The service step search graph for the example problem is shown in Figure 3. To determine the least-cost path in the network, Dijkstra's shortest path algorithm is run across the service step search graph. This shortest path algorithm is slightly modified to: (1) consider the processing cost of each service node along the path in addition to link cost, and (2) consider the resource constraints (i.e., the processing and link capacities that may have already been committed to earlier steps in the path).

The benefit of the service step search lies in its ability to consider that processing and link capacities may have already been committed to earlier steps in the path. Thus it solves one of the key problems of the layered graph algorithm. The drawback of the algorithm lies in the higher computational complexity due to the multiple instances of shortest path

computations for constructing the service step search graph.

The computational complexity of service step search is dominated by the cost for constructing the search graph, which requires $|s| \cdot |V|$ shortest path tree computations, and thus is $\mathcal{O}\big(|s||V||E| + |s||V|^2 \log |V|\big)$. The actual search for the best path can be done in $\mathcal{O}\big(|s||V|^2 + |s||V| \log(|s||V|)\big)$.

D. Capacity Constraints

Link and processing capacity constraints are the key consideration for solving the service placement problem. The capacity limitation can occur at two instances: (1) when links or nodes do not have sufficient capacity for a particular connection *before* the placement algorithm is run and (2) when capacity becomes insufficient *while* the algorithm is run.

The first case can be dealt with easily. Before attempting to map connection request R, all links $e_{i,j}$ that do not have sufficient bandwidth (i.e., $l_{i,j} < b$) are (virtually) removed from the network. Similarly, the set of available services u_i for node v_i is adjusted such that service S_k are (virtually) removed if the node does not have sufficient processing capacity for processing the service with required bandwidth (i.e., $p_i < b \cdot z_{i,k}$). The remaining graph thus contains only links and processing nodes that can accommodate request R.

However, it is also possible that the available resources run out while an algorithm is computing the best path (e.g., when reusing a link or a processing node). In such cases, the connection allocation may fail. This issue of capacity constraints has been explored in [12] and is partially addressed by the service step search algorithm. This algorithm can consider which nodes and links have been used along the (so far) shortest path and thus exclude the constrained links and nodes when calculating for the following steps. As shown in Figure 3, when searching the node for S_1, node v_1 is ignored. This is because the shortest paths from source s to all the available nodes for service S_3, map S_2 and S_4 both on v_1. The processing capability of v_1 is thereby used up. Thus the service step search algorithm can avoid oversubscribing a node or link. However, it may still not be able to find the optimal solution under all circumstances. Nevertheless, our results in Section V show that service step search performs considerably better than layered graph due to this improvement.

V. Evaluation Results

With an understanding of the placement problem and the three algorithms that we consider, we turn to the question of how well these algorithms perform, and what are the tradeoffs of using different algorithms to solve the service placement problem.

To evaluate algorithms from the perspective of an individual connection, we consider the following metrics: (1) Successful Connection Establishment: This binary metric considers if the algorithm was able to determine a valid path. As explained earlier, the connection establishment may fail due to resource constraints or due to the inability of the algorithm to find a path. (2) End-to-End Delay: This metric reflects the quality of the path found. Shorter paths and more powerful service processing nodes lead to lower end-to-end delay. This metric is equivalent to the cost of the path, $C(P)$, as described above. In our experiments, we use millisecond (ms) as the unit of the delays.

From the perspective of the entire system, we consider the following metrics: (3) Link Resource Usage: This metric reflects how much network link bandwidth is used by an algorithm to accommodate a set of connection requests. This metric is measured in bandwidth (Mbps) times link delay (ms). (4) Processing Resource Usage: This metric reflects the usage of processing power measured in MIPS. It considers the processing load balancing, i.e., how the required processing power is distributed on each individual service node. The more evenly load is spread across all available service nodes, the more balanced the algorithm performs. Instead of condensing load balance into a single metric, we compare the cumulative distribution function of each algorithm.

A. Evaluation Setup

We use simulation to evaluate performance of the three mapping algorithms. We use two different network setups to highlight different scenarios:

- Network 1 (not resource constrained): In this configuration, we use 96 nodes (divided into 12 ASes of 8 nodes each) with a choice of 4 different services. The Inter-AS link delays are set as 10ms, and the Intra-AS link delays are set as 2 ms. A total of 10,000 connection requests is tested on this network. For each request, the bandwidth is chosen randomly to be between 10 Mbps and 100 Mbps. The source and destination nodes as well as service requirements are also chosen randomly. The bandwidth for each link in the network is configured such that there is just enough link resource for accommodating the corresponding shortest-path connections. The processing capacity for each node is configured such that there is just enough processing resource for all 10,000 connections evenly distributed across all nodes capable of providing a particular service.
- Network 2 (resource constrained): In this configuration, we use again 96 nodes, but the network is divided into 2 subnets (6 ASs in each subnet) such that service s_1 is provided only in subnet 1 and service s_2 is only provided in subnet 2 (no other services are provided). The two subnets are connected by 192 parallel "bottleneck" Inter-AS links, whose bandwidth capacity is set at 100 Mbps. While the Intra-AS delay and Inter-AS delay within the same subnet are set as 2 ms and 10 ms, we set the delay for each of these bottleneck links to be slightly more than 100 ms. This setup is carefully chosen to illustrate the capacity constrained scenario where layered graph is forced to traverse the same bottleneck link twice during a connection setup while the link only has the bandwidth of supporting once. A total of 200 connection requests are tested on this network.

B. Performance Results

We first compare the performance of the three algorithms in terms of connection drop rate and end-to-end connection

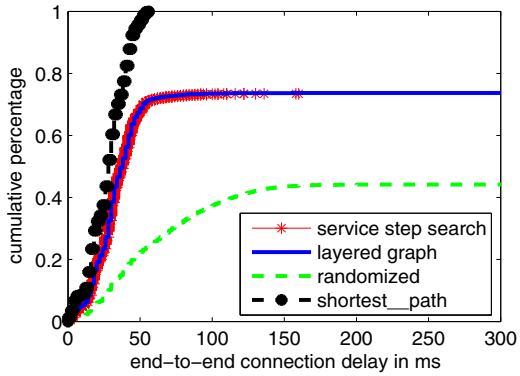


Fig. 4. Cumulative Distributed Function of Connection End-to-End Delay.

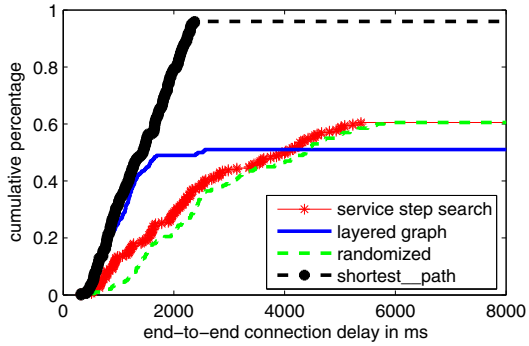


Fig. 6. Cumulative Distribution Function of Connection End-to-End Delay.

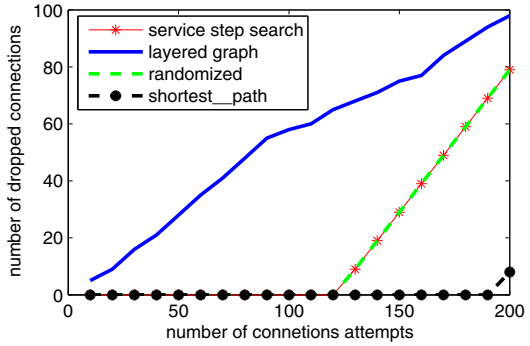


Fig. 7. Number of Dropped Connections over Number of Connection Attempts.

delay on Network 1. In Figure 4, we show the cumulative distribution function of connection delay for all three algorithms in comparison to the shortest path. Clearly, allocating services along a path increases the delay, no matter what algorithm is used. Since the bandwidth resource are limited to the maximum requirement for shortest path, we can also observe that some connections are dropped (y-axis cutoff). The randomized algorithm can only successfully allocate around 40% while layered graph and service step search algorithms can successfully allocate around 70%. This shows the (expected) inefficiencies in choosing service nodes randomly.

To consider how over-provisioning of link and processing resources changes the drop rates, consider Figure 5. The over-provisioning factor (shown on the x-axis for bandwidth and on the y-axis for processing capacity) ranges from $1\times$ to $2\times$. The color indicates the fraction of connections dropped (e.g., 0.3=30%). As expected, increasing either the bandwidth or the processing capacity helps to reduce the connection drop rate. However, it is apparent that for all three algorithms, increasing bandwidth decreases connection drop rate much faster than increasing processing capacity. That is, bandwidth has more impact on successfully allocate connections than processing capacity. Furthermore, the figures again show that randomized algorithm gives higher connection drop rate than the other two algorithms.

To illustrate the benefits of service step search over layered graph (which have both performed equally well in the non-resource-constrained case), we consider Network 2. Figure 6 shows the delay and flow drop rate for this scenario. Clearly, layered graph performs more poorly than the other two algorithms. This is due to resource bottlenecks that are discovered during connection setup and cannot be resolved by layered graph. In contrast, the service step search and randomized algorithms perform better. Looking more carefully at the number of dropped connections over the number of connection attempts, we see in Figure 7 that layered graph drops connections from the very beginning. The other algorithms start dropping when network resources are exhausted.

In terms of systems metrics, Figure 8 shows the link resource usage over an increasing number of successful connections for all the three algorithms as well as a shortest path baseline. As expected, the randomized placement algorithm requires most link resources to connect service nodes. This is due to the random choice of service nodes that does not consider the overall path cost. The service step search algorithm performs better and uses less link resources. The layered graph requires the least link resources among all algorithms, but as discussed before is limited in its ability to find valid paths for connections.

When considering processing load on service nodes, Figure 9 shows the cumulative distribution function of the load. We observe that randomized algorithm achieves the most balanced load, followed by layered graph and service step search. The imbalance in the latter two algorithms is due to nodes on the shortest path being chosen first before other nodes

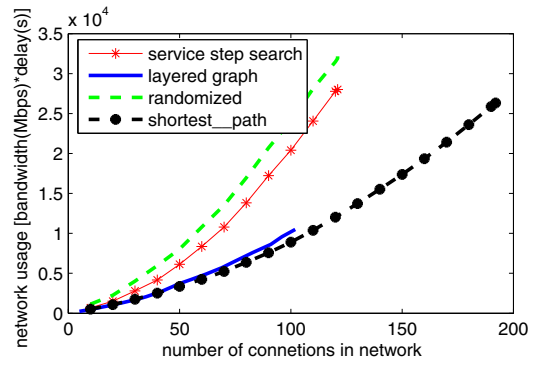
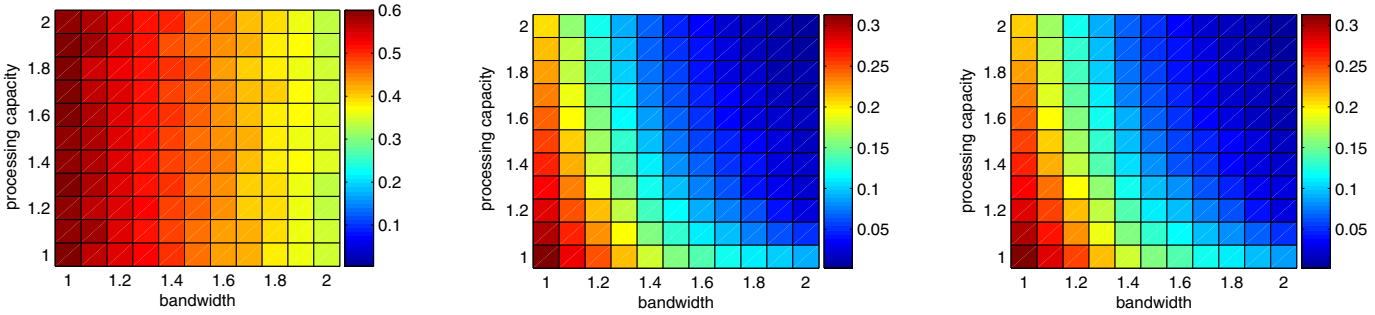


Fig. 8. Network Usage over Number of Successfully Allocated Connections.



(a) Randomized Algorithm (b) Layered Graph Algorithm (c) Service Step Search Algorithm

Fig. 5. Connection Drop Rate of 10,000 Connection Requests for Different Levels of Over-provisioning of Link Bandwidth and Processing Capacity.

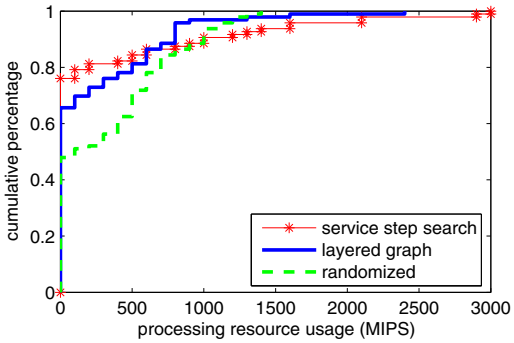


Fig. 9. Cumulative Distribution Function of Processing Node Utilization.

TABLE I
TIME TO COMPUTE PATH FOR CONNECTION REQUEST.

Algorithm	Computation Time	
	Network 1	Network 2
Randomized placement	2.83 s	40 ms
Layered graph	13.14 s	480 ms
Service step search	29.30 s	700 ms

are used. From the figure, we can see that nearly 80% of the nodes are idle for the service step search, 63% for the layered graph and 45% for randomized algorithm.

The above results show that service step search provides the best mapping result when considering both unconstrained and constrained networks. However, the improvements come at the price of higher computational complexity. The results in Table I show the time it takes to place 10,000 connection requests with 0-4 services in Network 1 and 200 connections with 2 services in Network 2. Both layered graph and service step search are considerably more computationally demanding than randomized mapping. Service step search is the most complex algorithm among these three.

In summary, these results show that there is a clear trade-off between the quality of mapping and the runtime of the algorithm. While service step search performs well in some of the scenarios, it has the highest running time. Randomized placement, while performing lower in terms of link resource usage and end-to-end delay, may still be a choice to consider due to its fast run time and ease of implementation.

VI. CONCLUSION

We have presented a discussion of the service placement problem, which occurs in networks where protocols and networking features are composed dynamically and service tasks are distributed across the network. We have evaluated three algorithms for the problem, among which the service step search is a novel contribution. The results show the service step search performs best in terms of finding low delay path and minimizing connection drop rate. We believe this algorithm is an important step towards making network services practical in next-generation network architectures.

REFERENCES

[1] A. Feldmann, "Internet clean-slate design: what and why?" *SIGCOMM Computer Communication Review*, vol. 37, no. 3, pp. 59–64, July 2007.
[2] T. Wolf, "Service-centric end-to-end abstractions in next-generation networks," in *Proc. of Fifteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Arlington, VA, Oct. 2006, pp. 79–86.
[3] R. Dutta, G. N. Rouskas, I. Baldine, A. Bragg, and D. Stevenson, "The SILO architecture for services integration, control, and optimization for the future internet," in *Proc. of IEEE International Conference on Communications (ICC)*, Glasgow, Scotland, June 2007, pp. 1899–1904.
[4] L. Ruf, K. Farkas, H. Hug, and B. Plattner, "Network services on service extensible routers," in *Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005)*, Sophia Antipolis, France, Nov. 2005.
[5] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Proc. of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.
[6] T. Anderson, L. Peterson, S. Shenker, and J. Turner, "Overcoming the Internet impasse through virtualization," *Computer*, vol. 38, no. 4, pp. 34–41, Apr. 2005.
[7] S. Y. Choi, J. S. Turner, and T. Wolf, "Configuring sessions in programmable networks," in *Proc. of the Twentieth IEEE Conference on Computer Communications (INFOCOM)*, Anchorage, AK, Apr. 2001, pp. 60–66.
[8] ——, "Configuring sessions in programmable networks," *Computer Networks*, vol. 41, no. 2, pp. 269–284, Feb. 2003.
[9] X. Huang, S. Ganapathy, and T. Wolf, "A distributed algorithm for network service placement," in *Proc. of Seventeenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, St. Thomas, USVI, Aug. 2008.
[10] ——, "A scalable distributed routing protocol for networks with data-path services," in *Proc. of 16th IEEE International Conference on Network Protocols (ICNP)*, Orlando, FL, Oct. 2008.
[11] R. M. Karp, "An introduction to randomized algorithms," *Discrete Applied Mathematics*, vol. 34, no. 1-3, pp. 165–201, Nov. 1991.
[12] S. Y. Choi and J. S. Turner, "Configuring sessions in programmable networks with capacity constraints," in *Proc. of IEEE International Conference on Communications (ICC)*, Anchorage, AK, May 2003.