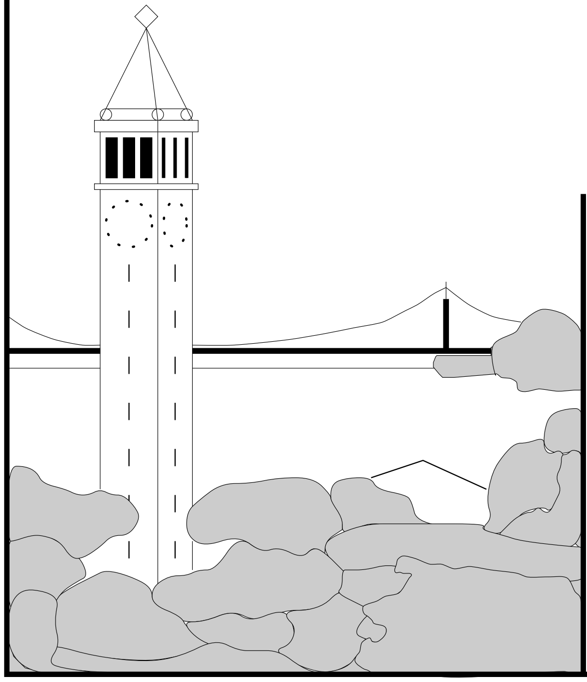
Routing as a Service

Karthik Lakshminarayanan *UC Berkeley*

Ion Stoica *UC Berkeley*

Scott Shenker *UC Berkeley and ICSI*



Report No. UCB/CSD-04-1327

Computer Science Division (EECS)
University of California
Berkeley, California 94720

Routing as a Service

Karthik Lakshminarayanan UC Berkeley
Ion Stoica UC Berkeley
Scott Shenker UC Berkeley and ICSI

Abstract

Many recent proposals have argued for giving end-hosts control over routing in the network to satisfy the growing demands of applications. However, these proposals either run at an overlay level independent of one another, or else lack support for end-hosts to discover the desired routes. In this paper, we propose a network architecture that addresses these limitations. At the basis of our solution lies the idea that specialized route computation should be provided as a *service* and not embedded in the infrastructure. This design allows the routing functionality to evolve without changing the infrastructure. Our architecture consists of three entities: (a) a forwarding infrastructure that enables end-hosts to setup routes, (b) Routing Service (ROSE) providers that compute routes (conforming to application requirements) based on network information that the infrastructure provides, and (c) end-hosts that setup the routes, obtained by querying ROSE, in the infrastructure. We address the issues of trust, scalability of the ROSE architecture, and deployability. We demonstrate the feasibility of our solution by conducting experiments on a prototype deployed on PlanetLab. To illustrate the benefits of our architecture we evaluate two applications: metric-sensitive multicast, and resilient routing.

1 Introduction

In the current Internet architecture, the routing functionality is embedded in the infrastructure with end-hosts having little control over the path that their packets follow. This simple architecture has played an important role in the success of the Internet: it has led to a simple interface for applications, and has provided routing that is "good-enough" for a wide range of popular applications such as e-mail, file transfer and web browsing.

However, the widespread adoption and commercialization of the Internet is posing an increasingly varied set of demands on the Internet. New applications such as Web services and IP telephony have different demands in terms of resilience, reliability, and latency from the infrastructure. To meet these demands, many researchers have proposed giving end-hosts more control over routing decisions. There proposals fall in two broad categories: using overlay networks, and giving end-hosts some control on routing in the Internet.

Overlay networks insert functionality at the application level, thereby circumventing the rigidity of the Internet. Overlay networks have been used for improving reliability and routing performance [6, 37], implementing multicast [14, 11, 18, 19, 25, 32, 36], distributing content [5, 12], and performing data transformation [21, 34]. While very successful, these overlay networks have been independent application-specific efforts; each services a different application (or application requirement) and each requires a large investment of design effort and/or deployment expense.

In contrast, proposals such as loose source routing, and, more recently, TRIAD [13] and NIRA [43] have argued for giving end-hosts more control over routing in the Internet. These proposals allow applications to define their own routes within a shared network infrastructure. While these proposals are not as flexible as overlays (*e.g.*, they cannot control what packets are dropped in the network), using a shared infrastructure has two benefits: (a) packet forwarding is highly efficient, and (b) new applications which require routing control can be immediately deployed. However, none of these proposals address how end-hosts compute the routes in the infrastructure. Having each host probe the network to find such routes is neither efficient nor scalable.

In this paper, we build on these proposals and go one step further by proposing a network architecture in which end-hosts *share* not only the network infrastructure, but also the route computation mechanisms. At the basis of our solution lies the idea that specialized route computation should be provided as a *service* and not embedded in the infrastructure. Our architecture consists of three entities: a forwarding infrastructure, Routing Service (ROSE) providers, and end-hosts. The forwarding infrastructure provides the ability for end-hosts to setup routes, and exports topology and performance related information to the ROSE providers. ROSE providers use this information to compute routes conforming to application requirements. End-hosts query ROSE for routes that satisfy their policy and performance requirements, and setup these routes in the infrastructure.

The fact that routing is provided as a service, and not embedded in the infrastructure, allows the network to evolve with changing needs. As new applications emerge, new routing services can be developed, and old ones can broaden their scope. Separating routing and forwarding functionalities in a modular fashion allows components which implement these to evolve and improve independent of one another.

In summary, this paper argues for providing routing as a service, and presents the design of an architecture to realize this idea. Our design aims to balance the desire of ISPs to control the overlay traffic, and the desire of applications to have more control over the routing decisions. To achieve this, our architecture gives ISPs control on defining the topology of the forwarding infrastructure, and end-hosts, through ROSE, control on selecting the paths restricted to the topology. In the design process, we make the following contributions:

- Enforce trust between different administrative and economic entities, by designing our architecture such that an entity can verify the information or the actions performed by another entity. For example, ROSE would be able to verify the performance information provided by the forwarding infrastructure.
- Propose a simple ROSE architecture that can scale to an infrastructures of up to tens of thousands of nodes.
- Illustrate the benefits of the architecture by implementing two popular applications targeted by overlays: metric-sensitive multicast, and fine-grained failover.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 describes the overall system architecture, and Section 5 presents our solution to enforce trust through verification. Section 6 gives an overview of the implementation, and presents a generic API. Section 7 details the experiments conducted to demonstrate the feasibility and benefits of our architecture. We list our sins and omissions in Section 8, and conclude in Section 9.

2 Related work

The paper touches upon many issues that have separately received attention in the literature.

Routing control. Several proposals have advocated giving end-hosts more control over routing in the network. Nimrod [10] architecture proposed computation of routes by the clients of the network, and give mechanisms for distribution of network maps. Clark *et al.* [15] have argued for giving end-hosts more control over routing as a way of promoting competition among ISPs. In this context, Yang [43] has proposed a solution that allows both senders and receivers to choose routes at the AS level. TRIAD [13] has proposed a name-based routing scheme where end-hosts specify the path across multiple address space domains. To provide resilience, feedback-based [44] routing presents an architecture where edge networks perform measurements to find efficient routes in the network. With the exception of feedback-based routing, none of the proposals address the issue of end-hosts discovering desirable routes. While feedback-based routing addresses this issue, it does not provide a mechanism to share this information across edge networks. Our solution addresses these issues, and, in addition, the issue of how to enforce trust across different entities.

$i3$ [39] provides support for basic communication primitives including mobility, anycast, multicast, and service composition. The notion of triggers and stacks in $i3$ is a richer form of the forwarding primitive introduced in this paper. Our work and $i3$ are largely complementary. While $i3$ focuses on supporting basic communication primitives, we focus on how end-host can discover desirable routes, and how trust can be enforced in the system.

Several companies such as Sockeye [3] and RouteScience [2] develop products for multi-homed organizations to pick their last hop ISP. However, these solutions operate only at the last-hop at a granularity of organizations, not applications.

Finally, our path primitive can be seen as a combination of *loose source routing* and *virtual circuit switching* primitives. However, unlike source routing, our primitive allows the receiver (not only the sender) to control the routing, and, unlike circuit switching, the path is inserted by end-hosts, and not by the infrastructure.

Division of functionality. The division of data and control plane functionalities between the network and end-hosts plays a central role in any network architecture. At one extreme, Active Networks [42] allow end-hosts to insert arbitrary functionality in the forwarding path. ESP [9] provides a middle-ground by allowing packets to create temporary state at routers via short, pre-defined computations. Our approach explores a different point in the design space as it allows end-hosts to modify only the control plane in the network.

We are not the first to argue for the separation of data and control planes. Generalized Switch Management protocol (GSMP) [17] proposes external switch controllers to establish and maintain paths in an ATM, frame relay or MPLS switch. The IETF ForCES [27] working group addresses forwarding and control element separation for IP forwarding devices in a small area system. Bandwidth broker [7] computes routes based on QoS requirements within a single domain. Akamai [8] implements route computation as a separate component, though the details of the patent are not public. In contrast to these approaches, we target a large-scale infrastructure where the control plane and data plane are implemented by *different* administrative entities, which raises issues of trust and scalability.

Shared infrastructures. PlanetLab [33] provides an infrastructure that is shared among various overlays. Since each overlay runs in an independent *slice*, the sharing is mainly at the hardware level. While PlanetLab is a perfect vehicle for research as it gives users complete control on overlay nodes, it raises security and efficiency concerns that may make it inappropriate for commercial use.

SPAND [38] performs passive network performance discovery to improve the performance of adaptive Internet applications. However, it does not consider trust issues. To address the problem of duplicate measurements across over-

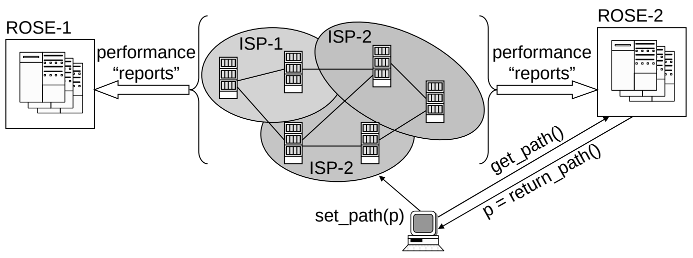


Figure 1: The main components of the architecture: (1) forwarding infrastructure, (2) one or more ROSE providers, and (3) end-hosts.

lay networks, Nakao *et al.* [29] have proposed a routing underlay that provides coarse-grained static information to the higher layers. However, users still need control over overlay nodes to perform packet forwarding and fine-grained measurements.

Indirect measurement techniques. Several measurement and monitoring techniques have been recently proposed in the context of the Internet [30, 20, 23]. These systems are more general in that they aim to estimate performance characteristics between any two hosts in the Internet (instead of between overlay nodes). However, they are either limited to estimating the delay alone, or require an infrastructure of their own [20] for measurements.

3 Architecture Overview

We first present a brief overview of our architecture, which, as shown in Figure 1, consists of three components:

1. *Forwarding infrastructure.* We envision a global infrastructure consisting of high-bandwidth nodes, owned and controlled by the ISPs, interconnected with one another in an overlay graph. The edges of the subgraph, called *virtual links*, are logical unidirectional paths along which packets are forwarded via IP. Paths can be setup across the network only using a combination of virtual links. The infrastructure monitors the performance of the virtual links, exports this information to ROSE, and allows end-hosts to set-up tunnels (data paths) computed by ROSE.
2. *Routing Service.* ROSE is a third-party entity that gathers network information from the forwarding infrastructure, and uses this information to compute application-specific paths between any two nodes in the infrastructure. There can be more than one ROSE provider in the system, in the same way there is more than one search engine for the WWW. As advocated by Clark *et. al.* [15], this design fosters competition among providers by giving choice to users
3. *End-hosts.* End-hosts which desire special routing query ROSE to obtain paths that satisfy specific application constraints, and then insert these paths into the forwarding infrastructure. Applications that do not require special routing simply use the default Internet routing.

3.1 Benefits

ISP control. On one hand, ISPs want more control over the traffic they forward. Overlay traffic, being outside ISPs' control, and more importantly, less predictable than traditional traffic, makes the task of performing traffic engineering difficult for ISPs. On the other hand, end-hosts want flexible control over the paths taken by the packets, and hence resort overlays. In our design, we try to balance these conflicting interests by giving the ISPs control over defining the forwarding topology, and giving end-hosts, through ROSE, control over selecting the paths restricted to the topology. In contrast, today, end-hosts have no control over routing in the Internet, and the ISPs have little control over the overlay traffic.

In general, each ISP will choose the virtual links between its nodes to reflect its performance and perhaps policy[1] constraints. For instance, an ISP may choose links within its domain to minimize the impact of sudden changes in the traffic along these links on the rest of its traffic. The ISPs may add new virtual links upon ROSE' request as long as it does not conflict with the ISP's constraints. In this paper, we do not discuss how the network infrastructure is configured and maintained. Since ROSE computes paths only along these virtual links, ISPs can isolate the overlay traffic from the rest of their traffic better.

Better performance. End-host based overlay solutions are inefficient as end-hosts sit typically behind high-latency low-bandwidth connections (as compared to the routers in the network). The ability of end-hosts to perform performance-based routing in the infrastructure would be beneficial for both applications, as they get better performance, and for ISPs, as they can avoid the extra traffic generated by end-hosts acting as routers.

Ease of deployment of applications. Our infrastructure will lower the barrier of deployment for applications that require more sophisticated routing than the default Internet routing. For example, our infrastructure might make large scale broadcast of live-media feasible. Accelerating the rate of introduction of new applications would be beneficial for both third-parties, which develop these applications, and ISPs, which can provide more services.

3.2 Challenges

We briefly outline the challenges that face our proposal.

Scalability of ROSE. ROSE must be able to receive measurement information from many infrastructure nodes, compute efficient paths quickly, and serve requests from the end-hosts. We address these issues by making ROSE distributed

[1]We do not refer to BGP policies in particular.

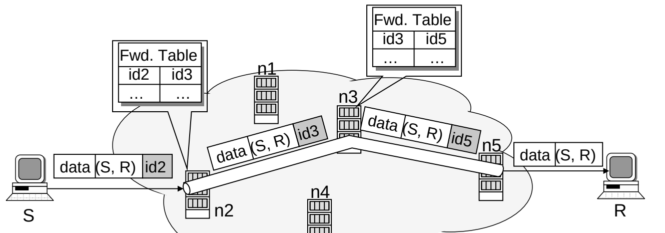


Figure 2: Host S sends packets via tunnel (id_2, id_3, id_5) to host R. Tunnel could be inserted by either host. Inserting the tunnel is equivalent to setting up forwarding state (id_2, id_3), and (id_3, id_5). Note that IPADDR$(id_2) = n_2$, IPADDR$(id_3) = n_3$, and IPADDR$(id_5) = n_5$.

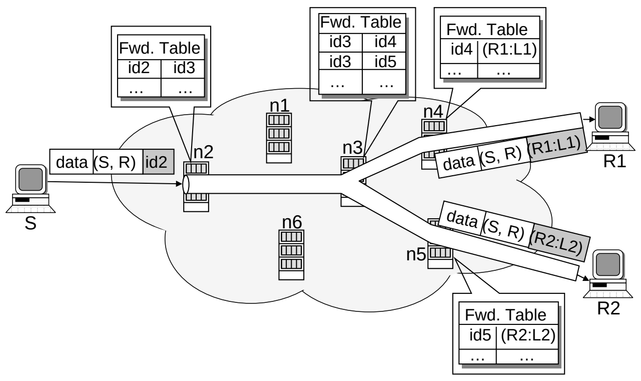


Figure 3: Node n_3 replicates all packets with ID id_3.

and partitioning the infrastructure among the many nodes of a ROSE provider. We detail these mechanisms in Section 4.2.

Trust and Security. A key challenge with any open system is how to make sure that all components are well-behaved. For example, how can we make sure that the information provided by the infrastructure to ROSE is accurate, or that the paths inserted by end-hosts are loop free? We address these questions by carefully designing the protocols such that a component can verify the information provided by another component. In particular, ROSE is able to verify the performance measurements reported by the infrastructure. In turn, the infrastructure ensures that the paths inserted by end-hosts obey certain crytographic constraints. Furthermore, ROSE signs the paths it returns to the end-hosts, and the infrastructure verifies the signatures upon path insertion. Thus, the infrastructure can make sure that all paths it inserts were computed by ROSE, which, in general, is a more trusted entity than an end-host. We address this topic in detail in Section 5.

Deployment. A major hurdle that many architecture proposals face is the barrier to deployment. While we do not claim that sweeping changes would happen to the Internet overnight, we do believe that the following incentives for both ISPs as well as the end-hosts are compelling enough for a possible deployment: (a) ISPs can attract existing overlay traffic to this infrastructure, where they have more control, (b) efficient replication and forwarding reduces ISP traffic, (c) end-hosts can receive better performance than overlay approaches, and (d) barrier to use for end-hosts is low.

4 Architecture Entities

In this section, we present the details of the architecture outlined in the previous section. While most of the mechanisms we use are straightforward, they nevertheless demonstrate the feasibility of providing routing as a service.

4.1 Forwarding Infrastructure

Each node in the infrastructure performs two basic operations. First, it performs performance measurements to each of its neighbors. These measurements include the round-trip time (RTT), loss rate, and available bandwidth of each of its virtual links. Second, nodes allow hosts to set-up paths (tunnels) and forward packets sent along these paths.

4.1.1 Paths and packet forwarding

The mechanisms for path maintenance and packet forwarding are similar to the mechanisms employed by MPLS [35], and ATM circuit switching. However, our infrastructure is open in that it allows potentially untrusted entities to insert paths, and performs packet forwarding at the overlay instead of the networking layer.

Each infrastructure node maintains a forwarding table whose entries are inserted and maintained (using a soft-state protocol) by end-hosts. Each entry in the forwarding table is of the form (id_1, id_2), where id_1 and id_2 are IDs. An ID is of the form *IPaddr:label*, where *IPaddr* represents the IP address of the infrastructure node where the entry is stored, and *label* is a locally unique 128-bit value. Let IPADDR(id_1) denote the first field of id_1, and LABEL(id_1) denote the second field of id_1. Then inserting a path (tunnel) along nodes $n_1, n_2, \ldots, n_k$, is equivalent to inserting forwarding entries $(id_1, id_2), (id_2, id_3), ..., (id_{k-1}, id_k)$, where IPADDR$(id_i) = n_i$, $(1 \le i \le k)$. Figure 2 shows a tunnel through nodes n_2, n_3, n_5. Establishing this tunnel requires node n_2 maintain forwarding entry (id_2, id_3), and node n_3 maintain entry (id_3, id_5).

To send an IP packet along a tunnel with the starting point id_1, a sender encapsulates the IP packet into an overlay packet with ID id_1, and sends it to IPADDR(id_1). Packet forwarding in the infrastructure is very similar to that in MPLS. When a node IPADDR(id_1) receives a packet, it performs a lookup for id_1, replaces id_1 with id_2 in the packet's header, and forwards the packet to IPADDR(id_2). If an infrastructure node does not find any match for the packet's ID, the node decapsulates the packet and sends it via IP. Figure 2 shows how a packet sent by host S to host R is forwarded along (n_2, n_3, n_5). When a virtual link fails, overlay packets are decapsulated and forwarded via IP. Applications that need improved failover protection can use explicit backup paths using a disjoint set of virtual links (see Section 7.1.2).

4.1.2 *Constraining forwarding entries*

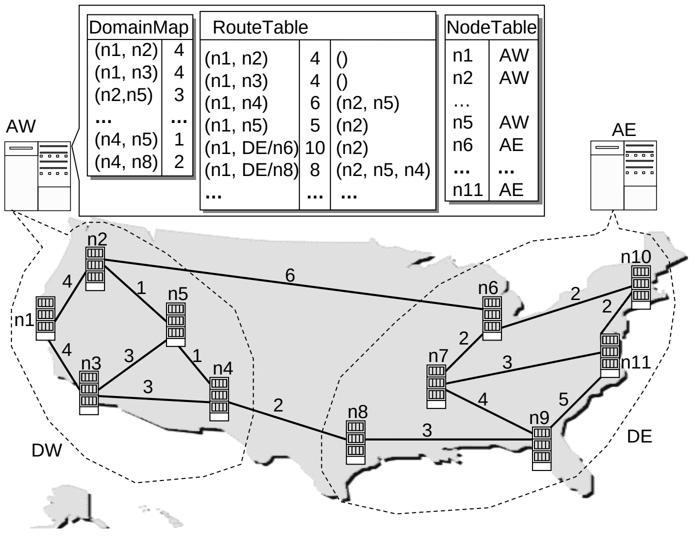
To avoid DoS attacks (*e.g.,* A inserts a path carrying a high bandwidth stream ending at B), we do not allow an end-host to insert a path that terminates at another end-host. Our implementation uses nonce-based challenges to ensure this.

As proposed by Adkins *et al.* [4], we use cryptographic techniques to constrain the identifiers in the forwarding entry. More precisely, given an entry (id_1, id_2), either $\text{LABEL}(id_1) = h_l(id_2)$ or $\text{LABEL}(id_2) = h_r(id_1)$, where h_l and h_r are one-way cryptographic hash functions. It is shown in [4] that these constraints do not sacrifice flexibility of paths that can be constructed, but guarantee that (i) construction of undesirable topologies such as loops is hard, and (ii) eavesdropping on the traffic of other hosts is hard, under cryptographic hardness assumptions. It is this constraint that restricts the API given in Table 1, *i.e.,* an end-host cannot retrieve a path between two completely specified IDs.

4.1.3 *Multicast*

The forwarding infrastructure supports multicast by simply allowing multiple forwarding entries with the same source ID. Suppose a source advertises an ID, id_{src} to which it sends packets. Then each receiver will query ROSE for a path (optimized according to a given metric) from id_{src} to itself. Figure 3 shows an example where source S advertises id_2, and receivers R_1 and R_2 insert paths $(id_2, id_3, id_4, (R_2 : L_2))$, and $(id_2, id_3, id_5, (R_2 : L_2))$, respectively. Node n_3 maintains two forwarding entries: (id_3, id_4) and (id_3, id_5). Upon receiving a packet with ID id_3, n_3 replicates the packet and sends a copy to $n_4 = \text{IPADDR}(id_4)$, and a copy to $n_5 = \text{IPADDR}(id_5)$. The hash-based constraint imposed on the forwarding entries would ensure that sub-paths traversing the same nodes overlap. We assume that all paths of a tree originate at a common source ID id_{src} and that all paths are constrained using either h_l only or h_r only. To prevent DoS attacks on end-hosts, we restrict multicast tunnels to terminate at end-hosts only (as shown in Figure 3). In the absence of this constraint, constructing a tree with all k leaves terminating at the infrastructure, and sending a packet addressed to E on this tree would result in k copies send to E.

We make two observations. First, out-degree of a node in a multicast tree is bounded by the out-degree of that node in the overlay topology. Since the ISPs determine the topology, they can easily bound the number of replicas generated by an infrastructure node. Second, if IP multicast is available in a subnetwork S, one can make use of this by terminating the multicast paths at the edge of S, and then using IP multicast within S.



DomainMap	
(n1, n2)	4
(n1, n3)	4
(n2,n5)	3
...	...
(n4, n5)	1
(n4, n8)	2

RouteTable		
(n1, n2)	4	()
(n1, n3)	4	()
(n1, n4)	6	(n2, n5)
(n1, n5)	5	(n2)
(n1, DE/n6)	10	(n2)
(n1, DE/n8)	8	(n2, n5, n4)
...	...	...

NodeTable	
n1	AW
n2	AW
...	
n5	AW
n6	AE
...	...
n11	AE

Figure 4: Example of ROSE consisting of two servers AW and AE. AW is responsible for infrastructure nodes $n_1, n_2, \ldots, n_5$; AE is responsible for nodes $n_6, n_7, \ldots, n_{11}$. The fi gure shows the relevant state maintained by AW.

4.2 Routing Service (ROSE)

In response to requests from end-hosts, ROSE returns application-specific paths between any two nodes in the infrastructure. For this, ROSE maintains a performance map of the infrastructure using the measurement reports provided by the infrastructure nodes. In addition, ROSE can perform measurements on its own, as discussed in Section 5. Augmenting the information that the infrastructure provides might be useful when ROSE wants to measure a metric that the infrastructure does not provide.

In the rest of this section, we present the design of ROSE. The design is generic in that it is not optimized for a particular metric. We expect that metric-specific optimizations to make each instantiation of ROSE more scalable and efficient.

In the case of an infrastructure with no more than few tens of nodes, ROSE can maintain the map of the entire infrastructure at a single server. To handle a large number of requests, ROSE can simply replicate this server. However, as the infrastructure becomes larger, a single server can no longer maintain the map of the entire infrastructure. We address this problem next. In the ensuing discussion, we target our design to a network with $O(10,000)$ nodes. We believe this is a reasonable size, as we expect that only applications with specialized routing need to use this infrastructure. For comparison, the number of nodes in the Akamai network falls in this ballpark.

To achieve scalability, we use a well-known technique, *graph partitioning*. ROSE partitions the set of infrastructure nodes into different *domains*[2], with a different set of replicated

[2]This is different from the administrative domains in the Inter-

servers assigned to each domain. Each server maintains information about all virtual links within its domain, and a subset of virtual links to other domains. More precisely, given any two domains D and D', ROSE ensures the existence of at least k pairs of nodes $n_1 \in D$ and $n_2 \in D'$ such that there is a virtual link from n_1 to n_2. Note that in this design, it is possible to compute a path between any two nodes n_1 and n_2 by contacting no more than two ROSE servers, one that is responsible for the domain of n_1, and another responsible for the domain of n_2. A redundancy factor of k is chosen to ensure that there are multiple paths between two domains.

The quality of the paths computed by ROSE is determined by the partition of the graph. In our implementation, we use METIS [26], a serial graph partitioning tool, to divide the graph. While, as shown in Section 7, our partition method works reasonably well in the case of a small infrastructure, there is room for improvement. We leave the development of algorithms that optimize the partition for specific metrics to future work. An important observation is that our modular design allows the partitioning algorithms to evolve without changes to the infrastructure.

4.2.1 Path Computation

A server responsible for domain D maintains (see Figure 4):

1. The entire map of domain D, called DomainMap. This map stores the performance metrics (*i.e.,* RTT, loss rate, available bandwidth) associated with every out-going virtual link of a node in domain D. Note that DomainMap also includes also virtual-links from nodes in domain D to nodes in other domains.
2. A routing table, called RouteTable, containing the "best" paths between all nodes in DomainTable. This set of nodes includes, besides all nodes in D, nodes in the other domains to which nodes in D have virtual links. The meaning of "best" depends on the metric being optimized (*e.g.,* lowest latency path).
3. A list of all nodes in the infrastructure, and the list of ROSE servers responsible for each of these nodes. This data structure is called NodeTable.

Consider a ROSE server s that receives a query for a path between infrastructure nodes n_1 and n_2. Assume s is responsible for domain D, and n_1 belongs to D. If n_2 belongs to D as well, then s consults its RouteTable and returns the best path (*e.g.,* shortest path) between n_1 and n_2. If n_2 belongs to another domain D', server s (a) retrieves from its RouteTable a set of k paths from node n_1 to k nodes in domain D', (b) consults NodeTable to find a server s' responsible for domain D', and (c) sends the set of partial paths computed at step (a) to s'. In turn, s' uses these paths to compute a good path from n_1 to n_2, and returns it to the querier.

net.

Consider the example in Figure 4 where ROSE partitions the infrastructure into two domains, DW covering nodes $n_1, n_2, \ldots, n_5$, and DE covering nodes $n_6, n_7, \ldots, n_{11}$. ROSE consists of two servers AW and AE responsible for domains DW and DE respectively. Assume an end-host queries AW for a low cost path from node n_1 to n_{11}. Upon receiving this query, AW consults its RouteTable and returns the two paths it finds to D_2, (n_1, n_2, n_6) and $(n_1, n_2, n_5, n_4, n_8)$. Next, AW sends both paths to AE, which completes the paths with the shortest paths it knows to n_{11}. The two resulting paths are $p_1 = (n_1, n_2, n_6, n_{10}, n_{11})$, and $p_2 = (n_1, n_2, n_5, n_4, n_8, n_9, n_{11})$. Since the cost of p_1 is 14, while the cost of p_2 is 16, AE returns p_1 to the querier.

A ROSE server typically maintains separate RouteTables for each performance metric, *i.e.,* delay, bandwidth, and loss rate. The server periodically recomputes all its RouteTables.

4.2.2 Scalability

We now discuss the scalability of ROSE in terms of memory, computation, bandwidth requirement and ability to support end-host queries. For simplicity, we assume that there are $M = \sqrt{N}$ domains, and each domain has $O(\sqrt{N})$ nodes. In each case, we perform back-of-the-envelope calculations for an infrastructure with $N = 10,000$ nodes.

Memory. Both NodeTable and DomainMap data structures require $O(N)$ memory space. Furthermore, assuming that the length of a path between two nodes is $O(\log(N))$, RouteTable requires $O(N \log(N))$ memory space (there are $O(N)$ pairs of nodes in each domain). For a forwarding infrastructure with $N = 10,000$, a ROSE server needs to maintain only $O(100,000)$ entries.

Computation. The most expensive computation performed by a ROSE server is computing RouteTable. This computation is equivalent (at least in the case of the latency) to solving the all-pairs shortest path problem. By using fast matrix multiplication algorithms [40], we can compute all-pair shortest paths between the nodes in a domain in $O((\sqrt{N})^{2.8}) = O(N^{1.4})$ time. Using optimized BLAS, computation for $M = 100$ nodes takes just 0.57 seconds on a Pentium 2.2 Ghz system [1].

Bandwidth. Assume each node reports every T sec, and that all reports from a node fit in one packet.[3] Then, a ROSE server needs to handle $O(\sqrt{N})$ packets every T seconds. If $T = 1$ sec, and assuming the size of a packet is $1,500$ bytes, then each ROSE server needs to handle of the order of 1.2 Mbps. As an optimization to reduce the bandwidth of measurement report traffic, reports can be suppressed when the measurement value does not change by more than a small factor ϵ. In addition, we need to account for the overhead incurred by the ROSE for monitoring virtual links on its own.

[3]We used 10 bytes to represent measurements to one node. We can pack information of > 100 nodes in one packet.

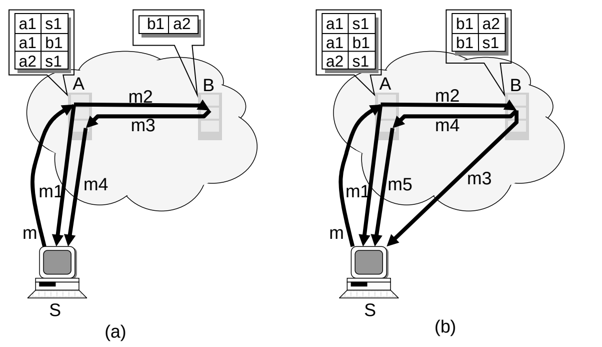


Figure 5: Communication pattern used to measure the (a) round-trip time (RTT) between two nodes A and B, and (b) loss rates along virtual links $(A \rightarrow B)$, and $(B \rightarrow A)$.

Query rate. For processing the queries generated by the end-hosts, we replicate each ROSE server S into a cluster of c nodes. Each query to S is sent at random to one of these c nodes. Let the average lifetime of an information retrieved (*e.g.,* , time for a path that ROSE returns to become stale) is t and Q distinct queries are seen in time t. Since each query hits atmost 2 ROSE servers, the average query rate to a node, N_q is less than $(2*Q)/(t*M*c)$. If $c = 20$ and $N = 10,000$, then 10 million queries performed by the clients in a period corresponding to a refresh period of 10 seconds would result in only 1000 queries/second at each node on average. Typical query messages are very short and replies are moderately short, which in our implementation are, on an average, 30 and 200 bytes respectively. Hence, bandwidth required per node is only 1.6Mbps upstream and 240 kbps downstream.

4.3 End-hosts

End-hosts query ROSE for paths between two nodes in the forwarding infrastructure, and then insert these paths (tunnels) in the infrastructure.[4] We assume that each end-host e knows at least one node in the infrastructure that is close to it. We call these infrastructure nodes, *home* nodes of end-host e. The home nodes can be obtained from ROSE when the end-hosts sign up for the service. To establish a tunnel to an end-host e, one has to obtain the address of e's home node. We rely on DNS (or DHT functionality if available) to store the IP address to home node mapping. Alternatively, one can contact node e and get the home node directly from e (although this would involve extra overhead). For viability reasons, we adopt the latter approach in our implementation.

5 Enforcing Trust By Verification

One of the main challenges of any open distributed system is ensuring that various components are well-behaved. For example, how can ROSE make sure that the reports from the forwarding infrastructure are accurate? Or how can the forwarding infrastructure make sure that the paths inserted by an end-host are well-formed (*e.g.,* they don't contain loops)? In this section, we address these problems.

The main idea behind our solution is to enable each component to verify the information received from another component. We believe that the ability of a component to verify another component, coupled with potential administrative and economic punishments triggered when the verification fails, would create incentives for each component to behave correctly. Figure 1 shows the information flow between various components in our systems:

1. Forwarding infrastructure provides reports to ROSE.
2. ROSE provides paths to end-hosts.
3. End-hosts insert paths into the infrastructure.

Now, we discuss (1) how ROSE can verify the reports from the infrastructure, and (2) how the infrastructure can verify an inserted path, and (3) how end-hosts can verify the paths returned by ROSE.

5.1 Verifying Reports from Infrastructure

We assume that reports provided by the forwarding infrastructure contain the round-trip time (RTT), the unidirectional loss rate, and the available bandwidth along each virtual link maintained by the infrastructure. Next, we show how ROSE can independently measure these metrics between any two nodes in the infrastructure. In doing so, ROSE relies exclusively on the ability of hosts to insert paths in the infrastructure; ROSE does not require any additional primitives to perform these measurements.

In addition to verification, these mechanisms can also be used to measure information that the infrastructure does not provide. For instance, if the infrastructure provides average RTT, but ROSE needs 95-percentile RTT, then ROSE can perform measurements on its own.

5.1.1 Round-Trip Time

Consider the problem of estimating the RTT between two arbitrary infrastructure nodes A and B.[5] Figure 5(a) illustrates the technique used by a host S to perform this measurement. Let a_1 and a_2 be two IDs associated to node A, b_1 be an ID associated to node B, and s_1 be an ID associated to host S. S first inserts paths (a_1, s_1) and (a_1, b_1, a_2, s_1), and then sends periodic probes to ID a_1. Figure 5(a) shows the forwarding table of each node after these paths are inserted, and the corresponding packets. Upon matching a_1, the probe packet is sent back to S, while a replica of the probe packet is sent along path (a_1, b_1, a_2) and finally back to s_1. The RTT be-

[4]Note that ROSE computes optimal paths between the infrastructure nodes only.

[5]Measuring the one way delay between A and B requires the clocks of the two nodes to be synchronized. Thus, measuring the one way delay is hard even assuming full control over the two nodes.

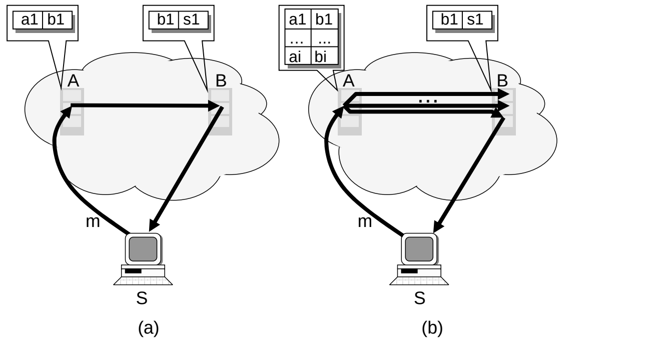


Figure 6: Communication pattern used to estimate the available bandwidth along virtual links $(A{\rightarrow}B)$.

tween A and B is then computed as the difference in the time between receiving copies m_4, and m_1, respectively.

5.1.2 *Loss Rate*

To measure the *unidirectional* loss rate between two nodes A and B, we use a setup similar to the one used to measure the RTT. The only difference is that S also inserts path (b_1, s) (see Figure 5(b)).

Let m be a probe sent by host S to ID a_1. For ease of explanation, in Figure 5(b), we label the copy of the probe along each virtual link of the path. Then S concludes that there was a loss from A to B (*i.e.*, m_2 was lost) if S receives m_1 but does not receive m_3 and m_5.

We now show that under the assumption that (i) the loss probability on each virtual link is $p \ll 1$ and (ii) the loss probabilities on virtual links are not correlated, the probability of false positives, *i.e.*, the probability that S incorrectly decides that m_2 was lost, is $O(p^2)$. Let $P(m_i)$ denote the probability that m_i is lost. Then the probability of false positive, P, is equal to the product of the probabilities of the following events: m_2 is not lost, either m_4 or m_5 are lost, and m_3 is lost.

$$\begin{aligned} P &= (1 - P(m_2))(1 - [1 - P(m_4))(1 - P(m_5)])P(m_3) \\ &\simeq 2p^2 \end{aligned} \tag{1}$$

Though S can compute the loss rate on the reverse link, $(B{\rightarrow}A)$, by inverting the communication pattern shown in Figure 5(b), S can estimate this loss rate without any additional measurements. In particular, while measuring the loss rate for m_2, S also records the following two events:

1. the receipt of m_3 but not of m_5, and
2. the receipt of m_3 or m_5 but not of m_1.

Let f_1 be the frequency of occurrence of event 1, and f_2 be the frequency of occurrence of event 2. Then f_1 estimates the loss rate on virtual link $(B{\rightarrow}S)$ while f_2 estimates the loss rate on virtual link $(A{\rightarrow}S)$. Finally, S estimates the loss rate on $(B{\rightarrow}A)$ as $f_1 - f_2$. Note that this estimation procedure assumes that the losses on links $(B{\rightarrow}A)$ and $(A{\rightarrow}S)$ are not correlated. Finally, it can be shown that if the loss probability on each virtual link is $O(p)$, the probability of false positives for both f_1 and f_2 is $O(p^2)$.

5.1.3 *Available Bandwidth*

Unlike RTT and loss rate, the notion of available bandwidth is not well defined. Various definitions of the available bandwidth on a path include the equivalent throughput of a TCP flow, the max-min fair share on the path and excess capacity available on the path. Furthermore, there is a wide variety of bandwidth estimation algorithms ranging from ones based on the TCP equation [31], to ones based on sophisticated active measurements techniques such as Pathload [24]. Due to this large variety, and the lack of general agreement on the semantics of the available bandwidth, we do not commit ourselves to a particular estimation algorithm. Instead, we aim to present a solution for the following more general question: assuming that node A uses a bandwidth estimation algorithm AB_{est} on link $(A{\rightarrow}B)$, then can a remote server S estimate the available bandwidth on $(A{\rightarrow}B)$. We note that while our solution won't work with any estimation algorithm, it is general enough to support active estimation algorithms such as Pathload and certain TCP-based algorithms (as we will discuss in Section 7.3.3).

Figure 6 illustrates our approach. First, S inserts a tunnel (a_1, b_1, s_1), and then uses the same estimation algorithm AB_{est} to estimate the available bandwidth along path $S{\rightarrow}A{\rightarrow}B{\rightarrow}S$. Let bw_1 be the estimate bandwidth at the end of this procedure. The problem is that bw_1 need not necessarily represent the available bandwidth on $(A{\rightarrow}B)$, as it can also reflect the available bandwidth on $(S{\rightarrow}A)$ or $(B{\rightarrow}S)$. To disambiguate between these possibilities, S performs the following iterative algorithm:

1. $i = i + 1$ (i is initialized to 1);
2. Set-up a new path from A to B, (a_1, b_i);
3. Re-run the estimation algorithm along $S{\rightarrow}A{\rightarrow}B{\rightarrow}S$ by sending packets with ID a_1;
4. If $i * bw_i \simeq (i-1) * bw_{i-1}$, report available bandwidth of $(A{\rightarrow}B)$ as $i * bw_i$. Otherwise, go to step 2.

In step (2), S creates one more replica for each packet on virtual link $A{\rightarrow}B$. When link $(A{\rightarrow}B)$ becomes the bottleneck link, the available bandwidth on $(A{\rightarrow}B)$ becomes $i * bw_i$, where bw_i is the bandwidth estimated by S at iteration i.

We make several observations. First, the convergence of the above algorithm can be improved by doubling the number of replicas on $(A{\rightarrow}B)$ at each iteration, and starting iteration i with an initial bandwidth of $bw_{i-1}(i-1)/(i+1)$. Second, this approach will not work for algorithms, such as Spruce [41], that use carefully spaced packets to estimate the available bandwidth. This is because the inter-arrival times would be affected by the replication operation at node A, and the additional delay variation introduced by link $(B{\rightarrow}A)$. Finally, larger RTTs on the indirect measurement path might be another source of inaccuracies.

5.2 Verifying Paths Inserted by End-Hosts

Implicit verification. All the paths that end-hosts insert must be cryptographically constrained. This provides an implicit verification mechanism for the infrastructure, as we are guaranteed that malicious topologies cannot be formed.

Explicit verification. While implicit verification provides guarantees on topologies at the level of IDs, one can still construct *confluences* terminating at an infrastructure node. For example, inserting paths $(id_1, \ldots, id_{n_i})$, for $i = 1 \ldots k$, where all id_{n_i} reside on the same infrastructure node would cause all replicas of the multicast to go through the same infrastructure node.

To verify these cases, an infrastructure node can reconstruct the inserted path by simply querying the other nodes that the path traverses. For example, suppose end-host e creates a confluence by inserting entries $(id_1, id_2), (id_2, id_3), (id_1, id_4), (id_4, id'_3)$, where IPADDR$(id_3)$ = IPADDR(id'_3). Now, every packet sent to id_1 would create two copies on IPADDR(id_3). Node IPADDR(id_1) can query nodes IPADDR(id_2) and IPADDR(id_4) for entries starting with id_2 and id_4 respectively. Using the replies, IPADDR(id_1) can detect a confluence.

Non-repudiation. ROSE signs every paths it returns to an end-host, and the forwarding infrastructure inserts only paths signed by ROSE. This allows the infrastructure to impose fine-grained policies specific to ROSE providers.

5.3 Verifying Paths Returned by ROSE

An end-host can verify the quality of the path returned by ROSE by measuring the end-to-end performance it sees. If the performance is unsatisfactory (*e.g.,* the latency along a low latency path returned by ROSE is consistently higher than the latency of the direct IP path), the end-host can simply switch to using another ROSE. This is similar to the way users can switch from one search engine to another when they are not satisfied with the quality of the search results. This design is consistent with the high-level goals pointed out in [15], namely, modularization of design at tussle boundaries and design for choice.

6 Implementation Overview

We have implemented the system: forwarding infrastructure, ROSE, the client API given in Table 1 and a proxy for using legacy applications with our system in C. In our implementation, LABEL(id) is 128-bits long for cryptographic strength.

6.1 ROSE Implementation

We have implemented a distributed ROSE that runs on multiple nodes, each node monitoring a portion of the infrastructure that is specified. We also instrumented the infrastructure code to log all probe packets sent by ROSE, and record local time-stamp at each hop the packet traverses in the probe packet itself. We used this information to compute the *actual* performance characteristics of the links and use them to evaluate the accuracy of our estimation algorithms.

For efficiency, all the one-way hash functions (*e.g.,* for cryptographic constraints) we use are based on the advanced encryption standard (AES) (aka. Rijndael block cipher) [16] in the Matyas, Meyer, and Oseas construction [28]. The output of the AES cipher is again XORed with the input to make the function irreversible.

6.2 Client API

Table 1 shows the core API of our system. An end-host can query ROSE for a path between two nodes in the forwarding infrastructure using the get_path API call. The first two arguments of get_path specify the starting and the ending points of the path in the infrastructure. An end-point of a path is identified by either the IP address of the node (IP_n) or an identifier id_n where IPADDR$(id_n) = IP_n$.[6] When an end-point is specified by its IP address IP_n, the ID in the path id_n that corresponds to IP_n is chosen such that it obeys the constraints mentioned in Section 4.1.2. If both end-points are IP addresses, then one of the IDs is chosen at random. We have an API call to set-up paths to/from IDs (and not just IP addresses) because of the need to establish a path to a particular identifier that represents the other end-point of the communication. We note that the API does *not* allow an end-host to identify both end-points using IDs (refer Section 4.1.2).

The third argument of get_path, rid, is either NULL or an ID such that IPADDR(rid) represents the IP address of the caller. If rid is NULL then the path ends in the infrastructure. Otherwise the path ends at the caller. Note that only the caller can insert a path that ends to itself. The last argument of get_path, opt, specifies the metric for which the caller wants the path optimized for. The existing metrics are delay, loss rate, and available bandwidth.

Once it gets a path p from ROSE, an end-host can insert p using the *insert_path* call. To maintain the path, the end-host needs to periodically refresh the path. This process is very similar to maintaining a path in RSVP. The refresh period we use in our implementation is 30 sec. To send a packet along a path p, an end-host uses the $send$ API call. The first parameter of $send$, id, represents the first ID of the path p.

All the API functions have been implemented in C for sup-

[6]Recall that the first part of an ID represents the IP address of the node on which the ID is stored.

Function Call	Description
$s.get\_path(n_1, n_2, rid, opt)$ or $s.get\_path(id_1, n_2, rid, opt)$ or $s.get\_path(n_1, id_2, rid, opt)$	Query ROSE server s for a path from $n_1/ipaddr(id_1)$ to $n_2/ipaddr(id_2)$. If $rid \neq NULL$, the path continues to rid, where IPADDR(rid) is caller's IP address. opt specifi es the metric for which the caller requests optimization.
$p = e.return\_path()$	Return path to end-host e
$n.insert\_path(p)$	Insert path p starting at node n, where n is the fi rst node on p
$n.send(id, IP\_packet)$	Send an IP packet along path with entry point id. $n = $ IPADDR(id).

Table 1: Generic API.

porting asynchronous model of programming. In particular, applications can associate callback functions with the different paths that they deal with. A callback is then invoked when a packet arrives along the corresponding path. Inserting and maintaining the paths is automatically handled by the client stub, thus relieving the applications of this work.

To use unmodified legacy Internet applications on our system, we developed a user-level proxy that runs with superuser privileges. We have used this proxy for performing experiments reported in Section 7.1.2. Providing details about how the proxy functions is out of scope of this text.

7 Evaluation

We micro-benchmarked the software implementation of the forwarding infrastructure on a Pentium 2.2 GHz machine running Linux 2.4.20. Each routing entry takes 120 bytes of memory. To evaluate the cost of inserting an entry into the table, we first populated the routing table with $100,000$ entries. Then the cost of insertion, computed as the average of next $10,000$ insertions, was $2.5\mu s$. Not including the cost of routing table lookups, the cost of packet forwarding averaged over $10,000$ packets of size 1 KB was $18.5\mu s$. This corresponds to a forwarding rate of up to $47,600$ packets/second.

In the rest of this section, we evaluate our architecture on three important fronts. First, we demonstrate two examples of how applications can benefit from our architecture – metric-sensitive multicast and fine-grained failover. Next, using real-world experiments, we show that the ROSE architecture shows promise in being able to return good paths by maintaining only $O(N)$ state per server. Finally, we present experiments that evaluate the accuracy of the indirect measurement techniques that ROSE employs to verify the infrastructure reports.

All our experiments were performed with up to 100 PlanetLab nodes distributed among 10 ($= \sqrt{100}$) ROSE servers using METIS [26]. For experiments in Section 7.1 and 7.2, we chose only one node per site mostly because many nodes in a location would skew results in our favor. Also, for validation purposes, we measured RTT, loss rate and bandwidth for *all* pairs of nodes in our set, which limited the number of nodes to keep the overall traffic reasonable.

Though the number of nodes we used here is two orders of magnitude smaller than the infrastructure size we envision, we believe that these experiments nonetheless give insights into how the ROSE architecture would perform by maintaining only $O(N)$ state. As pointed out in [22], the lack of tools and techniques to assign metrics (such as latency and bandwidth) to simulated graphs in a manner that would reflect Internet routing was an impediment in performing meaningful simulations. For most cases, due to lack of space, we present results for RTT and loss rate, and omit the bandwidth results.

7.1 Application-level Benefits

Two of the main applications that have been targeted by overlay solutions are multicast [14, 11, 18, 19, 25, 32, 36] and resilience [6]. We next describe how these applications are supported by our architecture.

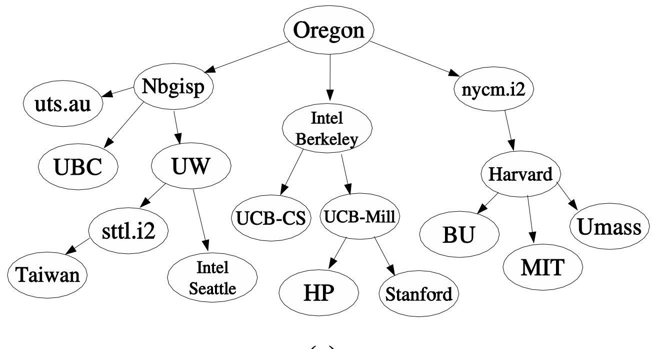
7.1.1 Single source Metric-sensitive Multicast

Multicast is a perfect example of an application that can benefit from separating the control and data plane: data path is highly efficient as packet are replicated in the infrastructure (not at end-nodes as in end-host multicast solution), and the control plane is highly flexible—introducing a new performance metric requires neither the deployment of a new infrastructure, nor the change of the existing one. More precisely, in this section we show that our architecture supports single source multicast trees which are optimized for two different metrics: delay and loss rate.

Constructing single-source multicast trees with bounded degree based on different metrics is straightforward. As illustrated in Section 4.1.3, the multicast tree is simply the union of unicast paths to each of the receivers.

Figure 7 shows a small portion of multicast trees (rooted at `planetlab1.cs.uoregon.edu`) for two different metrics, RTT and loss rate, involving 100 PlanetLab nodes. As expected, the paths chosen by the two metrics are quite different. For example, during the experiment, the link from Oregon to `nbgisp.com` helped in picking low RTT paths, but consistently had a loss rate of about 3% (which is much higher compared to the other paths we see on PlanetLab).

To quantify the benefit of supporting metric-sensitive multicast, we compute the *loss rate* along the *delay*-sensitive tree. This quantifies how much a loss-sensitive application would lose if the underlying infrastructure were to build trees using only the delay metric. Note that this is what would happen



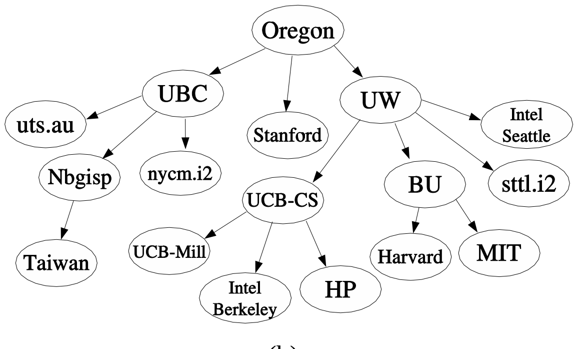


Figure 7: Portions of multicast trees constructed using two metrics (a) delay, (b) loss rate.

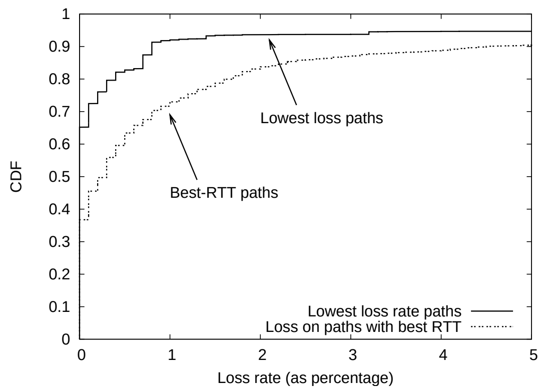


Figure 8: Loss rates on paths constructed using delay as metric

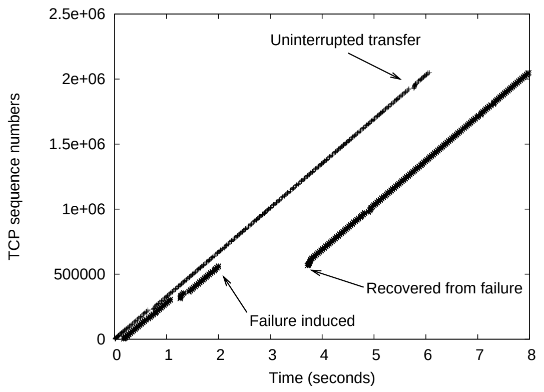


Figure 9: Progress of a TCP connection under link failure.

in today's IP multicast which optimizes the tree construction using the shortest-path metric, a metric over which applications have no control. Figure 8 shows the cumulative distribution of the loss rate both in (i) the loss-sensitive, and (ii) the delay-sensitive multicast trees. Overall, the loss rate in the latter case is appreciably higher than the former. For example, in case (i), 90% of the paths have loss rate less than 1%, whereas in case (ii) the number is only 70%. Furthermore, in more than 40% of the paths the loss observed in case (ii) is more than twice that in case (i).

7.1.2 *Fine-grained Application-level Failover*

As mentioned in section 3, when a virtual link between two infrastructure nodes fails, the infrastructure resorts to IP routing. While in this scenario, our infrastructure is no better than the Internet, next we show that by taking advantage of the ability to control routing, end-hosts can perform fail-over in the order of seconds, and at the granularity of a single flow. To achieve this, end-hosts insert for each path a backup path in the infrastructure. ROSE can provide a service that gives multiple disjoint paths. When the primary path fails or its performance degrades, the end-hosts can switch immediately to the back-up path. The problem of detecting the failure is left to end-hosts, as end applications are in the best position to decide when to switch.

We have implemented the maintenance of multiple paths in the user-level proxy (refer Section 7). For TCP connections, we switch to the backup path when multiple timeouts are experienced. In our experimental setup, we instantiate the infrastructure nodes on PlanetLab, and a sender and receiver on Internet2 hosts that are well-connected to PlanetLab.

Figure 9 shows the progress of a 2 MB file transfer using legacy file transfer client. The proxy establishes one backup path for each primary path. At around $t = 2$ seconds, we emulated a failure by removing one of the infrastructure nodes on the first path. Within 1.7 seconds, the proxy is able to detect and recover from the failure. This compares favorably with RON which provides similar recovery times [6]. Using multiple paths, an application can achieve zero-time failover by sending packets on both the paths using erasure coding techniques. Finally, note that when the primary and backup path start/end at different infrastructure nodes, our mechanism can be viewed as a generalization of multihoming [2, 3].

7.2 ROSE **Performance**

Prior work (Detour [37], RON [6]) has shown that Internet paths are not optimal. In this section, we study the extent to which our solution can take advantage of this sub-optimality. Unlike the studies in [6, 37] which monitor the virtual links between any two nodes, due to scalability and policy constraints reasons, we monitor only a subset of the virtual links. Still, we are able to show that in many cases ROSE finds better paths than the underlying Internet paths.

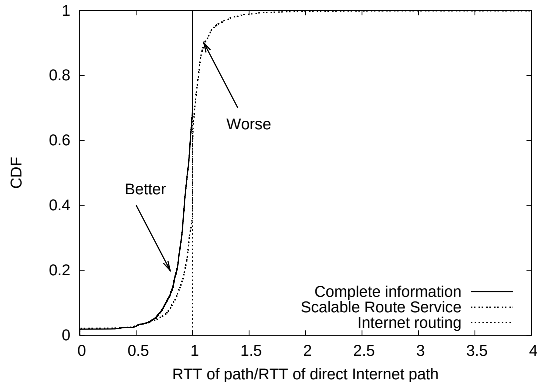


Figure 10: CDF of ratio of RTT using distributed ROSE mechanisms to RTT on Internet path.

Figure 10 shows the CDF (over all pairs of nodes) of ratio of RTT between pairs of nodes using the algorithm described in Section 4.2 to the Internet RTT. For comparison, we also show the CDF of the ratio of best possible RTT (using the complete $O(N^2)$ information) to the Internet RTT. We can infer that we do better than the Internet in about 40% of the cases, and within a factor of 1.01 of the Internet in 65% of the cases. We believe that one of the main reasons why we do worse in some cases is that our evaluation testbed has a sparsely distributed set of nodes in Europe and Asia. This is a departure from the assumption of the algorithm that the infrastructure nodes are well-distributed in the "middle" of the network.

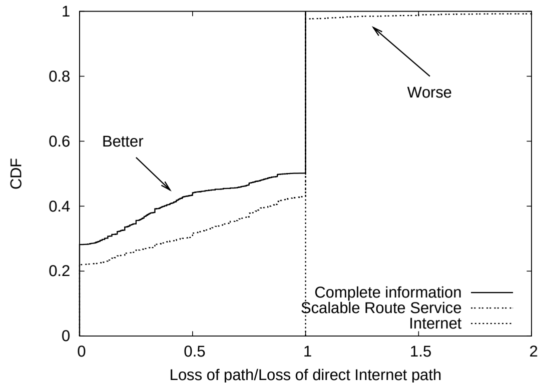


Figure 11: CDF of ratio of loss using distributed ROSE mechanisms to loss rate on Internet path.

Figure 11 demonstrates a similar experiment performed with the loss rate metric. In most cases when there are better paths than the Internet, our algorithm finds them. But (not surprisingly), unlike the previous case (RTT), the paths are not as efficient as the best possible path. However, the algorithm does well in identifying the lossy links (this cannot be inferred from the figure). In particular, of all the paths where the loss rate is greater than 5%, in over 40% of the cases, it was able to find a path that reduced the loss rate by at least half, and it found a better path in 96% of the cases.

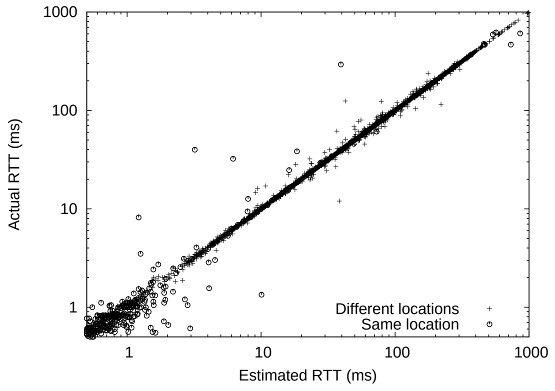


Figure 12: The scatter plot of the actual and the measured RTT between two arbitrary IDs.

7.3 ROSE Measurement Accuracy

In this section, we evaluate the estimation algorithms described in Section 5.1. For each performance metric (*i.e.,* RTT, loss rate and available bandwidth) we compare the *actual* values to the values estimated by our algorithms. To compute the actual values we instrument infrastructure nodes to log packets they receive (for delay and loss rate) and perform pairwise measurements between them (for bandwidth). Our experiments show that our estimation algorithms are accurate in the case of RTT and loss rate, and reasonable in the case of bandwidth estimation.

To understand how RTT and loss rate estimation algorithms performs in different cases, we divided the pairs of infrastructure nodes into two bins: pairs of nodes in the same location (*e.g.,* both in MIT), and pairs of nodes in different locations (*e.g.,* one in Stanford, other in Rice).

For the former, we considered 44 pairs of nodes, and for the latter we considered about 2450 pairs, *i.e.,* two data sets, each set comprising all pairs of nodes, one in each of 50 PlanetLab locations.

7.3.1 Round Trip Time (RTT)

Figure 12 shows the scatter plot of the estimated RTT versus the actual RTT for the samples over a 100 second interval. Every virtual link is sampled every 10 seconds[7], thus the plot contains 10 samples per virtual link. The results show that our RTT estimation algorithm (described in Section 5.1) is very accurate. In Figure 12, less than 2.7% of samples have an error $>10\%$. If we take the median among 10 consecutive

[7]We restricted our sampling rate as we monitored all virtual links for the sake of validating our techniques.

samples (not shown in figure), only 0.7% of the samples have a relative error $>10\%$, most of which are due to nodes in the same location.

7.3.2 Loss Rate

Figure 13 shows the scatter plots of the actual versus the measured loss rates for the aforementioned data sets. To estimate the loss rate between two nodes we use the scheme described in Section 5.1. Each data point is the result of sending 1000 probes. In most cases, the measured loss rates were quite small; only in about 8% of cases we measured a loss rate larger than 2%. Figures 13(a) and (b) show the loss rates for all links in the forward, and reverse directions. The points below the line $x = y$ are mainly due to false positives, *i.e.*, ROSE incorrectly decided that there was a loss on the link. The points above $x = y$ are due to the fact that ROSE ignores the probes for which it does not receive any response.

From the technique used, we see that inaccuracies could occur when, (i) the loss between ROSE server and the measured link ($A \rightarrow E, B \rightarrow E$ links in Figure 5(b)) is considerably larger than the loss rate on the measured virtual links ($A \rightarrow B$ and $B \rightarrow A$ in Figure 5(b)) and (ii) losses on links are not independent. To verify this, we identify the nodes that are responsible for the highest loss rates, and eliminate them from the measurements.[8] As expected, the estimation accuracy improves considerably, especially on the reverse path. The loss rate estimates were within 90% of the actual loss rate in 70% of the cases in the forward direction and in 74% of the cases in the reverse direction. Finally, if we consider only the virtual links with loss rate more than 0.2%, then the estimates were within 90% of the actual for 81% of cases in forward direction and for 83% of the cases in the reverse direction. In summary, the algorithm performs very well in identifying moderately/highly lossy links, and does reasonably well for links with very low loss rate.

We make two further observations. First, in the forward direction (Figure 13(a)), we are more likely to overestimate than underestimate the loss rates (all points below $x = y$ represent overestimations). Second, one can easily obtain better results for the reverse path by simply reversing the measurement setting for that links. However, this results in doubling the overhead since now we have to send probes in both directions of the measured link.

7.3.3 Available Bandwidth

To evaluate the technique for determining the available bandwidth (avail-bw), we chose one node at each PlanetLab site.[9]

[8]During the experiments reported here we identified five such nodes: two at `cs.unibo.it`, one at `Intel Pittsburgh`, one at `CMU`, and one at `diku.dk`.

[9]Only one node per site was used to economize on the bandwidth that we use for our experiments.

Though we implemented the technique described in Section 5.1.3 on both Pathload as well as a vanilla TCP Vegas implementation, we use the latter because it was difficult to obtain consistent results for the former on PlanetLab for validation. A detailed investigation of how our technique works on a variety of tools is deferred to future work.

For validation, we compute bandwidth between any two nodes by transferring 1 MB of data directly between the two nodes, and recording the bandwidth over the transmission of the last 500 KB.[10] To understand the performance of our algorithm clearly, we divided the pairs of nodes into the following three vastly different cases, primarily motivated by the fact that longer RTTs might affect the estimation process: (i) ROSE server is close to one (or both) of the two nodes (labeled Close in Figure 14) (ii) ROSE server is far away from the two nodes which are themselves far apart (labeled Far-Far), and (iii) ROSE server is far away from the two nodes which are close to each other (labeled Far-Close). Studying results based on this classification is useful as it gives some insights into how to assign nodes to ROSE servers.

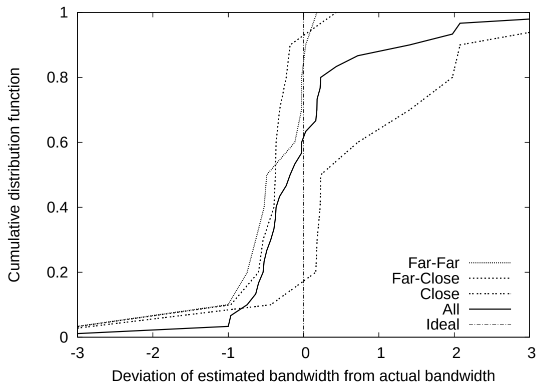
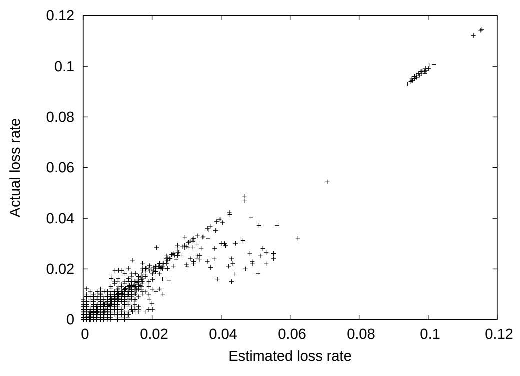


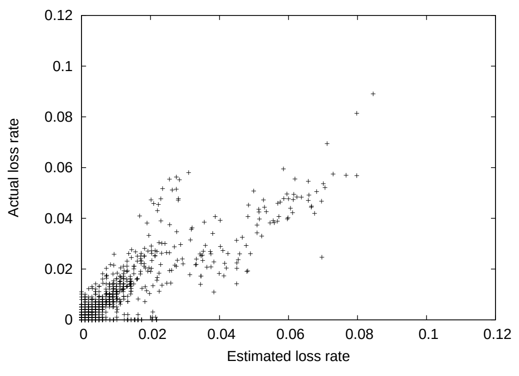
Figure 14: CDF of deviation in estimated bandwidth from measured bandwidth.

Figure 14 shows the CDF of the relative deviation of the estimated bandwidth from the stable direct bandwidth (ratio of difference between estimated and actual to the actual bandwidth). We make the following observations; a detailed analysis about the timing effects of the measurement is beyond the scope of this paper: (a) Overall, in 60% of the cases, the relative error in estimated bandwidth is less than 0.5. For comparison, in 43% of the pairs, the relative deviation in two values measured using direct transfers was itself 0.5 or more; (b) The technique did not work well when we estimated high-bandwidth links that were very far from the ROSE server – a node in USA could not estimate the high bandwidth between `uu.se` and `diku.dk` well. This was because the two Europe nodes were separated by only 11ms, whereas the ROSE server was over 200ms away from each of them. In a deployed system, we can alleviate the problem by

[10]This is to allow the flow to get into congestion avoidance phase, so that we see the stable bandwidth.



(a)



(b)

Figure 13: Scatter plots of the actual versus the measured loss rates for: (a) forward path, (b) reverse path

having a distributed ROSE service and assigning infrastructure nodes to topologically close ROSE servers and (c) In the case when one of the nodes is close to the ROSE node, the available bandwidth technique over-estimates in some cases. We believe that this is because the overlay and IP paths are the same, and a delay-based technique sees fewer losses. Finally, we recognize that this particular technique would work only for drop-tail queues. Exploring these measurement techniques further by incorporating topological and delay information is an area of future work.

8 Discussion and Future work

We list topics that are not fully addressed in the paper. A thorough study of these issues is foremost in our agenda.

Optimality of paths. ROSE chooses sub-optimal paths due to two reasons. First, it picks only a subset of the links between two domains. Second, it makes use of at most two domains while computing paths between two infrastructure nodes. We believe that this approach provides a tradeoff between complete flexibility of path selection and scalability. Furthermore, if the infrastructure is widely deployed, the multiplicity of paths might alleviate this sub-optimality.

Stability of the network. The routing decisions of end-hosts are controlled by ROSE, where the aspect of stability of routes can be handled. However, when end-hosts receive service from multiple non-cooperating ROSE, stability might be a concern. In such cases, ROSE can use randomization techniques to return a path which is almost as good, but not the same as the best path. Furthermore, techniques such as hysteresis can be employed while performing load-sensitive path switching at the end-hosts.

Payment model. One possibility is for ROSE providers and ISPs to charge the end-hosts based on the number of queries made, and the number of forwarding entries used up, respectively. The advantage is that this prevents malicious users from inserting long paths in the infrastructure. Further exploration is beyond the scope of this paper.

9 Conclusion

In this paper, we propose a network architecture for giving end-hosts control over routing in the network. The basic tenet of our proposal is to provide specialized routing as a service, not embed it into the infrastructure. We proposed an architecture to realize this idea, addressed the issue of trust, and presented a solution for the routing service. We believe that this approach has the benefit that routing can evolve as needs change without any change to the infrastructure.

Our prototype, while deployed and tested on PlanetLab, is not in widespread use yet. Such use will undoubtedly prompt many design refinements. One area of improvement is the quality of paths returned by ROSE. For scalability, ROSE uses a simple performance-agnostic mechanism to partition the infrastructure nodes. This partition can be made sensitive to the performance metric that ROSE optimizes. Another improvement is providing shared performance discovery between the end-hosts and the first infrastructure node. Only a large-scale deployment and widespread use will ultimately give a complete answer to questions regarding the stability and the scalability of our architecture.

References

[1] High-Performance BLAS. `http://www.cs.utexas.edu/users/flame/goto/`.

[2] RouteScience. `http://www.routescience.com`.

[3] Sockeye Networks. `http://www.sockeye.com`.

[4] D. Adkins, K. Lakshminarayanan, A. Perrig, and I. Stoica. Towards a More Functional and Secure Network Infrastructure. Technical Report UCB/CSD-03-1242, UCB, 2003.

[5] Akamai Technologies. `http://www.akamai.com`.

[6] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. In *Proc. SOSP*, 2001.

[7] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service. RFC 2475, 1998.

[8] C. F. Borenstein, G. L. Miller, S. B. Rao, and T. K. Canfi eld. Optimal Route Selection in a Content Delivery Network. *US Patent*, Sept. 2002. #WO02071242.

[9] K. L. Calvert, J. Griffi oen, and S. Wen. Lightweight Network Support for Scalable End-to-End Services. In *Proceedings of ACM SIGCOMM*, Pittsburgh, PA, 2002.

[10] I. Castineyra, N. Chiappa, and M. Steenstrup. The Nimrod Routing Architecture. RFC 1992, 1996.

[11] Y. Chawathe, S. McCanne, and E. Brewer. An Architecture for Internet Content Distribution as an Infrastructure Service, 2000.

[12] Y. Chen, R. H. Katz, and J. D. Kubiatowicz. SCAN: A Dynamic Scalable and Effi cient Content Distribution Network. In *Proc. Pervasive Computing*, Aug. 2002.

[13] D. R. Cheriton and M. Gritter. TRIAD: A New Next Generation Internet Architecture, Mar. 2000. `http://www-dsg.stanford.edu/triad/triad.ps.gz`.

[14] Y.-H. Chu, S. G. Rao, and H. Zhang. A Case for End System Multicast. In *ACM SIGMETRICS*, 2000.

[15] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in Cyberspace: Defi ning Tomorrow's Internet. In *Proc. of ACM SIGCOMM*, 2002.

[16] J. Daemen and V. Rijmen. AES proposal: Rijndael, Mar. 1999.

[17] A. Doria, F. Hellstrand, K. Sundell, and T. Worster. General Switch Management Protocol (GSMP) V3. RFC 3292, 2002.

[18] Fastforward Networks. `http://www.ffnet.com`.

[19] P. Francis. Yoid: Extending the Internet Multicast Architecture, 2000.

[20] P. Francis, S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. IDMaps: A Global Internet Host Distance Estimation Service. *IEEE/ACM ToN*, October 2001.

[21] S. Gribble, M. Welsh, R. von Behren, E. Brewer, D. Culler, N. Borisov, S. Czerwinski, R. Gummadi, J. Hill, A. Josheph, R. Katz, Z. Mao, S. Ross, and B. Zhao. The Ninja Architecture for Robust Internet-Scale Systems and Services, 2000.

[22] K. P. Gummadi, R. Gummadi, S. D. Gribble, S. Ratnasamy, S. Shenker, and I. Stoica. The Impact of DHT Routing Geometry on Resilience and Proximity. In *SIGCOMM*, 2003.

[23] K. P. Gummadi, S. Saroiu, and S. D. Gribble. King: Estimating Latency between Arbitrary Internet End Hosts. In *Proceedings of IMW*, Marseille, France, November 2002.

[24] M. Jain and C. Dovrolis. End-to-End Available Bandwidth: Measurement Methodology, Dynamics, and Relation with TCP Throughput. In *Proc. ACM SIGCOMM*, 2002.

[25] J. Jannotti, D. K. Gifford, K. L. Johnson, M. F. Kaashoek, and J. J. W. O'Toole. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of OSDI*, 2000.

[26] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. In *SIAM Journal on Scientifi c Comp.*, 1995.

[27] H. Khosravi and T. Anderson. Requirements for Separation of IP Control and Forwarding. RFC 3654, Nov. 2003.

[28] S. Matyas, C. Meyer, and J. Oseas. Generating Strong Oneway Functions with Cryptographic Algorithm. *IBM Technical Disclosure Bulletin*, 27:5658–5659, 1985.

[29] A. Nakao, L. Peterson, and A. Bavier. A Routing Underlay for Overlay Networks. In *Proc of ACM SIGCOMM*, 2003.

[30] T. S. E. Ng and H. Zhang. Global Network Positioning: A New Approach to Network Distance Prediction. *CCR*, 2001.

[31] J. Padhye, V. Firoiu, D. Towsley, and J. Kurose. Modeling TCP Throughput: A Simple Model and its Empirical Validation. In *SIGCOMM*, 1998.

[32] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. ALMI: An Application level Multicast Infrastructure, 2001.

[33] Planet Lab. `http://www.planet-lab.org`.

[34] B. Raman and R. H. Katz. An Architecture for Highly Available Wide-Area Service Composition. *Computer Communications Journal*, May 2003.

[35] E. Rosen, A. Viswanathan, and R. Callon. Multiprotocol Label Switching Architecture. RFC 3031, Jan. 2001.

[36] A. Rowstron, A.-M. Kermarrec, M. Castro, and P. Druschel. SCRIBE: The Design of a Large-Scale Event Notifi cation Infrastructure. In *NGC*, pages 30–43, 2001.

[37] S. Savage, T. Anderson, A. Aggarwal, D. Becker, N. Cardwell, A. Collins, E. Hoffman, J. Snell, A. Vahdat, G. Voelker, and J. Zahorjan. Detour: A Case for Informed Internet Routing and Transport. Technical Report TR-98-10-05, 1998.

[38] S. Seshan, M. Stemm, and R. H. Katz. SPAND: Shared Passive Network Performance Discovery. In *Proc USITS*, 1997.

[39] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM*, 2002.

[40] V. Strassen. Gaussian elimination is not optimal. *Numerische Mathematik*, 1969.

[41] J. Strauss, D. Katabi, and M. F. Kaashoek. A Measurement Study of Available Bandwidth Estimation Tools. In *Proc IMC*, 2003.

[42] D. Wetherall. Active Network Vision and Reality: Lessons from a Capsule-based System. In *Proc. of SOSP*, 1999.

[43] X. Yang. NIRA: A New Internet Routing Architecture. In *Proc FDNA-03*, 2003.

[44] D. Zhu, M. Gritter, and D. Cheriton. Feedback-based Routing. In *Proc Hotnets-I*, 2002.