# A Scalable Distributed Routing Protocol for Networks with Data-Path Services

Xin Huang, Sivakumar Ganapathy and Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts
Amherst, MA 01003, USA
{xhuang,sganapat,wolf}@ecs.umass.edu

***Abstract*—Advanced packet processing functions in the data path of routers are commonly used in the current Internet and are likely to expand in next-generation networks. One of the key challenges related to such "network services" is to develop a suitable control plane infrastructure for routing and connection setup. In this paper, we present a novel distributed algorithm and routing protocol to facilitate allocation of services during connection setup. The protocol is scalable for deployment in large networks and can obtain globally optimal solutions for a single service and approximate solutions for two or more services. We have implemented a prototype on Emulab to evaluate the proposed protocol. Our implementation shows that the proposed protocol is indeed an effective and scalable solution to the routing problem in networks with data-path services.**

## I. Introduction

Computer networks have become a critical infrastructure in today's world. Network systems achieve interoperability among heterogeneous systems through the use of a set of well-established protocols. The generality and wide acceptance of the TCP/IP protocol suite has significantly contributed to the vast success of the Internet. However, since the inception of these Internet protocols in the 1970's, technological advances in areas of embedded computing systems, wireless networks, and distributed computing have expanded the functional requirements for networks.

Modern networks need to be able to adapt to evolving security threats, provide connectivity to a variety of different types of end-systems, and support emerging networking paradigms ranging from peer-to-peer networks to content networks. This need for flexibility is not only realized through the use of novel protocols, but also through a fundamental shift in the architecture: instead of limiting networks to provide simple store-and-forward packet processing, more complex "network services" are moved from end-system to the network. Such services provide a variety of processing capabilities that go well beyond packet forwarding (e.g., content inspection, payload transcoding, etc.).

In this paper, we address the problem of how to manage such services in a scalable fashion. We focus questions related to the control plane of such a service-enabled network: What information do routers need to exchange in order to determine where different services are available? What routing algorithm can provide an optimal or near-optimal connection setup? How can such a system be designed with Internet-level scalability? Similar questions have been addressed in prior and related work, but solely for *centralized* systems (i.e., nodes either are aware of the entire network, or a single node controls all connection setups). Clearly, a centralized solution imposes limitations that make it unrealistic for large-scale deployment. In our work, we develop a *distributed* routing algorithm and a matching protocol, where our two-level hierarchy follows the Internet's Autonomous System (AS) concept.

The specific contributions of this paper are:

1) A distributed algorithm for solving the service placement problem. We describe Distributed Service Matrix Routing (DSMR) and its approximation as novel approaches to determining the best instantiation of service in the network based on a connection request.
2) A scalable routing protocol for implementing DSMR on a service-enable network. Our Distributed Service Routing Protocol (DSRP) manages the exchange of control information between Autonomous Systems to ensure that connection requests can be routed optimally.
3) An extensive evaluation of the performance of DSMR and DSRP on a prototype implementation on Emulab. Using a large, 60-node configuration, we have tested and evaluated our design. On testing the prototype with nearly 150,000 connection requests, the results show that our protocol is scalable and efficient.

The remainder of the paper is organized as follows: Section II introduces related work. Section III discusses network services in more detail and provides several examples. The service placement problem and routing algorithms to solve the problem (both centralized and distributed) are introduced in Section IV. The DSRP protocol that implements our distributed algorithm is discussed in detail in Section V. Results from our prototype implementation are shown in Section VI. Section VII summarizes and concludes this paper.

## II. Related Work

The term "network service" has been used in a variety of different contexts ranging from application-layer end-system service (e.g., services in a grid computing environment [1] or Cisco Service-Oriented Network Architecture [2]) to packet processing services on routers (e.g., data-path services for the next-generation Internet [3] or programmable networks [4]). While we consider the latter context, our routing algorithm and

1-4244-2507-5/08/$20.00 ©2008 IEEE

protocol can be used for any type of service that is performed between communicating end-systems.

As indicated above, our focus in this work is on the control plane. Implementing packet processing services in the data plane has been studied extensively on different router architectures: workstations routers [5], programmable routers [4], virtualized router platforms [6], and Planetlab [7]. While this approach is distantly related to active networks [8], we consider more controllable programmable routers [9], where users can choose from predefined processing features (i.e., services).

To manage network services, an IETF working group has attempted to define Open Pluggable Edge Services (OPES) [10]. In such an architecture, end-systems can specify a set of data flow operations that are implemented on nodes throughout the network. TCP tunnels are used to create a multi-hop end-to-end stream. Also, several protocols for locating services have been proposed (e.g., Service Location Protocol [11]). Our work ties in with these efforts, but focuses on the routing aspect of service-enabled architectures.

The service placement problem that we address in Section IV has received some attention. Choi et al. have developed one of the earlier algorithms [12] that we use as a comparison in our work. This algorithm was later expanded to consider capacity constraints [13]. Ruf et al. consider service placement across network nodes and with multiprocessor packet processing systems [4]. All of these approaches are centralized and thus exhibit limited scalability. Our decentralized approach can scale to larger networks.

## III. Data-Path Services

Before discussing our routing algorithm and protocol, we first illustrate the concept of network services in more detail. Such services appear both in the current Internet and in proposals for next-generation network architectures.

### A. Service in Current Internet

There are several examples of network services that have been implemented in the current Internet. While terminology may vary, these services are characterized by processing functions performed in the data path of the router:

- Virtual Private Networking (VPN) ( [14])
- Intrusion Detection Systems (IDS) (e.g., snort [15])
- Firewalls [16]
- Network Address Translators (NAT [17])
- Web Switching [18]

Typically, such services are not controllable from the perspective of an end-system. If a router administrator enables certain functions (e.g., IDS or NAT), then packets traversing this node are provided with this service. There are recent efforts to manage such transparent "middleboxes" (e.g., NAT) in the existing Internet [19].

### B. Service in Next-Generation Internet

As networks need to handle more diverse end-systems (e.g., mobile devices, embedded systems) and novel networking paradigms (e.g., sensor networks, ad-hoc wireless networks), new network architectures are being proposed. The concept of data-path processing services is being proposed as an inherent feature of the network:

- The SILO project proposes services as building blocks for custom protocol stacks [20], [21].
- Recent work proposes a decomposition of end-to-end connections into a sequence of services [3], [22].

In both approaches, services are implemented on routers and can be located throughout the network. In most cases, there is an inherent sequential ordering of services (e.g., intrusion detection needs to be performed after VPN decryption).

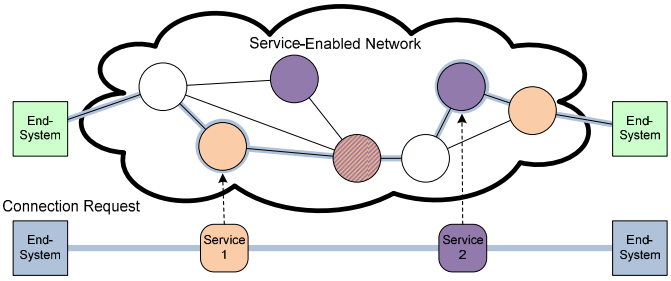


Fig. 1. Connection Setup with Network Services. Colored nodes indicate capability to handle a particular service.

### C. Control Plane Challenges

Services in both the current Internet as well as the next-generation Internet require some level of support from the control plane. In the current Internet, this support typical involves manual configuration. In the next-generation network, we expect that many more functions get automated. In this paper, we address one of the most fundamental questions in this context:

> *How can we achieve optimal or near-optimal routing of connection requests that involve data-path services?*

This problem is illustrated in Figure 1, where a connection that requires two services is shown. Several nodes are potential candidates for handling each service. The goal of our work is to find the optimal (i.e., least-cost) path through the network such that both services are performed along the way. In this context, we need to determine how this problem can be solved algorithmically (see Section IV) and how a protocol can efficiently implement this solution (see Section V).

We make the following assumption in our work:

- We assume that the network supports the establishment of connections with fixed paths. This is required since traffic needs to traverse a particular set of service nodes as determined by our algorithm.
- We assume that services are performed in a fixed sequence that is specified at connection setup time and will not change over the life-time of the connection.
- We assume that capacity constraints (for communication and processing) are not considered during routing. Considering capacity limitations makes the routing problem intractable [13].

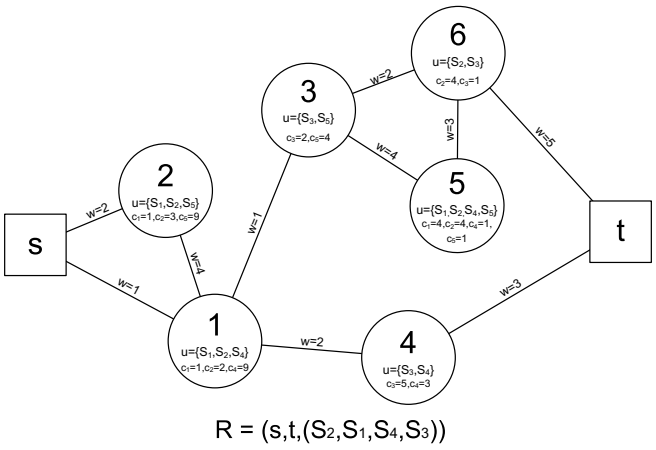


Fig. 2. An Example of the Service Placement Problem.

- We assume that cost for communication and cost for service processing can be represented using a single metric (e.g., delay).

Despite these necessary assumptions, the problem remains general enough to be representative of scenarios that occur both in the current Internet as well in potential future networks.

## IV. Service Placement Algorithms

In this section, we first discuss and formalize the routing problem with service placement in a service-enabled network. Then we describe one state of the art centralized solution, the layered graph algorithm [12]. Next, we present a distributed algorithm, DSMR, followed by its approximation as better solutions for solving the problem in large hierarchical networks where the centralized solution is neither applicable nor scalable. Finally, we analyze and compare the complexity of all the three algorithms.

### A. Service Placement Problem

Routing traffic that requires various services through a network can be divided into two subproblems: (1) determining the placement of services (i.e., which node should perform the services), and (2) determining the shortest path between these service nodes. The latter – finding the shortest path – is a well-understood problem that has been solved. Determining which nodes should be used for services is more challenging. We define this problem, which we call the *service placement problem*, as follows. An example of the service placement problem is shown in Figure 2.

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. Each edge $e_{i,j}$ that connects nodes $v_i$ and $v_j$ is labeled with a weight $w_{i,j}$ that represents the communication cost (e.g., delay). Each node $v_m$ is labeled with the set of services that it can perform $u_m = \{S_j | \text{service } S_j \text{ is available on } v_m\}$ and the processing cost $c_{m,j}$ (e.g., processing delay) of each service. A *connection request* is represented as $R = (s, t, (S_{j_1}, \ldots, S_{j_k}))$, where $s$ is the source node, $t$ is the destination node, and $S_{j_1}, \ldots, S_{j_k}$ is an ordered list of services that are required for this connection.

Given a network $G$ and a request $R$, we need to find a path for the connection such that the source and destination are connected and all the required services are 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, $M^P$: $P = ((e_{s,v_{i_1}}, \ldots, e_{v_{i_h},t}), (S_{j_1} \rightarrow v_{m_1}, \ldots, S_{j_k} \rightarrow v_{m_k}))$. To determine the quality of a path, we define the total cost $C(P)$ of accommodating connection request $R$ as the sum of link cost and processing cost: $C(P) = \left(\sum_{\{(x,y)|e_{x,y} \in E^P\}} w_{x,y}\right) + \left(\sum_{\{(j_i,m_i)|S_{j_i} \rightarrow v_{m_i} \in M^P\}} c_{m_i,j_i}\right)$.

In many cases, it is desirable to find the *optimal* connection setup. We view this optimality in terms of the least cost allocation of a single connection request.

### B. Centralized Solution

To solve the problem state above, a solution has been proposed by Choi et al. in prior work [12]. This solution is a "centralized" solution in the sense that it requires a complete view of the network in order to optimally allocate a connection request. The main idea is to represent the network as a graph with multiple layers, where each layer indicates which service of the request has already been processed. The optimal path can be found by simply employing Dijkstra's shortest path algorithm on the multi-layer graph and mapping it back to the original network.

The construction of this so-called "layered graph" is done as follows: To route a request $R = (s, t, (S_{j_1}, \ldots, S_{j_k}))$, a total of $k + 1$ instances of the original network are used to represent the graph layers. The top layer (layer 0) is used for communication before service $S_{j_1}$ is performed. The next layer (layer 1) represents communication that is performed after service $S_{j_1}$ (and before service $S_{j_2}$) is completed. Layers $i-1$ and $i$ are connected with vertical edges on all nodes $v_m$, where service $S_{j_i}$ processing is possible (i.e., $S_{j_i} \in u_m$). The costs of these vertical edges are $c_{m,j_i}$, which correspond to the cost of processing service $S_{j_i}$ on node $v_m$. Vertical edges are added between all $k+1$ layers. Then, using Dijkstra's algorithm, the shortest (least-cost) path is calculated from the layer 0 instance of source node $s$ to the layer $k$ instance of destination node $t$. The 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 details of the algorithm are discussed in [12].

However, there are several fundamental limitations of the layered graph algorithm that limit its scalability to large network deployment: (1) The centralized algorithm is not scalable. It needs a complete view of the network including all links and nodes, and for each connection request a different layered graph may need to be constructed and a shortest path algorithm needs to be calculated. (2) The algorithm is not compatible with the "hierarchical" networks, where nodes are organized into ASs and detailed topology information from inside an AS is not visible to other ASs.

To address these issues, we present a distributed algorithm that can solve the service placement problem in a more scalable and autonomic way.

### C. Distributed Solution

In this section, we first present the general concepts of two distributed algorithms - the Distributed Service Matrix Routing (DSMR) algorithm and the approximate DSMR algorithm. Later, we explain the working of the algorithms in detail.

*1) Service Placement as Dynamic Programming Problem:* The difference of service placement problem from the shortest path problem is that instead of only considering the path cost we also need to consider a second dimension, the services. Following the idea of the Distance-Vector (DV) Routing Algorithm, we have discovered that a solution to this problem. The basic idea is to use a dynamic programming approach similar to what Bellman proposed for shortest path routing [23]. Let $c_v^{j_1,\ldots,j_k}(t)$ denote the cost of the shortest path from node $v$ to node $t$ where services $S_{j_1},\ldots,S_{j_k}$ are performed along the way. For shortest path computation (i.e., no services), we use the notation $c_v^-(t)$. Thus, a node $v$ can determine the least-cost path by considering to process $i$ ($0 \le i \le k$) services and forwarding the request to any neighboring node $n_v$ ($n_v \in \{x \in V | e_{v,x} \in E\}$):

$$c_v^{j_1,\ldots,j_k}(t) = \min_{0\le i\le k}\left(\sum_{l=1}^{i} c_{v,j_l} + \min_{n_v}\left(w_{v,n_v} + c_{n_v}^{j_{i+1},\ldots,j_k}(t)\right)\right). \quad (1)$$

The argument $i$ on the right side determines how many of the $k$ services that need to be performed should be processed on node $v$. Note that if $i = 0$, no service is processed, i.e., $\sum_{l=1}^{i} c_{v,j_l} = 0$. If $i = k$, all the services will be processed on node $v$, i.e., $c_{n_v}^{j_{i+1},\ldots,j_k}(t) = c_{n_v}^-(t)$. The argument $n_v$ determines to which neighbor of $v$ the remaining request should be sent.

*2) DSMR Algorithm:* Following the above idea, we can design a distributed algorithm to solve the service placement problem. We structure our discussion by distinguishing two important steps of the algorithm: (1) routing information exchange, where information pertinent to routing and service placement is exchanged between nodes, and (2) request routing, where the routing decisions for connection requests are made by nodes based on the information exchanged by step (1).

*a) Routing Information Exchange:* In this procedure, nodes exchange their $c_v^*(\star)$ values ('$*$' represents any service sequence and '$\star$' represents any destination) with neighboring nodes (similar to how it is done in RIP for distance vector routing [24]).

To capture this information, we define a control plane data structure called "service matrix" (see Figure 3). This matrix is a two-dimensional extension of a distance vector, where the second dimension is an enumeration of all possible services. As in distance vector routing, the first dimension is a list of destinations (or destination prefixes).

At the initialization stage, each node builds up the matrix based on the knowledge it possesses about itself (e.g., services available, processing capacity, link delays to neighbors etc.). Next, each node advertises its service matrix to its neighbors. In this way, all nodes can collect all $c_n^*(*)$ values from their neighbors and thus solve Equation 1 to populate their own service matrices. Finally, every node will get all the advertisements from other nodes and the matrices of all the nodes will converge to a consistent state.

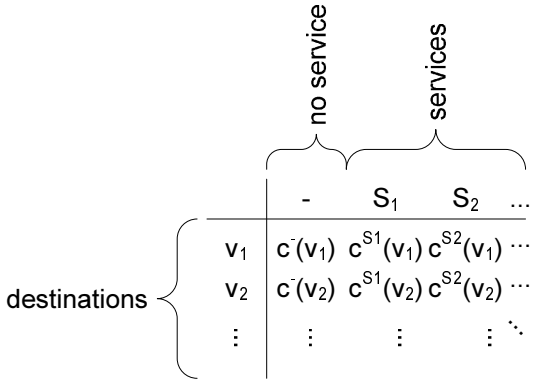


Fig. 3. Service Matrix Data Structure.

*b) Request Routing:* When handling connection routing and setup, a request $R = (s, t, (S_{j_1},\ldots,S_{j_k}))$ is propagated from $s$ to $t$. A node $v$ along the path needs to determine from its service matrix which services $S_{j_1},\ldots,S_{j_i}$ should be processed locally. This can be done simply by looking up $c_v^{S_{j_1},\ldots,S_{j_k}}(t)$. Rather than having to compute Equation 1 for each connection request, we assume that the control plane precomputes $c_v^*(*)$ and the corresponding number of local services $i$ and next hop $n_v$.

Using a simple lookup into the service matrix, a node can determine that it should allocate $i$ services to itself and pass a request for the remaining services along to its neighbor $n_v$ (i.e., $R' = (s, t, (S_{j_{i+1}},\ldots,S_{j_k}))$). By the time the request reaches $t$, all services (except for those that may be optimally placed onto $t$) are allocated to nodes along the path.

*c) Practical Constraints:* The DSMR algorithm provides the globally optimal path and service allocation. However, it requires that all possible service sequences $S_{j_1},\ldots,S_{j_k}$ are listed in the service matrix, which is impractical for large network implementation. Suppose there are $|S|$ different services available in a network and all of them could be combined arbitrarily to form a connection request. If the maximum number of services in any request is $k_{max}$, then a total of $\mathcal{O}\left(|S|^{k_{max}}\right)$ columns would be required in the service matrix. Clearly, that does not provide a level of scalability necessary for large networks.

To address this problem, we discuss an approximate solution to the service placement problem, where only the routing information for no services or single service (i.e., $|S| + 1$ columns in the service matrix) are necessary.

*3) Approximate DSMR:* To reduce the size of the service matrix, we can use an approximation that requires only information about the optimal placement of single services, no matter what sequence of services are requested. The idea is illustrated in Figure 4 for a sequence of three services.

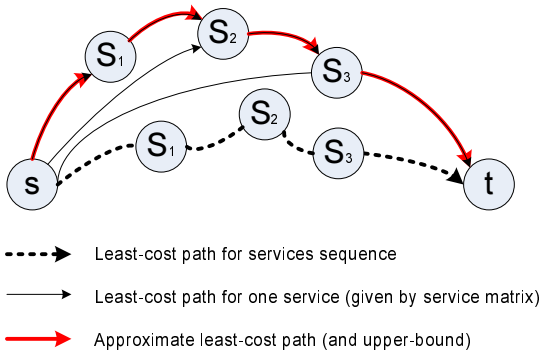


Fig. 4. DSMR Approximation and Path Bounds.

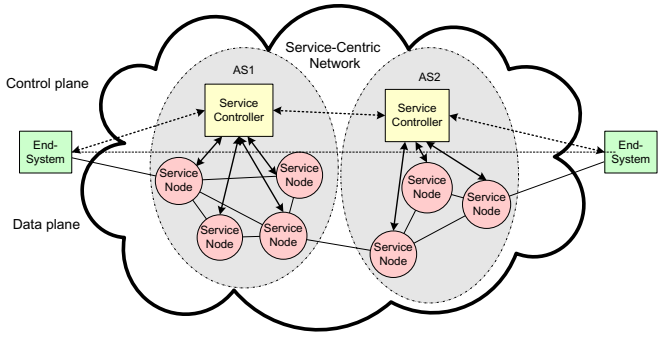


Fig. 5. Architecture of a Service-Enabled Network.

TABLE I
TIME AND SPACE COMPLEXITY OF ALGORITHMS

| Algorithm | Time (route lookup) | Space |
|---|---|---|
| Layered graph | $\mathcal{O}(k(\|E\|+\|V\|)\log(k\|V\|))$ | $\mathcal{O}(k\cdot(\|E\|+\|V\|))$ |
| DSMR | $\mathcal{O}(1)$ | $\mathcal{O}\left(\|S\|^{k_{max}}\cdot\|V\|\right)$ |
| Approx. DSMR | $\mathcal{O}(k)$ | $\mathcal{O}(\|S\|\cdot\|V\|)$ |

The approximate DSMR algorithm processes the sequence of services from request $R=(s,t,(S_{j_1},\ldots,S_{j_k}))$, from back to front. First, the optimal path between $s$ and $t$ for $S_{j_k}$ is determined. Since $S_{j_k}$ is a single service, its optimal path and allocation to node $v_{m_k}$ can be determined from the reduced service matrix, which contains only $|S|+1$ columns. This step corresponds to placing $S_3$ in Figure 4. With $S_{j_k}$ placed, the same service matrix can now be used to determine where to place $S_{j_{k-1}}$ along a path from $s$ to $v_{m_k}$. This step corresponds to placing $S_2$ in Figure 4. The process is repeated until all services have been placed.

Using this approximate placement algorithm, a node can compute an upper bound for the optimal path cost $C(P^{opt})$: The upper bound path, $P^+$, is the path that is determined by the approximate DSMR algorithm. Its cost is the sum of all paths for individually placed services minus the redundant portions of the path. Thus, $C(P^+)=\sum_{i=1}^{k-1}\left(c_s^{S_{j_i}}(v_{m_{i+1}})-c_s^-(v_{m_{i+1}})\right)+c_s^{S_{j_k}}(t)$. Since $c_s^{S_{j_i}}(v_{m_{i+1}}) > c_s^-(v_{m_{i+1}})$, the sum within the expression yields a positive cost.

An important question is how tight the upper bound is, since it determines the quality of the approximate DSMR algorithm compared to the optimal solution (provided by either DSMR or the layered graph algorithm). This question will be evaluated and answered in Section VI-B.

### D. Analytical Comparison of the Algorithms

The time and space complexity of the three algorithms, centralized layered graph, DSMR, and approximate DSMR, are compared in Table I. The layered graph algorithm requires a significant amount of time to perform a route lookup since it needs to create the layered graph on demand and compute the least-cost path. In contrast, DSMR requires only constant time to perform a lookup since the service matrix contains the optimal path for all possible destinations and service combinations. However, the data structure size for DSMR is prohibitively large. Approximate DSMR strikes a good balance between lookup time and space requirement with lookup times proportional to the number of services. The service matrix data structure is also small and proportional in size to the number of services and destination nodes. This comparison, again illustrates the benefits of using a distributed algorithm over a centralized algorithm, but also highlights the need for using approximate DSMR over the original DSMR to limit space requirements.

## V. DISTRIBUTED SERVICE ROUTING PROTOCOL

In this section, we introduce the routing protocol designed to facilitate route selection for service-enabled networks. For scalability, we envision a hierarchical network as shown in Figure 5, with groups of nodes combined into Autonomous Systems (AS). Each AS has a service controller and a set of service nodes. We assume that every AS is small enough for the controller to have a complete view of its topology. For larger ASs, this design can be easily extended to have more than one service controller within an AS.

The protocol can be divided into two different levels: Intra-AS routing which determines the path within the AS, and Inter-AS routing which routes between ASs. For the Intra-AS routing, we use the centralized layered graph algorithm described in Section IV-B. For Inter-AS routing, the distributed algorithm (DSMR or approximate DSMR) described in section IV-C is used.

Next, we move on to the design details of the routing protocol. We divide the discussion into three different stages:

- DSRP Setup: describes how nodes register to their local service controller and how controllers collect information about topology and services offered within their AS.
- DSRP Routing Information Exchange: describes how service matrices are initialized, exchanged and updated.
- DSRP Connection Setup: describes how a request for connection by an end-system is translated into the actual setup of a path.

To keep the initial design of the protocol simple, we make a few reasonable assumptions: (1) every node knows the IP address of its local service controller, (2) every AS has a globally unique ID (AS_ID), (3) all nodes register at the DSRP Setup stage, and (4) all information on the control plane is transmitted using a reliable communication channel (e.g. TCP).

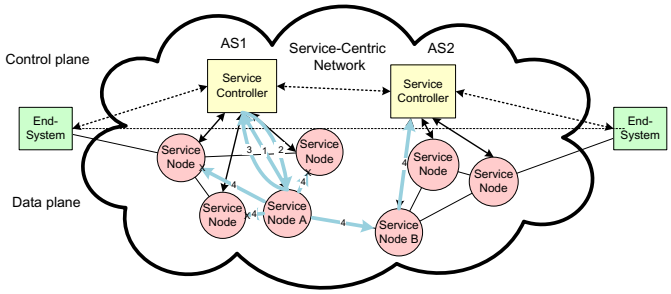


Fig. 6. DSRP Initialization.

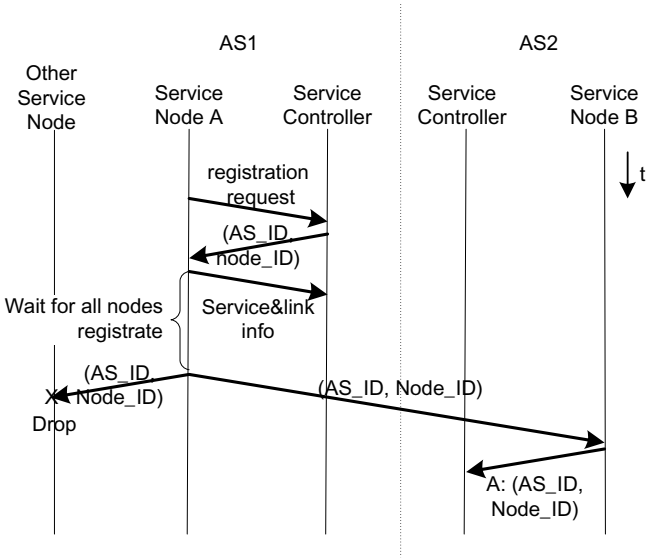


Fig. 7. Space-Time Diagram for DSRP Initialization.

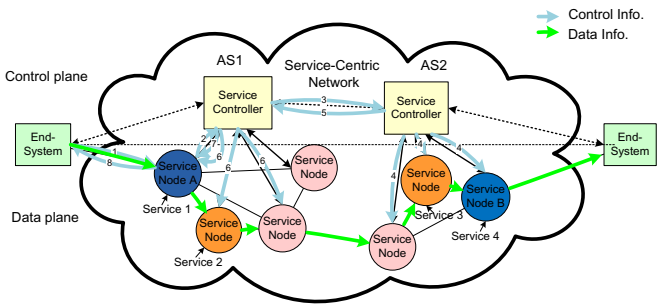


Fig. 8. DSRP Connection Setup.

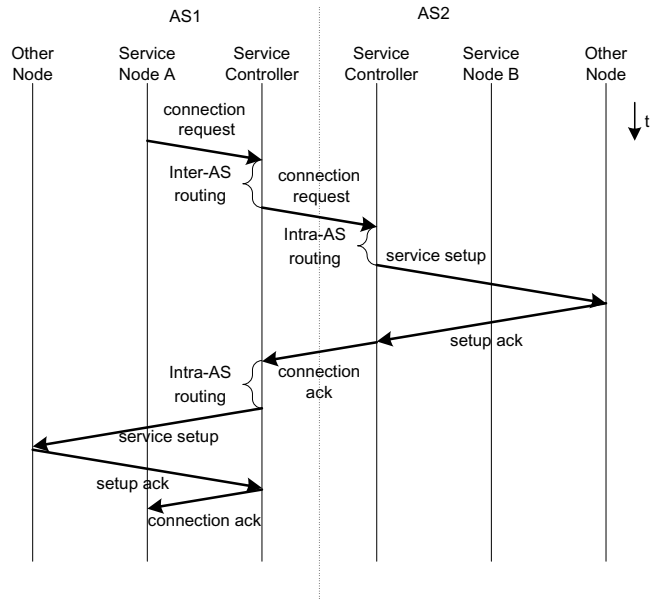


Fig. 9. Space-Time Diagram for DSRP Connection Setup.

### A. DSRP Setup

As illustrated in figures 6 and 7, the DSRP setup stage consists of the following steps.

Step 1: The node (e.g., node A in AS1) sends a registration request to its local controller.

Step 2: On receipt of the registration request, the local controller assigns a locally unique ID (node_ID) to the node, and replies to the request with the (AS_ID, node_ID) pair.

Step 3: The node then sends information about all its interface connections (the IP address pairs of all its links), the services it offers, its processing capacities etc. to its local controller.

Step 4: Finally, the node exchanges its (AS_ID, node_ID) pair with all its neighbors. Information about neighboring nodes from different ASs are passed on to the local controller. This enables the controller to know about the connections to its neighboring ASs.

At the end of the DSRP setup stage, the service controllers have a complete view of their AS's topology and know exactly what services are provides on each node within their AS. In addition, they have also learnt about the nodes connecting to neighboring ASs.

### B. DSRP Routing Information Exchange

After the setup stage, each node probes its neighbors to obtain the corresponding link delays. In addition, the nodes constantly check their CPU utilization to get an estimate of the time to process each of their available services. All this information is periodically reported to their controllers, which now have enough information to run the centralized layered graph algorithm.

The distributed Inter-AS routing is the main challenge and contribution of DSRP, which makes use of the DSMR or the approximate DSMR. The following text describes the initialization, exchange and update of the service matrices.

The service matrix is initialized using the centralized layered graph algorithm after the service controller has enough information about its AS.

After initialization, the service controllers send an OPEN message to establish a TCP connection between each of their neighboring controllers (i.e., service controllers belonging to neighboring ASs). This TCP connection is used to exchange service matrices at the end of pre-defined exchange intervals (e.g., 30 seconds). On receipt of a service matrix from a neighbor, the service controller recalculates a new service matrix. The current matrix is updated only when it is different from the new one. At the end of the exchange interval, the service matrix will be exchanged using an UPDATE message only if it was updated during this current interval. All the

service matrices eventually converge to a stable state and the routing information across the network becomes consistent.

When a service controller detects any change in its AS (e.g., a new node comes up or an existing link goes down or its connection with a neighboring AS changes, etc.), its service matrix is updated accordingly. This change is sent to all the neighboring service controllers who once again recalculate their matrices and exchange them if they get updated. This change too will eventually get propagated through the network.

### C. DSRP Connection Setup

The procedure for connection setup in a service-enabled network is illustrated in figures 8 and 9. When an end-system requests the use of network services, the service node (Node A) to which it is connected sends a connection request to its local service controller (also called source AS controller). On receipt of a connection request, the controller immediately looks up the corresponding entry in the service matrix to determine the services to be performed within its AS and the next AS to which the remaining services are to be passed. This procedure is repeated at the next hop AS controller too, until the destination AS controller is reached. The destination AS controller performs the Intra-AS routing (centralized layered graph algorithm) to determine the local path within the AS and the mapping of services to the local service nodes. The local service nodes along the selected path are notified and hop-by-hop UDP connections are established between them. That is, each node is informed what service(s), if any, it needs to perform on the data and to which IP address and port number the (processed) data is to be sent. After setting up this path, the destination AS sends a connection acknowledgement to the (previous) controller which sent the request. Upon receiving the acknowledgement, the previous AS controller too performs the Intra-AS routing, sets the path up within its AS, and sends a connection acknowledgement to its previous controller. This procedure continues till the connection acknowledgement reaches the source AS controller. The source AS controller too sets up its local service path, and the end-system is finally informed of the successful establishment of the connection through an acknowledgement. This marks the end of the setup stage. The end-system can now send its data traffic through the hop-by-hop UDP connection established and the data will be processed accordingly before reaching the intended destination.

## VI. Evaluation

In this section, we evaluate the performance of the approximate DSMR and demonstrate the scalability and efficiency of DSRP.

### A. Prototype Implementation

We built a prototype of the service-enabled network architecture described in Section V, using Emulab and implemented DSRP. As shown in Figure 10, the prototype consists of 60 nodes organized into 12 ASs. Each AS consists of 1 service controller and 4 service nodes. The service nodes are capable

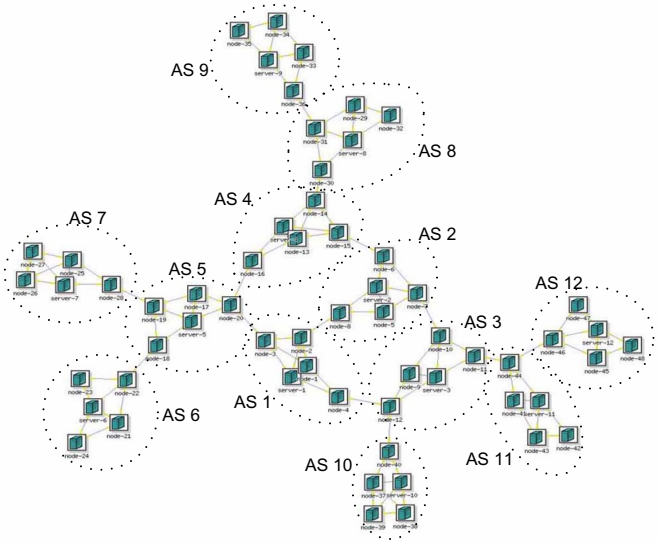


Fig. 10. Topology Configuration in Emulab Prototype.

of performing a maximum of four different services. We use the approximate DSMR to implement the Inter-AS routing and the centralized layered graph for the Intra-AS routing. The multi-threaded controllers and nodes in the prototype have been coded using C.

### B. DSMR Algorithm

To compare the results of the various evaluations of the prototype, we use the results obtained from the simulation of the centralized layered graph algorithm.

As described in section IV, both the centralized layered graph algorithm and the distributed DSMR algorithm give the optimal solution for the service placement problem. However, the approximate DSMR only provides an approximate solution, which may deviate from the optimal solution. To quantify the quality of the approximate DSMR, we first setup connections between all possible combinations of source and destination, with all possible permutations of the 4 different services in the prototype. This gave us a total of 149,760 connections. We compare the path costs for all these connections ($C(P^+)$) to a simulation setup of the centralized layered graph algorithm ($C(P^{opt})$).

We can quantify how many requests are served how well by considering the cumulative distribution function (CDF) shown in Figure 11. The x-axis shows the relative cost of the approximate DSMR solution comparing to that of the optimal path (i.e., $C(P^+)/C(P^{opt})$). We can see that for zero or one service, approximate DSMR indeed provides the optimal solution since $C(P^+) = C(P^{opt})$ for all such connection requests. This result validates that the service matrix exchange (i.e., DSMR) is a correct substitute of the layered graph algorithm. We can also conclude from the figure that the approximation quality of the approximate DSMR deteriorates with the increase of number of services required in the connection request. However, for two services, around 68% of requests are routed with identical cost as the optimal solution. More than 95% of the requests are less than 50% more costly

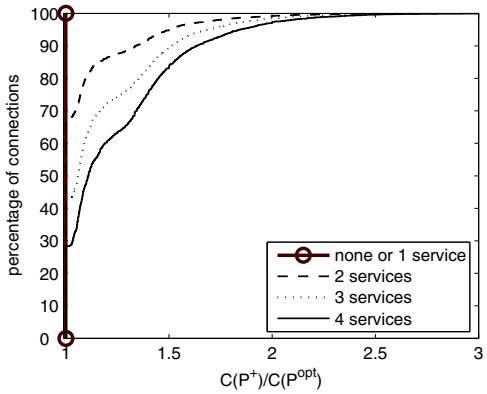


Fig. 11. CDF of Approximate DSMR Path Cost to Optimal Path Cost.

than the optimal path (i.e., $C(P^+)/C(P^{opt}) \leq 1.5$). For four services, these percentages drop as it becomes more difficult to find the optimal path by approximation. Still, nearly 30% of requests are routed optimally, and 95% are less than double the optimal cost.

It is important to note that from the above results, majority (more than 95%) of the connection requests have an approximation of less than double of the optimal cost. For the current Internet architecture, an approximation of an optimal path that yields nearly $2\times$ longer paths would be unacceptable. Since only shortest path routing (without any services) is currently used and optimality in finding the path is expected. However, when multiple services are required for a connection, longer paths and less tight approximations are acceptable. It may be acceptable to route a connection along a longer path in order to avoid the heavy computational overhead that finding the optimal solution with the layered graph algorithm would entail. We should also note that although the entire network may provide tens of services, in practice, the number of services requested for any connection will only be a few.

### C. DSRP Protocol

In this section, we evaluate the scalability and the efficiency of DSRP using the results obtained from the Emulab prototype. The results include the evaluation of the service matrix convergence time across the entire network, the connection setup time, the amount of control information required for connection setup, and the amount of state information maintained in service controllers.

*1) DSRP Service Matrix Convergence Time:* As discussed in IV-C, routing is valid only when the service matrices converge to a consistent state. It should also be noted that the time for convergence is closely related to the size of the network. We hence measure this convergence time as one of the metrics to evaluate the efficiency and scalability of DSRP. For our experiments, we define the convergence time for each service controller to be the time between the first exchange (after initialization) and last update of its service matrix, assuming an unchanged topology.

Table II shows the maximum, average, and minimum service matrix convergence times among all controllers for networks

TABLE II
SERVICE MATRIX CONVERGENCE TIME

| Network Size | | Convergence Time (iterations) | | |
|---|---|---|---|---|
| topology | diameter(hops) | maximum | average | minimum |
| 3 AS | 1 | 2 | 1.33 | 1 |
| 6 AS | 3 | 3 | 2.17 | 2 |
| 9 AS | 5 | 4 | 2.78 | 2 |
| 12 AS | 6 | 4 | 3.42 | 3 |

of different sizes. We use the number of iterations as the unit of the convergence time since the absolute value (e.g., time in seconds) depends on how often the service matrices are exchanged between the controllers (we set this interval as 20 seconds for our experiment). The size of the network is measured by network diameter, which is defined as the longest distance (in hops) between any two ASs in the network. From the table, we can observe that the time to converge increases as the size of the network increases from 3 AS (diameter of 1) to 12 AS (diameter of 6). Even in the worst case (maximum), the time to converge increases linearly. This indicates that the DSRP is indeed scalable and efficient with regards to service matrix convergence. Moreover, we observe that the service matrix convergence time is not only related to the network size, but also is affected by several other factors such as the topology and the distribution of the services among various service nodes.

*2) DSRP Connection Setup:* In this section, we evaluate the performance of the routing protocol in terms of the cost to set up a connection. The evaluation metrics are: connection setup time, amount of control information needed, and amount of state information maintained at controllers. We once again setup connections between all possible combinations of source and destination, with all possible permutations of the 4 different services in the prototype.

*a) Connection Setup Time:* First we evaluate the time it takes to setup the connection for each connection request. This is the time from the moment a node sends out a connection request to its controller to the moment it receives an acknowledgement for the successful setup of the connection. We use the time to establish a TCP connection between the corresponding source and destination pairs as the basis for our comparison.

In Figure 12, y-axis is the connection setup time for DSRP while x-axis is the corresponding TCP connection setup time. From the figure, we can notice that majority of the DSRP connection setup times vary from 1 to several times (within 10) the corresponding TCP connection setup times. To take a closer look, we plot the CDF of this data in Figure 13. We observe that all the TCP connections are established within 150 ms and 95% of that of DSRP are established within 250 ms. Figure 14 shows the CDF of the relative connection setup time of DSRP with respect to that of TCP for requests with different number of services. From the figure, we can see that the relative time increases only by a small margin with increase in the number of services. Even for 4 services, the setup times for 85% of the requests are less than 5 times that of TCP. The figures indicate that the DSRP is indeed efficient and scalable in terms of connection setup time even though so much more

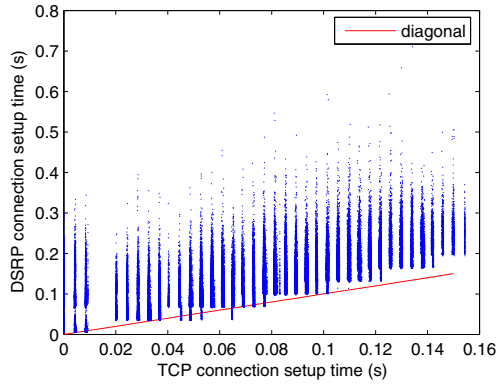


Fig. 12. Connection Setup Time (DSRP vs. TCP). Diagonal indicates equal time for DSRP and TCP.

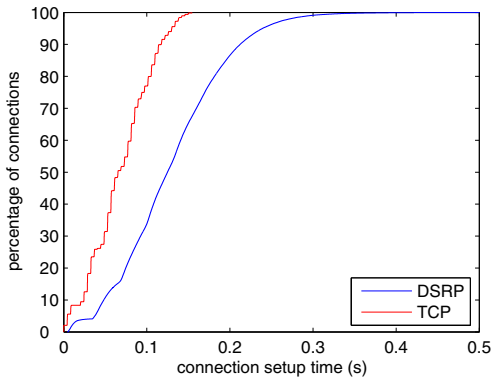


Fig. 13. CDF of Connection Setup Time (DSRP vs. TCP).

work is being done in the control plane for establishment of a connection with service placement.

*b) Control Information:* Next we evaluate the DSRP based on the amount of control information needed to setup a connection. This includes all the messages exchanged between nodes and controllers during various stages of the connection setup. Again, we use the TCP connection setup (messages exchanged during the three-way hand shake procedure) as the basis for comparison. We define a metric called "byte seconds"

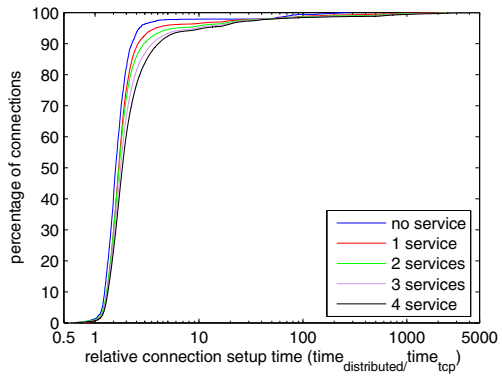


Fig. 14. CDF of Relative Connection Setup Time for Connection Requests with Different Number of Services.

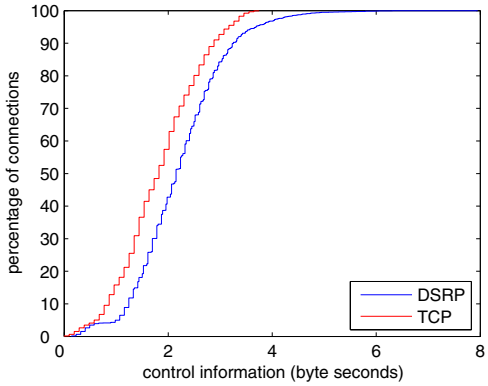


Fig. 15. CDF of Amount of Control Information Used for Connection Setup (DSRP vs. TCP).

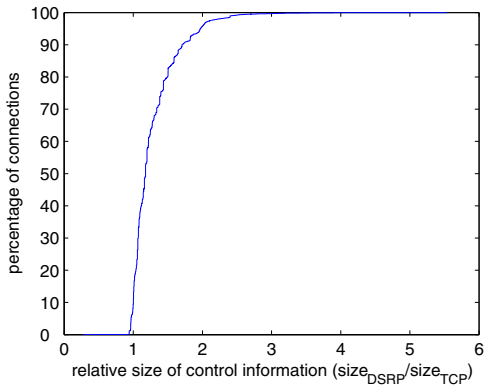


Fig. 16. CDF of Relative Amount of Control Information Used for Connection Setup (DSRP vs. TCP).

(i.e. the size of message multiplied by the time for delivering that message) to evaluate the amount of the control messages. Since this metric considers both the size of the message and the time (and hence distance) for transmitting the message, it gives a better picture of how much resource of the network is utilized by these control messages.

Figure 15 shows the CDF of the amount of control information used by the DSRP comparing to that used by the TCP. The CDF of the relative values is shown in figure 16. From the figures, as expected we observe that the DSRP in general uses more control messages for path setup than TCP. We can notice that the difference between the two sets of data is quite small. Figure 16 shows that for about 85% of the connections, the amount of control information used by the DSRP is less than 1.5 times that of TCP. This proves that the DSRP has low protocol overhead when setting up a connection.

*c) State Information:* Finally we evaluate the amount of state information that needs to be maintained at the controllers for the duration of active connections. For each active connection, every time the connection goes through an AS there is a corresponding record (we call it "entry") maintained at the controller of the AS. The size of an entry is 24 bytes. The entry stores connection information such as flow_ID, the previous AS_ID, the next hop AS_ID, etc. Depending on the topology, routing, and the mapping of the services, there may

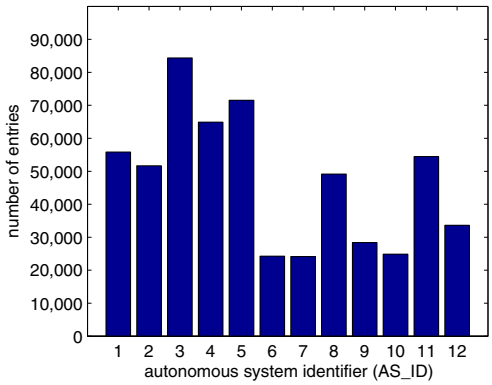


Fig. 17. Number of Entries Maintained on All 12 Controllers.

be more than one entry at a controller for the same connection. In the experiment, we collect the entries from all the service controllers for all the 149,760 connection requests. Since the size of an entry is fixed, we use "number of entries" as the evaluation metric.

Figure 17 shows how many entries are maintained on the controllers when all the 149,760 connections are active at the same time. We observe that the number of entries on each controller depends on the position of the controller inside that network. The AS controllers at the edge of the network have lesser entries (controllers of AS-6,7,9,10.,12) while the AS controllers at the heart of the network have more entries. When all the 149,760 connections are active, the maximum number of entries needed at any controller (controller of AS3) is only 84,352, which is 0.5632 times of the number of active connections. This indicates that the DSRP achieves a balanced mapping for routing and service placement.

## VII. Summary and Conclusion

In this paper, we have presented a distributed protocol for routing in a service-enabled network architecture. The protocol can determine the optimal or near-optimal routing for connection request that require data-path services to be performed in the network. We have discussed DSMR, a distributed algorithm that uses a service matrix to exchange routing information. We have implemented this algorithm in DSRP, a scalable and hierarchical routing protocol. Our prototype implementation on Emulab illustrates the capabilities of our system in terms of fast convergence, efficient connection setup, and low protocol overhead. We believe this work is an important step towards efficiently managing data-path service in future networks.

## Acknowledgements

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

## References

[1] I. Foster and C. Kesselman, Eds., *The Grid – Blueprint for a New Computing Infrastructure*, 2nd ed. Morgan Kaufmann, 2004.

[2] *Cisco Unified Communications and Service-Oriented Network Architecture*, Cisco Systems, Inc., Jul. 2007.

[3] 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.

[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] N. C. Hutchinson and L. L. Peterson, "The x-kernel: An architecture for implementing network protocols," *IEEE Transactions on Software Engineering*, vol. 17, no. 1, pp. 64–76, Jan. 1991.

[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] A. Bavier, M. Bowman, B. Chun, D. Culler, S. Karlin, L. Peterson, T. Roscoe, T. Spalink, and M. Wawrzoniak, "Operating system support for planetary-scale network services," in *Proceedings of the 1st Symposium on Network System Design and Implementation (NSDI '04)*, San Francisco, CA, Mar. 2004.

[8] D. L. Tennenhouse and D. J. Wetherall, "Towards an active network architecture," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 5–18, Apr. 1996.

[9] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Proc. of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.

[10] A. Barbir, P. Reinaldo, R. Chen, M. Hofmann, and O. Hilarie, "An architecture for open pluggable edge services (OPES)," Network Working Group, RFC 3835, Aug. 2004.

[11] E. Guttman, "Service location protocol: Automatic discovery of IP network services," *IEEE Internet Computing*, vol. 3, no. 4, pp. 71–80, Jul. 1999.

[12] S. Y. Choi, J. S. Turner, and T. Wolf, "Configuring sessions in programmable networks," *Computer Networks*, vol. 41, no. 2, pp. 269–284, Feb. 2003.

[13] 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.

[14] B. Gleeson, A. Lin, J. Heinanen, G. Armitage, and M. A., "A framework for IP based virtual private networks," Network Working Group, RFC 2764, Feb. 2000.

[15] *The Open Source Network Intrusion Detection System*, Snort, 2004, http://www.snort.org.

[16] J. C. Mogul, "Simple and flexible datagram access controls for UNIX-based gateways," in *USENIX Conference Proceedings*, Baltimore, MD, Jun. 1989, pp. 203–221.

[17] K. B. Egevang and P. Francis, "The IP network address translator (NAT)," Network Working Group, RFC 1631, May 1994.

[18] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, "Design, implementation and performance of a content-based switch," in *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 1117–1126.

[19] S. Guha and P. Francis, "An end-middle-end approach to connection establishment," in *SIGCOMM '07: Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, Kyoto, Japan, Aug. 2007, pp. 193–204.

[20] I. Baldine, M. Vellala, A. Wang, G. Rouskas, R. Dutta, and D. Stevenson, "A unified software architecture to enable cross-layer design in the future internet," in *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, HI, Aug. 2007.

[21] G. N. R. Rudra Dutta, 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, Jun. 2007, pp. 1899–1904.

[22] S. Ganapathy and T. Wolf, "Design of a network service architecture," in *Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN)*, Honolulu, HI, Aug. 2007, pp. 754–759.

[23] R. Bellman, "On a routing problem," *Quarterly of Applied Mathematics*, vol. 16, no. 1, pp. 87–90, Jan. 1958.

[24] C. Hedrick, "Routing information protocol," Network Working Group, RFC 1058, Jun. 1988.