Intradomain Overlays: Architecture and Applications

Christopher Kommareddy, Tuna Güven, Bobby Bhattacharjee, Richard La, Mark Shayman
UMIACS, University of Maryland, College Park
{*kcr@cs, tguven@eng, bobby@cs, hyongla@eng, shayman@eng*}*.umd.edu*

***Abstract*— We introduce an architecture for "Intradomain Overlays", where a subset of routers within a domain is augmented with a dedicated host. These strategically placed hosts form an overlay network, and we describe a number of applications for such overlays. These applications include efficient network monitoring, policy- and load-based packet re-routing, and network resource accounting.**

In this paper, we elaborate on the network monitoring application and describe a distributed protocol for monitoring routers within an AS which has been augmented with a few overlay nodes. The routers and other infrastructure are unaware of the overlay nodes, and the monitoring of individual elements is conducted using plain SNMP. We describe techniques for efficiently synthesizing and transporting the monitored SNMP data, and present results using trace data collected from an AS with 400+ routers. Our results show that the overlay-based monitoring reduces overheads by 2–4 orders of magnitude, and thus enables much finer grained monitoring and traffic engineering than is otherwise possible.

***Index Terms*— System design, Simulations, Network measurements, Monitoring, Traffic Engineering, Overlay Networks, Intradomain, Distributed.**

I. Introduction

We introduce "intradomain overlays", where a subset of routers within an AS are augmented with dedicated end-hosts. These end-hosts form an overlay network, and allow network administrators to run different distributed "overlay applications" for network management and administration. In this paper, we outline a complete architecture for deploying such overlays, and describe a number of applications of such overlays.

A primary goal of this work is to enable true traffic engineering and robust network monitoring and administration. All of our proposed protocols are incrementally deployable, and do not require any changes to deployed IP infrastructure.

We should note that a router augmented with a dedicated end-host could be considered a viable platform for active networking [1]. However, our goals here are much more constrained: we do not allow user-programmability, and we envision extremely limited and bounded computations (e.g. encapsulation and decapsulation only) on the data path. These constraints vastly simplify our resource accounting and security problems at the cost of much less flexibility: this is a conscious choice for our platform which is designed entirely for single domain administration and traffic engineering.

Why Intradomain?

A distinguishing feature of our work is the constraint that our overlays are deployed withing a *single* domain. Clearly, if there is gain in deploying overlays within one domain, then a natural question is to consider gain that could be gotten by a larger inter-domain deployment. However, we constrain our focus to single domain deployments for the following reason:

Single Administrative Entity: We assume that the AS is administered by a single entity, and that the administrators of each router trust a single principal. While this may not be the case for all domains, we believe it holds for a large majority of domains. Further, since the overlay is deployed within a single domain, we can assume that the domain topology is available at a central node, and that all changes to the underlying IP network are managed by (or reported to) a single authority. Further, the overlay software can be updated at any time since these changes are not visible outside the domain, and all the intradomain nodes can be updated at once. These assumptions vastly simplify the overlay deployment and maintenance protocols since there is a single trusted centralized entity that knows of all topology changes and can mandate policy that is global for all deployed overlay nodes.

Thus, our focus on intradomain overlays is due to pragmatic issues. We believe that deploying overlays within a single domain provides sufficient benefit without necessitating yet another round of standardization that would be required if these protocols were deployed across domains. As an added bonus, in the single domain case, we can use extremely simple protocols which are more likely to be implemented correctly and be robust. Note that intradomain overlays do not preclude inter-domain deployments, though we do not consider such an extension in our work.

A. Applications

Intradomain overlays are beneficial only if there are useful applications that can be deployed over the overlay nodes. Specifically, we are interested in applications that both *require* the overlay nodes for efficient implementation, and are *useful* in practice. We believe there are a number of classes of such applications, and we discuss two in detail next.

Network Monitoring, Accounting and Management: The in-network overlay nodes can form an extremely efficient coordinated network monitoring platform. The overlay nodes can use plain SNMP to monitor nearby routers, but can then synthesize and compress relevant data and answer sophisticated queries about the state of the network. We present a scheme for such monitoring in this paper, and show that the overlay nodes can reduce overhead by several orders of magnitude. Such efficient monitoring can be used for pre-emptive network management, and can be used to account for resource usage within (and across) a domain. Monitoring can also be used to implement many reactive routing policies, described next.

Flexible Intradomain Routing: The in-network overlay nodes form a complete platform for re-routing. Packets can be diverted off the underlying shortest paths onto specific policy-based alternate paths. Note that the underlying network and paths are still used for forwarding, but packets are encapsulated and explicitly routed to specific overlay nodes. Such rerouting essentially allows MPLS [2] like circuits between overlay nodes without requiring investment in new infrastructure, or standardizing on new protocols and headers.

Once a rerouting mechanism is instantiated in the network, packets can be rerouted using various policies, e.g. real-time Voice-over-IP packets could use a specific path with small queues which is insulated from best-effort peer-to-peer downloads. In this case, either the VoIP or the p2p traffic (or both) would be classified and routed using the overlay.

Rerouting also allows better load-balancing among the links within a domain, and we have developed an optimal load-based routing algorithm [3]. This scheme, like other load-based rerouting algorithms [4], [5], require fine-grained load information from different links in the network. The overhead of gathering such information using traditional means would render moot any benefit from the load balancing; however, overlay-based monitoring makes such algorithms immediately viable.

Contributions

The contributions of this paper are as follows:

- We introduce intradomain overlays and present algorithms for efficient creation and maintenance of secure single domain overlays.
- We describe how such overlays can be used to efficiently monitor IP routers, and present detailed simulations that compare our approach against plain SNMP-based monitoring. Our results show that overlay monitoring is more efficient by several orders of magnitude.

Roadmap*:* The rest of the paper is structured as follows: In the next section, we describe our overlay network architecture. In Section III, we present the overlay node software architecture and describe the capabilities required by overlay node hardware. In Section IV, we discuss network monitoring using SNMP and show how to efficiently monitor an AS using an overlay. In Section V, we present simulation results that quantify the benefits of overlay-based monitoring. In Section VI, we briefly discuss traffic engineering and present an algorithm that achieves optimal routing using overlays. In Section VII, we give a detailed comparison of our work with other related work and conclude in Section VIII.

II. Overlay Network Architecture

The overlay network operates on top of the network layer of the AS and consists of the set of overlay nodes deployed within the AS. Each overlay node is attached to one of the network interfaces of an AS router. The overlay nodes coordinate with each other to aid in the correct and efficient functioning of the AS network. Multiple distributed *overlay applications* can run over these overlay nodes, and these applications can be used to monitor the status of the network, to re-route traffic according to network policy, to enforce specific security policies, etc. It is important to note that in this paper, an overlay application refers to these *in-network* processes and is not related to end-host based overlay applications (or p2p applications) that may be deployed by end users of the AS. The overlay nodes form a virtual topology over the AS network. We will refer to this physical (AS) network as the base network.

Consider the base network consisting of nine routers (r_1–r_9) shown in Fig. 1. In this AS, there are four overlay nodes, and overlay hops (e.g., N_1–N_7) may span multiple base hops ($\{r_1$–r_2, r_2–$r_7\}$). As we will describe in Section VI, such overlay hops can divert traffic from the default paths in the base network, and can be used for policy-based re-routing. In general, the overlay applications communicate using these overlay hops.

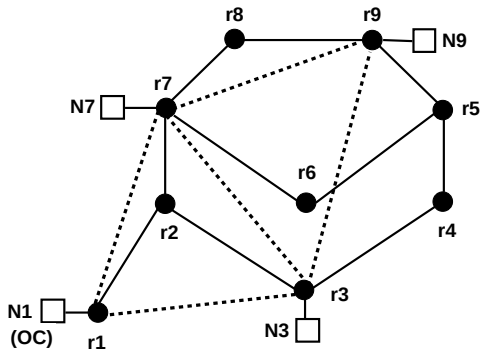

Fig. 1. An example overlay network

A. Overlay Management

Our framework supports incremental deployment (and re-deployment) of overlay nodes. The overlay network is also designed to withstand overlay- and underlying-node failures. Since our overlay design is targeted for a single AS, we employ a centralized node, called the overlay controller (OC) for overlay management and configuration. The AS topology is input to the OC, and the OC is also responsible for failure detection and recovery of other overlay nodes (described below).

The Overlay Tree*:* The overlay nodes within an AS form a logical tree rooted at the OC. This base communication infrastructure is used to maintain the integrity of the overlay network; it can also be used by some overlay applications to exchange data (e.g., the monitoring protocol described in Section IV uses it to convey node state). Once again, we appeal to the single AS trusted domain scenario, and use a centralized protocol to compute the overlay tree as follows: each overlay node is seeded with the address of the OC. Upon startup, the overlay nodes send a message to the OC, which causes the OC to recompute a minimum spanning tree and individually send tree updated neighbor information to each affected overlay node. Obviously, a distributed (and more sophisticated) tree building algorithm can be incorporated into this base structure, but we believe this simplistic scheme is sufficient for many, if not most ASes.

Once the overlay tree is constructed, the overlay nodes use a heartbeat-based mechanism to detect failed overlay nodes. Specifically, each overlay node periodically sends a *keepalive* message to its tree parent. If such a message is not received from a child for a predefined timeout, the parent tries to directly communicate with the child node. If this attempted communication fails, the child is declared failed, and a message to this effect is sent to the OC. (Note that the "child failed" message is unicast to the OC since the integrity of the overlay tree cannot be guaranteed during overlay node failures.) Upon receiving these messages, the OC reconstructs a new overlay tree and notifies affected nodes of changes in the overlay topology. This procedure reconstructs the tree only if the base network is not partitioned, and is not able to distinguish whether an overlay node has failed, or the first hop router for an overlay node has failed. In case the network does get partitioned, this can be detected by the OC (assuming there are overlay nodes that detect specific router failures), and overlay applications can continue to function on the component that contains the OC.

Obviously, the OC is a single point of failure, and has to be monitored. The obvious idea is to maintain multiple (replica) OCs, and all of our protocols can be extended to this case. However, in our implementation, and in the simulations we present in Section V, we assume that the OC does not fail.

B. Communication and Computation on the Overlay

By default, the overlay topology supports unicast, multicast, and concast [6] modes of communication. Two overlay nodes may choose to unicast messages using overlay links. The OC and other overlay nodes can send messages to the entire set of overlay nodes using the overlay tree. These multicast messages are often administrative updates which carry information about changes in the underlying topology, or updates to the overlay tree, etc.

The overlay topology also supports *concast*. Specifically, messages from multiple overlay nodes can be sent to a single overlay node, and aggregated using specific rules as they traverse upstream on the overlay tree. Such aggregation can be used to suppress duplicates, count messages, or in general, compute a constrained function on messages from different incident tree edges to create a new upstream message. This facility will be heavily used in the monitoring protocol we describe in Section IV. Further, we allow specific overlay applications to set up and maintain a small (configurable) amount of state at each overlay node and compose different distributed computations using this state much like the ephemeral state processing paradigm [7].

Router Assignments*:* A number of different overlay applications, e.g., monitoring, load-based rerouting, etc., require that each router in the underlying network be monitored by some overlay node. We use a static mapping of routers to overlay nodes: the map is computed such that each router is monitored by the overlay node that is closest to it. This router proximity map is computed centrally by the OC, and multicast to the overlay nodes using the overlay tree.

C. Security

The communication between the overlay nodes must be secure, otherwise the entire AS would be open to multiple catastrophic attacks. In our constrained domain, standard security primitives can be employed for maintaining source authenticity and data integrity. We assume that the overlay nodes themselves are trusted, and upon bootup, they are provided with a symmetric key that they share with the OC. (Note that it is trivial to change this scheme such that a public-key based scheme is used instead of symmetric keys; we use symmetric keys entirely for run-time efficiency. Even if public keys are used, we would not require an explicit PKI, since the OC's public keys can be manually distributed upon overlay node startup, and the OC can supply all nodes with the other nodes' public keys.) For most communication, the symmetric key is used to construct a keyed-MAC for integrity checks. Obviously, this key can also be used to encrypt any sensitive data, though none of our protocols require this level of security. The OC can be used to generate symmetric keys that two overlay nodes can use for secure unicast. Note that if public keys are used, and each node has its own public key known to all other nodes, then these keys can be used to exchange per-pair symmetric keys. In general, we believe any level of security for our managed overlays can be achieved using well-known techniques, and the real challenge is in proper implementation and deployment of the security protocols, and not in the design of the protocols themselves.

III. Overlay Node Architecture

In this section, we describe the internals of individual overlay nodes, and outline the default services provided by these nodes.

Fig. 2 shows a schematic overlay node software architecture. The functionality of each module is as follows:

Monitoring: The monitoring module is responsible for monitoring the routers assigned to the overlay node. Monitoring is implemented using SNMP; however, the monitoring module can also use active probes to measure losses, delays, and available capacity on overlay paths.

Logging and Reporting: The data gathered by overlay nodes needs to be processed and either logged or reported. While the reporting is usually done to the OC, it is possible to configure the nodes such that different information is reported to different entities. For example, data from an intrusion detection module might be reported to a different site than regular network monitoring data.

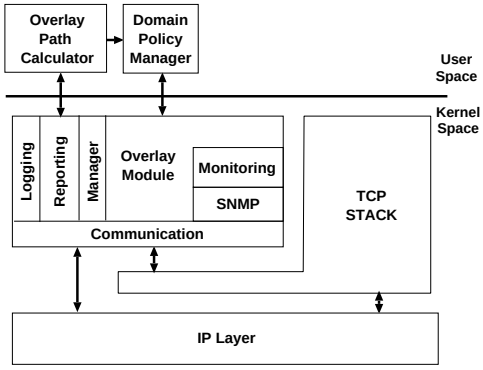

Fig. 2. Overlay Node Software Architecture Schematic

Communication: The communication interface provides facilities for secure group and unicast communication. Along with usual transport-layers, our implementation provides access to raw IP for fast packet forwarding and tunneling over the overlay links.

Policy Manager: The AS administrator sets the network policy as well as policies for overlay applications. The overlay network policies determine relative priorities of overlay applications, and also determine resource allocation at overlay nodes.

Admin. and App.-Specific Modules: Overlay nodes include a number of other modules that are used for regular node administration. These include modules for resource accounting, path computations, failure recovery, etc. Each overlay application can also start application-specific computations that can avail of services provided by the support modules described here.

A. Overlay Node Hardware

For most overlay applications, the overlay nodes do not require any special hardware. However, for line-speed re-routing, the overlay nodes may require special hardware for fast encapsulation, lookup, and decapsulation. If an AS overlay is used for re-routing, a few *special* overlay nodes must be deployed. All packets that can potentially be re-routed have to pass through an overlay node which would encapsulate a subset of these packets and direct these encapsulated packets off the default path onto overlay links. This overlay node can be situated as a pass through *before* the ingress router into the AS: this would ensure that all incoming traffic will in fact go through this overlay node (node A in Fig. 3) , but the node has to be fast enough to process, at line speeds, all packets that come into the network. It is possible to put the intercept overlay node at other parts of the network (e.g., node B in Fig. 3), but then packets destined for egress E_2 could not be rerouted in this network.

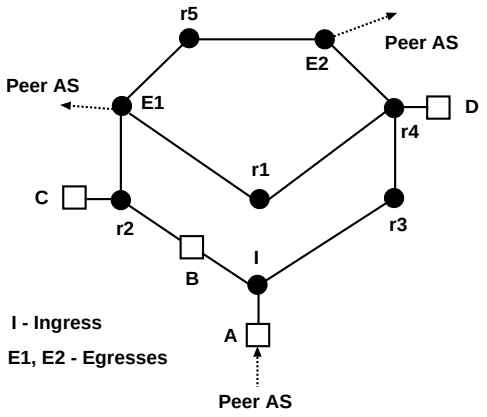

Fig. 3. Overlay Node Intercepting Base Network Traffic

In general, the encapsulation (and the decision to encapsulate) must be performed at line speeds. In our current implementation, we have implemented the encapsulation logic and mechanisms within the Linux kernel and this implementation running on a 266 Mhz Intel pentium II processor is sufficient to maintain 100 Mbps throughput. However, in a real AS, link speeds are likely to be much higher, and packets from multiple links may be incident upon the overlay node for rerouting. We believe network processors are an ideal platform for implementing such functionality, and are working on a network processor based platform for faster implementation of this logic.

IV. Overlay-based Monitoring

In this section, we describe in detail how an AS-level overlay can be used to efficiently monitor the links and routers in an AS. Note that we do not assume that the routers in the AS support any new protocols, thus all monitoring that involves querying a router must eventually use SNMP. However, we will describe how the overlay network can help in significantly reducing the overhead of SNMP, especially when a large domain is monitored with high frequency. In order to understand the source of benefits due to the overlay, it is instructive to consider the details of SNMP operation. We begin with a description of SNMP (next), and then describe how the overlay can reduce SNMP overheads (Section IV-C)

A. SNMP

The Simple Network Management Protocol (SNMP) [8], [9] is an IETF standard protocol for network management. SNMP classifies SNMP elements of the network as managers and agents. Agents can also act as proxies to other manageable nodes that do not implement SNMP protocols.

Agents implement the SNMP protocol, store and retrieve management information defined in Management Information Bases (MIBs) and in some cases, act as proxies. As a proxy, agents use non-SNMP or proprietary protocols to interact with non-SNMP manageable nodes. Agents also can asynchronously signal events to the manager; however, there are only limited, non-extensible sets of events for which this can be done.

An SNMP manager is also known as Network Management Station or NMS. It acts as a management console and can retrieve data from the agents using queries, set variables in the agents, and accept asynchronous signals from the agents. The manager retrieves the state of the network from the associated agents by polling agents. In the poll messages, it provides the object identifiers and in the replies, the agents fill the corresponding fields with the values of those object instances.

SNMP assumes connectionless communication and expects no reliability from the transport layer. Hence, SNMP data can be encapsulated and transmitted by TCP, UDP or any other transport protocols. UDP is the most commonly used protocol.

B. SNMP Overheads

SNMP is a generic protocol designed to work under all possible scenarios and for all manner of management information. SNMP messages include fields for authentication and privacy and other security features. Because of their general nature, SNMP messages incur high overhead. Specifically, SNMP messages have fields for protocol version, message identifier, message size, security flags, security model, and security parameters. While these fields are necessary for SNMP to function across versions and for protecting the integrity and privacy of the data, these fields are not necessary for the actual transmission of information.

The MIB objects are arranged in a tree and object identifiers trace this tree. For example, the identifier of the speed of the link object in the network interface MIB is 1.3.6.1.2.1.2.2.1.5 and the name of the object is *iso.org.dod.internet.mgmt.mib-2.interfaces.ifTable.ifEntry.ifSpeed*. These identifiers can be up to 128 bytes long. While such a hierarchy is required in the general case, it adds significant overhead to each SNMP message.

The SNMP agent tracks discontinuities in any of the interface's counters by recording the time when the counters started afresh. This time is stored in *ifCounterDiscontinuityTime* and if this value changes since the

last time any of the interface counters were polled, the current and previous polled values of the counters cannot be used. Similarly, when the *sysUpTime* object is reset (upon reinitialization), polled values cannot be used. While these objects are not necessary for monitoring the state of the AS, they are required to verify the integrity of the polled data. Thus, the NMS must periodically poll these objects to detect counter resets, etc., which all adds more overhead to SNMP.

C. Overlay-based Monitoring

Overlay nodes deployed in the network can be used to efficiently monitor the network. The overlay nodes can actively probe the overlay links to determine the available bandwidth, loss rate, etc. They can also monitor the overlay traffic to passively determine loss rates, link latencies and so on. In this section, we describe how to efficiently use the overlay nodes for SNMP based network monitoring.

The overlay monitor performs two main functions (shown in Fig. 4):

- It performs the role of an NMS node and polls the agents of the routers assigned to it.
- It processes the SNMP messages it receives and forwards only the relevant information on the overlay tree to the OC. (In this section, we assume that the NMS is co-located or connected to the overlay OC; if this is not the case, then a monitoring-specific tree rooted at the NMS has to be constructed. This tree can be constructed using exactly the same protocol as the overlay tree. In the rest of the section, we use OC and NMS interchangeably.)

In order for the overlay nodes to process the SNMP data, they now have to maintain object-specific state such as *ifCounterDiscontinuityTime* and *sysUpTime*. Further, in order to calculate time-varying metrics such as average bandwidth utilization, etc., the overlay nodes also have to maintain historic information such as last time of polling, last counter values, etc.

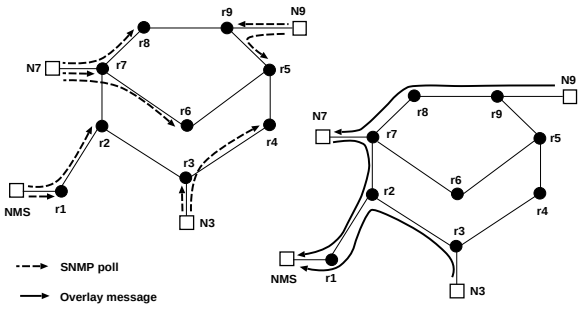

Fig. 4. Overlay Node functions: polling and forwarding

The overlay monitoring proceeds as follows: the OC builds the overlay tree and assigns routers to specific overlay nodes. The overlay nodes poll their respective routers to determine the indices of the interfaces at each of the routers in the SNMP tables and report this information to the OC. The OC needs this information to build the object identifiers of the object instances related to the interfaces.

The OC then *subscribes* to the objects it is interested in at each of the overlay nodes. This subscription is done by providing the object identifiers to the overlay nodes and the frequency with which it wants this information updated. The OC also provides the asynchronous events which the overlay nodes should report back. The asynchronous events include usual SNMP traps such as an interface failure, but can also include synthesis of other SNMP data, e.g., the number of IP packet errors exceeding a certain number. This subscription is refreshed periodically (once every 30 minutes in our testbed).

Efficient Monitoring*:* The overlay nodes poll their respective SNMP agents, process the response messages, and the messages received from their children since the last time they sent a message to their parent. They consolidate all these messages and forward an appropriate digest to their own tree parent. The message processing is a crucial step since the overlay nodes can process the polled data on behalf of the NMS and hence, reduce the processing overhead at the NMS. The processing also drastically reduces the bandwidth consumption of the monitoring, which directly allows data to be captured at much higher frequencies.

We further reduce bandwidth requirements by using a compressed representation when reporting values back on the overlay tree. Specifically, instead of sending SNMP object identifiers and node ids back up the tree, during the subscription step, the OC publishes a map of the type $\{node,\ object\text{-}id \rightarrow hash\text{-}value\}$. When this object is updated, the new values is published using the hash value only. The OC stores this map in memory, and upon receiving an update, can decipher which counter/object has changed.

Lastly, we implement an optional bandwidth reduction step in which values are only reported if they change by a pre-defined fraction. This *quantization* step assures that only interesting changes are sent back to the NMS.

Fig. 5 gives example messages that are encountered in the different phases of overlay based network monitoring. The messages in the subscription phase are from the NMS node to the overlay nodes and contain fields

Subscription Phase : NMS ---> Overlay Node

agent_id(4)	obj_id(>8)	hash(1)
172.16.16.2	1.3.6.1.2.1.2.2.2.5	0

Polling Phase : Overlay Node ---> Agent

SNMP_hdr(>=20)	req_id(4)	err_status(4)	err_index(4)	obj id(>8)	value(4)
	0	0	0	1.3.6.1.2.1.2.2.2.16	0
	Get			interface[2].ifOutOctets	

Response Phase : Agent ---> Overlay Node

SNMP_hdr(>=20)	req_id(4)	err_status(4)	err_index(4)	obj id(>8)	value(4)
	0	0	0	1.3.6.1.2.1.2.2.2.16	100000000

Publish Phase : Overlay Node ---> NMS

hash(1)	value(4)
0	10000000

Fig. 5. Monitoring related messages

identifying the agents to be contacted, object identifiers to identify the object instances to be polled, and the hash values for the object identifiers. The polling phase has SNMP poll messages going from every overlay node to each of the SNMP agents assigned to it. The agents fill in the value fields in the poll messages and transmit them in the response phase. Finally, in the publish phase, the overlay nodes forward the processed data to the NMS node on the overlay tree. The paths for polling messages and overlay messages are shown in Fig. 4. The subscription messages are unicast directly to the overlay nodes and the poll responses take the regular network path from the agents to the overlay nodes.

V. Simulation Results

In this section, we present a set of simulation results that compare the overhead of plain SNMP monitoring versus overlay-based monitoring. We use monitoring trace data from a relatively large AS (the Univ. of Maryland campus network, 400+ routers, 9000+ hosts) as input to our simulations, and compare the overhead of SNMP versus overlay-based monitoring on this topology. In our results, we consider the total bandwidth hops and message hops consumed by the two different types of monitoring, and also consider the processing and bandwidth overhead at the NMS. We describe our input data in the next section, and present detailed simulations in Section V-B.

A. Simulation Setup

We simulated our overlay network based monitoring on the Univ. of Maryland campus network with four hundred thirty-eight routers. We compared the results against using centralized SNMP where the NMS is the only manager in the network. Twenty-nine of the four hundred thirty-eight routers form the backbone of the AS. Subsets of the remaining routers have independent

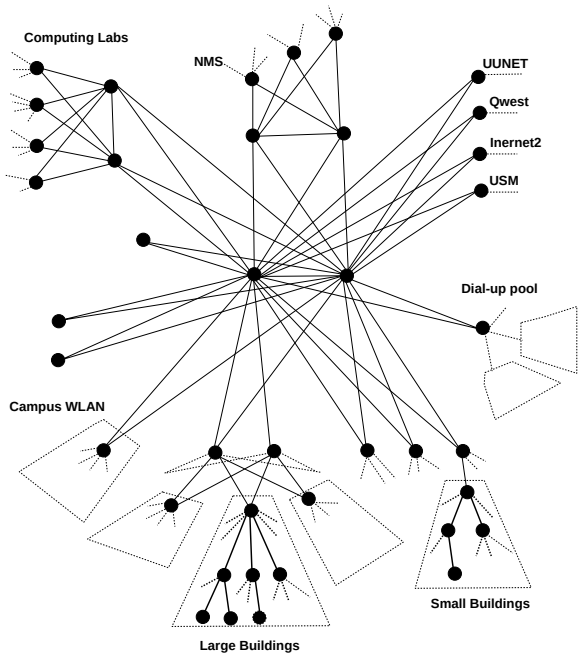

Fig. 6. Structure of UMD AS network used for the simulations

tree structures and link to one or more backbone routers. The links in the network are predominantly 100 Mbps; the backbone nodes are connected using 1 Gbps links. The structure of this network is shown in Fig. 6.

The SNMP data for this network is collected over a period of seven days in the first week of May 2003. For the backbone routers, we polled data for transmitted and received bytes, unicast and non-unicast packets, error packets, and packets discarded on the receiving as well as transmitting channels of each of the interfaces. For non-backbone switches, we polled transmitted and received bytes, and the count of packet errors. While the backbone routers are polled on average every minute, other switches are polled approximately once every five minutes.

B. Results

For these simulations, we use the *bytes transmitted* as the monitoring parameter. There were multiple reasons for choosing this particular parameter: pragmatically, we had polled this information for all interfaces and it is the most time varying data that we collected. Somewhat more importantly, this value is useful in computing link utilizations which can then be used to infer the state of the network and can also be used for load-based re-routing (See Section VI). In the simulation, the routers are polled every second for 30 minutes and we assume no communication overhead between an overlay node

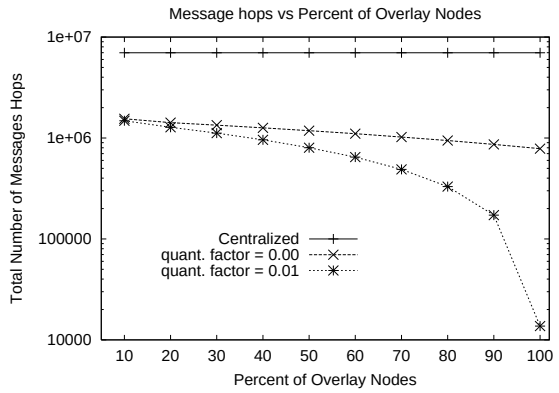

Fig. 7. Centralized SNMP vs. overlay monitoring: message hops consumed

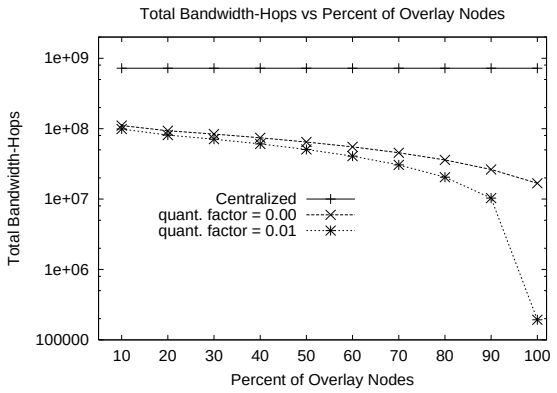

Fig. 8. Centralized SNMP vs. overlay monitoring: BW hops consumed

and its attached router since we have a dedicated link between them.

Total Monitoring Overhead*:* We first consider the overheads associated with each of the monitoring mechanisms. In Figs. 7 and 8, we plot the total message-hops and bandwidth-hops consumed by all of the monitoring messages. For the overlay protocol, we vary the number of overlay nodes and hence the number of routers associated with each overlay node. Note that the y-axes on these plots are on a log-scale, and from the plots, it is clear that overlay monitoring is more than an order of magnitude more efficient even with a very small fraction of deployed overlay nodes. (In these experiments, the overlay nodes were deployed using a greedy strategy described in Section V-D.) Each plot for the overlay scheme includes two curves, one corresponding to the case when all changes are reported back to the NMS (quantization=0.0), and one in which values are reported only if they change by 1% (quantization = 0.01). With the 1% quantization, which we believe is a useful measure for any practical algorithm, a fully deployed overlay scheme is 100 *times* more efficient than simple SNMP. While such efficiencies may not be useful if nodes are monitored every minute or so, they are certainly of great importance for sophisticated online traffic engineering, where a large network might be monitored every few hundred milliseconds. Algorithms that can work with such detailed data are known [3], [4], but are not usually deployed partly because the monitoring overhead is prohibitive; our scheme allows these algorithms to be practically deployed without changing existing hardware.

In Figs. 9 and 10, we plot the number of messages received by the NMS node and the bandwidth consumed by these messages. For base SNMP (centralized), these messages include poll messages sent by the NMS nodes and their respective response messages. Overlay monitoring is based on the publish-subscribe paradigm, and the NMS node subscribes with the overlay nodes only once. Hence, the number of messages sent is the same as the number of overlay nodes and the bandwidth is proportional to the number of variables in the network being monitored. Once again, the y-axis is on a log-scale, and the benefit due to the overlay monitoring is two (or more) orders of magnitude. Moreover, this benefit is essentially constant after a relatively small fraction (10%) of the routers in the AS have an associated overlay node. As expected, the quantization helps, and in this case reduces the overhead by a further factor of up up to 10.

Monitoring Paths and Trees*:* We extend our analysis to the case when a single unicast or multicast path is monitored for bottlenecks. Often, it is important to find the bottleneck link on a unicast or multicast path. This information can then be used to reroute around this link, or to reassign new traffic, etc. Using overlay monitoring, we can compute the bottleneck information inside the network and inform the NMS of this precise information. Specifically, the NMS assigns a unique id to monitored paths (trees) and informs specific overlay nodes to publish bottleneck information. All overlay nodes along the monitored paths (trees) send bottleneck information about the path segment (tree-component) they monitor up the overlay tree towards the NMS. The individual paths are identified using the path-id originally published by the NMS node. If the overlay node receives information (either gathered locally or from downstream overlay nodes) about multiple segments from a single path or tree, it only forwards information about the most congested link. In this manner, the path/tree bottleneck information reaches the NMS node which can then use it as needed. In comparison, SNMP-based monitoring

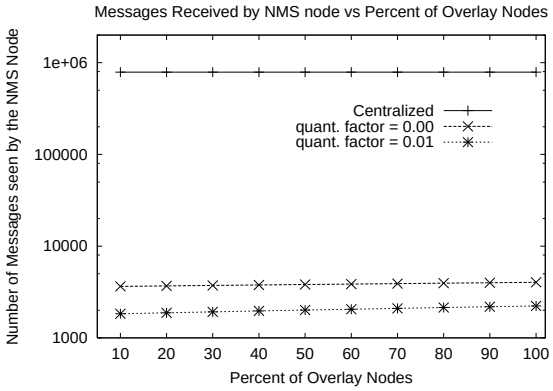

Fig. 9. Centralized SNMP vs. overlay monitoring: # messages at NMS

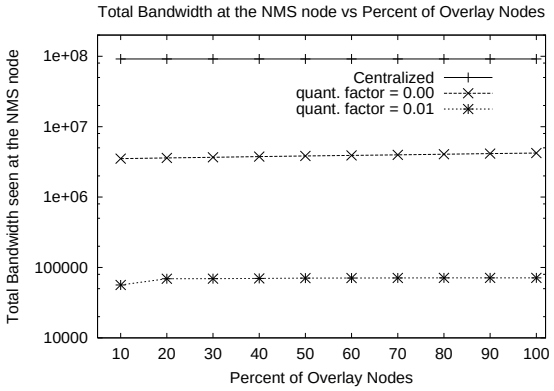

Fig. 10. Centralized SNMP vs. overlay monitoring: BW usage at NMS

requires polling all routers on the path or tree, and incurs much higher overheads. We present results for monitored paths and trees next. In these results, all routers in the network have an associated overlay node.

In Fig. 11, we present the monitoring overhead for monitoring different paths over the one week monitoring period. These paths were chosen uniformly at random from the UMD network, and ranged from 2–6 hops in length. Even with such short paths, the overlay monitoring scheme outperforms plain SNMP by two orders of magnitude. For larger domains, we expect the advantage due to overlay monitoring to increase. In Fig. 12, we plot the monitoring overhead when different numbers of multicast trees are monitored. These monitored links were chosen by constructing shortest path trees between 10 randomly chosen routers. As expected, the overlay based monitoring is more efficient, once again by about two orders of magnitude. In general, we believe the advantage due to overlay monitoring will increase as the AS size increases, and as more complex conditions are monitored.

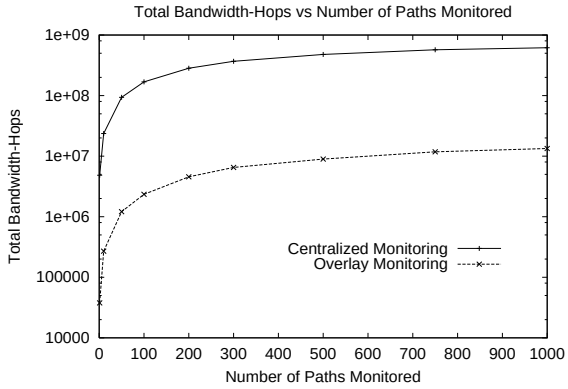

Fig. 11. Centralized SNMP vs. overlay monitoring: BW hops consumed when paths are monitored

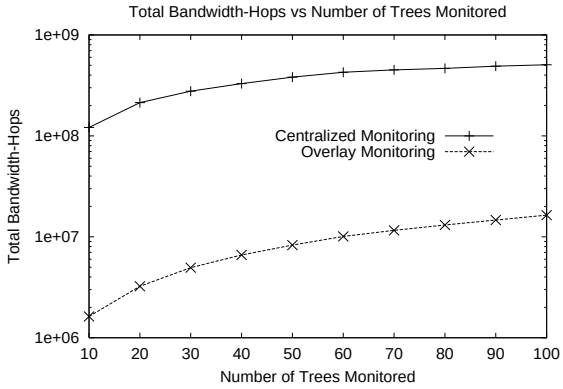

Fig. 12. Centralized SNMP vs. overlay monitoring: BW hops consumed when trees are monitored

Monitoring Trap Events*:* Efficient monitoring of asynchronous events are an important capability in network management, and overlay monitoring allows unprecedented flexibility and efficiency for defining and monitoring new asynchronous events. In this experiment, we consider a new trap which generates an event whenever the traffic on any link of the AS crosses a certain utilization level (either from above or below). In Fig. 13, we plot the total bandwidth hops consumed over the monitoring period if such events are signaled using the overlay versus plain SNMP. The overlay monitoring scheme is better by 4-6 orders of magnitude, and once again, we expect the performance to be comparatively better on larger networks. We should note that the large difference in overheads happens because trap events do not occur frequently, but still have to monitored constantly. Hence, local monitoring used in our scheme eliminates the large message overhead for monitoring these events regularly. The efficiency of overlay monitoring will allow network administrators to define more trap events, and also monitor them much more aggressively.

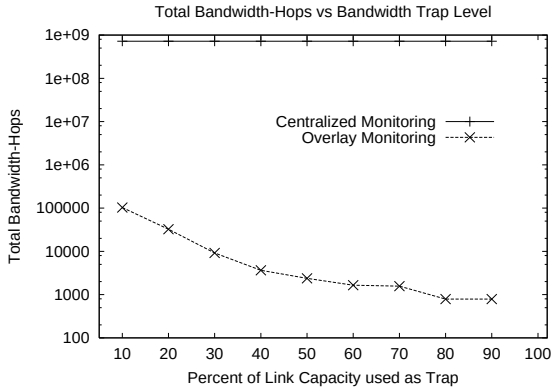

Fig. 13. Centralized SNMP vs. overlay monitoring: BW hops consumed when events are monitored

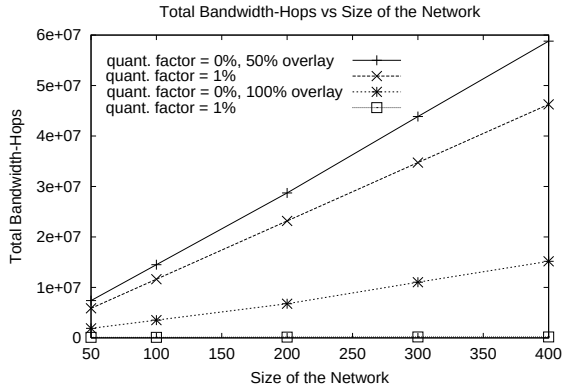

Fig. 14. Scaling of Overlay Monitoring with Network Size

C. Scalability

An important property of the overlay-based monitoring is its scalability: once deployed, it is a fully distributed scheme which we believe would work well for large networks. Unfortunately, we did not have access to a larger AS than the Univ. of Maryland network to test our scalability hypothesis; instead we used subsets of our available data to infer scalability properties. Specifically, we chose five router subsets of size 50, 100, 200, 300, 400. We plot the bandwidth hops consumed for monitoring each subset network when 50% and 100% overlay nodes are deployed. As can be seen from figure 14, the resources consumed by the monitoring messages linearly increases. This is clear from the fact that, as the network becomes larger, more links have to be monitored and the monitoring information has to travel longer paths. However, with a 100% deployment and using quantization, along with the resources consumed, rate of increase is also small. For this example, the resource usage increased by 77% when the network size increased by a factor of 8.

D. Overlay Node Placement

In all experiments in this section, we have employed a simple, greedy strategy for placing overlay nodes. Specifically, we place the next overlay node at the router with the highest degree (interface count) that does not already have an overlay node. We believe this is an intuitive strategy since these routers connect more networks and are likely to require most attention. We have also experimented with placing overlay nodes purely uniformly at random, and as shown in Fig. 15, such placement can increase bandwidth usage by a factor of 2 when the number of overlay nodes is small.

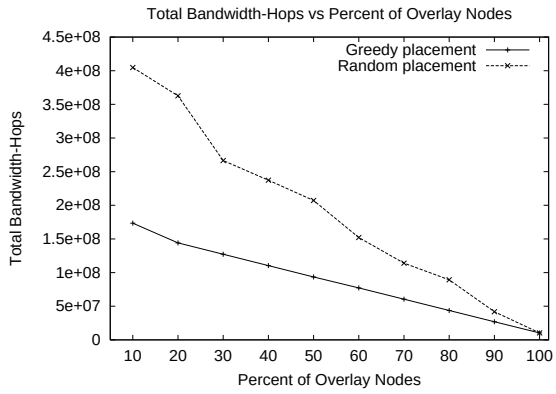

Fig. 15. Comparison of Random vs Greedy placement of Overlay Nodes

VI. Traffic Engineering

In this section, we briefly consider the traffic engineering aspect of overlay applications. This is a complete topic by itself and we only present a single example that illustrates both the benefits of in-network overlay and of our monitoring scheme. Specifically, we describe the Simultaneous Perturbation Stochastic Approximation (SPSA) optimal routing algorithm [3] developed in the context of our overlay network architecture and present results that exhibit the effectiveness of this algorithm.

Consider the network in Fig. 16. Traffic on the default paths from r_3 and r_5 to r_8 share the r_5-r_8 path. If this r_5-r_8 path is congested, the overlay node N_3 redirects the traffic to N_1 which forwards the packets to their destination, r_8.

SPSA based Optimal Routing*:* The goal of the algorithm is to balance traffic among the several available alternate paths between ingress-egress pairs to achieve optimal routing. Each ingress node of the AS identifies alternate overlay paths to each egress. The ingress then

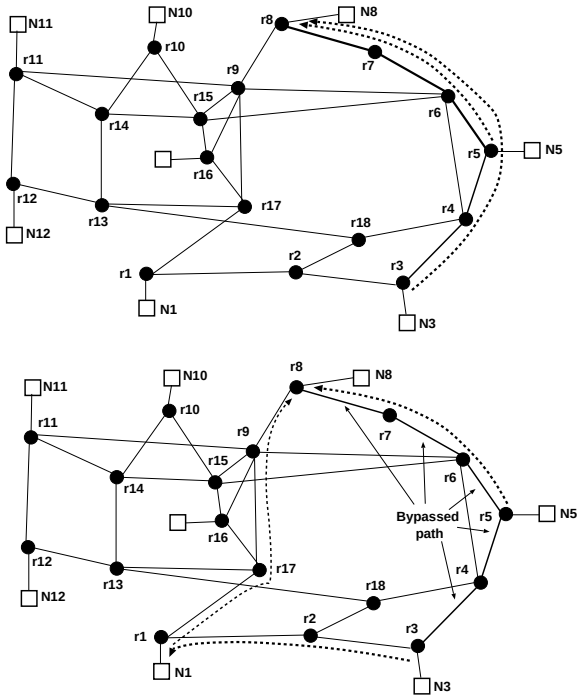

Fig. 16. Traffi c Engineering using overlay network

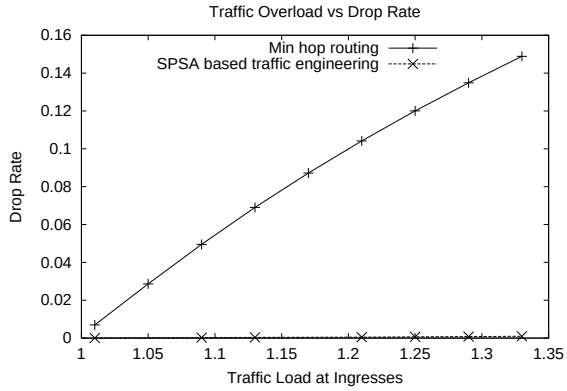

Fig. 17. A comparison of SPSA with Minimum Hop routing

executes the SPSA algorithm for all the ingress-egress flows originating at itself. For each ingress-egress flow, the ingress incrementally moves traffic from paths with higher costs to paths with lower costs until the algorithm converges. While the cost metric for the algorithm is dependent on what the algorithm is optimizing, we use a combination of packet loss and link utilizations on the path. The rate at which the cost metrics are calculated and how soon the algorithm converges is dependent on the frequency at which the state is updated at the ingresses by the overlay nodes. More details of the algorithm can be found in [3].

We implemented this algorithm on a packet level simulator and compared the algorithm against minimum hop routing using drop rates. The AS topology used for the simulation is the 19 node network shown in Fig. 16 and the simulation time is 15 minutes. The arrival traffic at the ingresses has Poisson distribution and traffic load is varied from 1.00 to 1.35 times the bandwidth on the minimum hop path of the ingress-egress flows. Fig. 17 plots the fraction of incoming packets dropped inside the network for SPSA based optimal routing and minimum hop routing as the traffic load at ingresses is varied. The drops observed when the traffic load is 1.00 is because of the Poisson nature of the input traffic.

This result is an indication that load-based rerouting can significantly improve performance within an AS. These algorithms require extremely efficient monitoring (in these experiments, each link is monitored every second) that is simply not possible using SNMP-based polling. We believe an overlay-based monitoring architecture is a promising approach to introduce realistic traffic engineering within IP-domains.

VII. Related Work

Rerouting based on overlays has been proposed for Resilient Overlay Network (RON [10]) and Detour [11] projects. The goal of RON is to form overlay networks of RON clients and reroute traffic among the clients based on the quality of the paths between the clients. RON clients are end-hosts and RON networks have group sizes of up to 50 RON clients. Detour is similar to RON except that Detour nodes form peers at the network layer. The Detour network has no restriction on the group size and spans the entire Internet. While Detour nodes can be network elements, the majority are end-hosts. Both RON and Detour are interdomain rerouting approaches and use active probing to determine the quality of routes among the members. Our overlay network is an intradomain in-network overlay framework and is thus different from both RON and Detour. In our work, we form an in-network overlay using hosts co-located with routers. Our approach also differs from both RON and Detour in that, it is administered by a single entity which makes configuration and maintenance simpler. Our approach utilizes the network state information available at the AS level instead of exclusively relying on active probing as done in RON and Detour. In general, Detour and RON are end-to-end techniques for improving application performance in the wide-area; our work requires in-network deployment and administrative support, and is specifically designed to be a distributed network management and monitoring platform.

Network monitoring has been extensively studied in recent literature. Asgari, et. al., in [12], propose an architecture similar to ours for network monitoring. In their architecture, the monitor nodes that augment the routers play a similar role in monitoring the routers and processing the messages. Unlike this work, we have described how overlay nodes can coordinate in processing and forwarding monitoring messages in a distributed manner, and how to implement more sophisticated queries and traps. Finally, unlike our work, the framework described in [12] does not include services for policy-based rerouting, or network management.

Dilman, et. al. [13] propose using reactive monitoring to decrease the monitoring overhead. The network elements generate reports in reaction to certain events in the network and send the reports to the management station. The management station then polls individual routers. This approach assumes that the network elements implement traps and event reporting but such functionality exists only for a limited set of events. The authors in [14] and [15] use mobile agent technology to reduce the monitoring overhead in the network. These approaches assume that routers have enough processing capabilities to execute agent code, and in general, require a new or upgraded infrastructure.

VIII. Conclusions

In this paper, we have presented an architecture for intra-domain overlay networks and have outlined various uses for such a structure. We have described the functional components of overlay nodes, and described protocols for instantiation and maintenance of the overlay network. We have shown how all of these applications, including policy- and load-based re-routing, efficient monitoring, etc. can be implemented without changing existing IP infrastructure.

Along with the overlay architecture, the primary contribution of this paper is an algorithm that uses the overlays to provide extremely efficient monitoring of large domains. We have presented simulations from input data gathered from a large AS, and shown that our schemes reduce bandwidth and processing requirements by 2 to 6 orders of magnitude. These gains permit much finer granularity monitoring than is currently feasible. We have described how such monitoring data can be used to implement sophisticated load-based routing.

Lastly, we believe in-network trusted nodes are an ideal platform for deploying monitors for network security, specifically for distributed intrusion detection. The benefits of such an approach are likely to be greatest in very large domains, where many compromised hosts can exist within the domain. This is a promising application for intradomain overlays, and is an avenue for future work.

References

[1] K. Calvert, S. Bhattacharjee, E. Zegura, and J. Sterbenz, "Directions in active networks," 1998, IEEE Communications Magazine.

[2] E. Rosen, A. Viswanathan, and R.Callon, "RFC 3031:multiprotocol label switching architecture," Jan. 2001.

[3] T. Guven, C. Kommareddy, R. La, M. Shayman, and S. Bhattacharjee, "Measurement based optimal multi-path routing," UM Institute of Advanced Computing Sciences, Tech. Rep. UMIACS-TR-2003-69, July 2003.

[4] A. Elwalid, C. Jin, S. H. Low, and I. Widjaja, "MATE: MPLS adaptive traffic engineering," in *INFOCOM*, 2001, pp. 1300–1309.

[5] S. Phuvoravan, K.-T. Kuo, T. Guven, L. Sudarsan, H. S. Chang, S.Bhattacharjee, and M. A. Shayman, "Fast timescale control for MPLS traffic engineering," in *Globecom*, 2003.

[6] K. Calvert, J. Griffioen, B. Mullins, A. Sehgal, and S. Wen, "Concast: Design and implementation of an active network service," *IEEE Journal on Selected Area in Communications*, vol. 19, no. 3, mar 2001.

[7] K. Calvert, J. Griffioen, and S. Wen, "Lightweight network support for scalable end-to-end services," in *Proceedings of SIGCOMM*, 2002.

[8] K. McCloghrie, D. Perkins, and J. Schoenwaelder, "RFC 2578: Structure of Management Information version 2 (SMIv2)," Apr. 1999.

[9] D. Harrington, R. Presuhn, and B. Wijnen, "RFC 3411: An architecture for describing simple network management protocol (snmp) management frameworks," Dec. 2002.

[10] D. Andersen, H. Balakrishnan, M. Kaashoek, and R. Morris, "The case for resilient overlay networks," in *Proceedings of the 8th Annual Workshop on Hot Topics in Operating Systems (HotOSVIII)*, May 2001.

[11] 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," *IEEE Micro*, vol. 19, no. 1, Jan. 1999.

[12] A. Asgari, P. Trimintzios, M. Irons, G. Pavlou, R. Egan, and S. V. den Berghe, "A scalable real-time monitoring system for supporting traffic engineering."

[13] M. Dilman and D. Raz, "Efficient reactive monitoring," in *INFOCOM*, 2001, pp. 1012–1019.

[14] T. M. Chen and S. S. Liu, "A model and evaluation of distributed network management approaches," *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 4, pp. 850–857, may 2002.

[15] S. Papavassiliou, A. Paliafito, O. Tomarchio, and J. Ye, "Mobile agent-based approach for efficient network management and resource allocation: Framework and applications," *IEEE Journal on Selected Areas in Communication*, vol. 20, no. 4, pp. 858–872, may 2002.