Topology Discovery in Heterogeneous IP Networks: The *NetInventory* System

Yuri Breitbart, Minos Garofalakis, *Member, IEEE*, Ben Jai, Cliff Martin, Rajeev Rastogi, and Avi Silberschatz, *Fellow, IEEE*

***Abstract*—Knowledge of the up-to-date physical topology of an IP network is crucial to a number of critical network management tasks, including reactive and proactive resource management, event correlation, and root-cause analysis. Given the dynamic nature of today's IP networks, keeping track of topology information manually is a daunting (if not impossible) task. Thus, effective algorithms for automatically discovering physical network topology are necessary. Earlier work has typically concentrated on either 1) discovering logical (i.e., layer-3) topology, which implies that the connectivity of all layer-2 elements (e.g., switches and bridges) is ignored, or 2) proprietary solutions targeting specific product families. In this paper, we present novel algorithms for discovering physical topology in heterogeneous (i.e., multi-vendor) IP networks. Our algorithms rely on standard SNMP MIB information that is widely supported by modern IP network elements and require no modifications to the operating system software running on elements or hosts. We have implemented the algorithms presented in this paper in the context of the *NetInventory* topology-discovery tool that has been tested on Lucent's own research network. The experimental results clearly validate our approach, demonstrating that our tool can consistently discover the accurate physical network topology with reasonably small running-time requirements even for fairly large network configurations.**

***Index Terms*—IP network management, physical network topology, SNMP MIBs, switched Ethernet.**

I. Introduction

PHYSICAL NETWORK TOPOLOGY refers to the characterization of the physical connectivity relationships that exist among entities in a communication network. Discovering the physical layout and interconnections of network elements is a prerequisite to many critical network management tasks, including reactive and proactive resource management, server siting, event correlation, and root-cause analysis. For example, consider a fault monitoring and analysis application running on a central IP network management platform. Typically, a single fault in the network may cause a flood of alarm signals emanating from different interrelated network elements. Knowledge of element interconnections is essential to filter out secondary alarm signals and correlate primary alarms to pinpoint the original source of failure in the network [14], [29]. Furthermore, a full physical map of the network enables a proactive analysis of the impact of link and device failures. Early identification of critical failure points allows the network manager to improve the survivability of the network (e.g., by adding alternate routing paths) before outages occur.

Despite the critical role of topology information in enhancing the manageability of modern IP networks, none of the network management platforms available on the market today offers a general-purpose tool for automatic discovery of physical IP network connectivity. Most systems (including HP's OpenView Network Node Manager and IBM's Tivoli for AIX) either feature an IP mapping functionality for automatically discovering routers and IP subnets and generating a *network layer* (i.e., ISO *layer-3*) topology showing the router-to-router interconnections and router interface-to-subnet relationships, or require that switches from different IP subnets are not directly connected. Discovering a layer-3 topology is relatively easy provided that standard routing information is available, because routers must be explicitly aware of their layer-3 neighbors in order to perform their basic function. Unfortunately, layer-3 topology covers only a small fraction of the interrelationships in an IP network, since it fails to capture the complex interconnections of *layer-2* network elements (e.g., switches and bridges) belonging to different IP subnets. As more switches are deployed to provide more bandwidth through subnet microsegmentation, the portions of the network infrastructure that are invisible to a layer-3 mapping will continue to grow. Under such conditions, it is obvious that the network manager's ability to troubleshoot end-to-end connectivity or assess the potential impact of link or device failures in switched networks will be severely impaired.

The lack of automated solutions for capturing physical (i.e., layer-2) topology information means that network managers are routinely forced to manually input such information for each management tool that they use. Given the dynamic nature and the ever-increasing complexity of today's IP networks, keeping track of topology information manually is a daunting (if not impossible) task. This situation clearly mandates the development of effective, general-purpose algorithmic solutions for automatically discovering the up-to-date physical topology of an IP network. An additional challenge in the design of such algorithms is dealing with the lack of established, industry-wide standards on the topology information maintained locally by each network element, and the diversity of elements and protocols present in today's multi-vendor IP networks. This essentially means that

Manuscript received January 17, 2002; revised January 24, 2003; approved by IEEE/ACM Transactions on Networking Editor M. Grossglauser.

Y. Breitbart is with the Department of Computer Science, Kent State University, Kent, OH 44244 USA (e-mail: yuri@cs.kent.edu).

M. Garofalakis, C. Martin, R. Rastogi are with Bell Labs, Lucent Technologies, Murray Hill, NJ 07974 USA (e-mail: minos@bell-labs.com; cliff@bell-labs.com; rastogi@bell-labs.com).

B. Jai is with Google, Inc., Mountain View, CA 94043 USA (e-mail: jai@google.com).

A. Silberschatz is with the Computer Science Department, Yale University, New Haven, CT 06520 USA (e-mail: avi@cs.yale.edu).

Digital Object Identifier 10.1109/TNET.2004.828963

the actual connectivity information maintained by different elements can vary widely; further, different elements may store such information in different (e.g., proprietary) parts of their internal database. The combination of the aforementioned factors implies that any practical solution to the problem of discovering physical IP topology needs to deal with three fundamental difficulties.

1) *Limited local information.* The algorithm should make only *minimal assumptions* about the availability of information at the elements; that is, it should only utilize information that most managed elements are most likely to maintain locally. Furthermore, since layer-2 elements are not explicitly aware of their immediate physical neighbors, inferring physical interconnections at layer-2 is definitely not straightforward.
2) *Transparency of elements across protocol layers.* The algorithm should correctly establish interconnections between network elements operating at different layers of the ISO protocol stack. This is not trivial, since layer-2 elements in switched IP subnets are completely transparent to the layer-3 router(s) directing traffic in and out of the subnets.
3) *Heterogeneity of network elements.* The discovery algorithm should be able to gather topology information from heterogeneous network elements, making sure that the relevant data collected in the elements of different vendors are accessed and interpreted correctly.

Related Work: Discovering the topology of the Internet is a problem that has attracted the attention of many networking researchers. A major motivation behind mapping out the Internet is the ability to use the Internet topology in order to study important performance bottlenecks, and find the optimal placement for various network-measurement tools and network servers. As a consequence, the bulk of the Internet-mapping work has concentrated on the automated discovery of wide-area network (WAN) topologies and, more specifically, the topology of router interconnections (i.e., layer-3 topology).

Siamwalla *et al.* [23] propose heuristics for inferring layer-3 topology that employ basic ICMP commands, like `ping` and `traceroute` [15]. A similar approach, based on ICMP primitives, for discover routers and their layer-3 connections has also been proposed by Burch and Cheswick [4]. The hop-by-hop feedback provided by the ICMP `traceroute` command is employed in the context of the Mercator project [13], whose goal is to discover adjacency between routers. More specifically, Mercator discovers adjacent router connections by sending traceroute and `ping` commands from a single network location and using only hop-limited probes; it also employs several clever heuristics to discover the layer-3 network map. The Rocketfuel project [28] uses results from 294 public `traceroute` servers to build router-level ISP topologies; it also employs techniques that can reduce the amount of probing and resolve address aliasing. A related problem is that of inferring the (layer-3) topology of *IP multicast trees*; proposed solutions to this problem rely on either correlating end-to-end multicast measurements (e.g., packet-loss characteristics) [10], or using some specialized mechanism that requires router cooperation [16]. Tutschku and Baier [26] use round-trip-time measures obtained through ICMP packets to define a metric that characterizes the performance (e.g., effective bandwidth usage) in the underlying network; however, they do not explicitly address the problem of discovering element interconnections.

Since finding a complete layer-3 topology of the Internet may be infeasible due to its enormous size, several researchers have turned to the design of network *models* that attempt to accurately capture the Internet topology. In [27], Waxman has introduced what appears to be one of the most popular early network models. Waxman graphs are generated probabilistically considering the nodes as points in a Euclidean space. Calvert *et al.* [5] discuss different graph-based models for representing the topology of large networks, focusing particularly on aspects of locality and hierarchy present in the Internet. Zegura *et al.* [30] introduce a comprehensive graph model that includes several earlier models, and combines simpler topologies (e.g., Waxman graphs and trees) in a hierarchical structure. In more recent work, Faloutsos *et al.* [11] propose several empirical *power laws* that characterize the Internet inter-domain topology, and can be used to estimate important topology parameters (such as the average neighborhood size).

SNMP-based [6] algorithms for automatically discovering layer-3 network map are featured in many common network management tools. For instance, HP's OpenView Network Node Manager (www.openview.hp.com) and IBM's Tivoli for AIX (www.tivoli.com) both offer an IP-network management functionality for automatically discovering routers and IP subnets and generating a layer-3 topology. Other commercially available tools for inferring layer-3 network topology using SNMP include Actualit's Optimal Surveyor (www.actualit.com) and the Dartmouth Intermapper (intermapper.dartmouth.edu).

On the other hand, there has been comparatively very little work on the automated discovery of *layer-2* topology for large local-area networks (LANs). Nevertheless, it is well known that the most interesting portions of LAN topology are formed by layer-2 devices (e.g., switches/bridges) which, essentially, renders them transparent to tools that only infer IP-router interconnections. Furthermore, several studies have demonstrated the impact that layer-2 topology can have on the performance of distributed and parallel tasks in LAN architectures [7], [12], [17]. Recognizing the importance of layer-2 topology discovery, a number of vendors have recently developed proprietary tools and protocols for inferring layer-2 connectivity between different network elements. Examples of such systems include Cisco's Discovery Protocol (www.cisco.com) and Bay Networks' Optivity Enterprise (www.baynetworks.com). Such tools, however, are typically based on vendor-specific extensions to *SNMP Management Information Bases* (MIBs) [6], [24] and are not useful on a heterogeneous network comprising elements from multiple vendors. Peregrine's Infratools software (www.peregrine.com), Riversoft's NMOS product (www.riversoft.com), and Micromuse's Netcool/Precision application (www.micromuse.com) claim to support layer-2 topology discovery, but these tools are based on proprietary technology to which we do not have access. In an effort to come up with some standards for physical-topology discovery, IETF has recently published an RFC proposing a standard for a physical network topology MIB (originally proposed by Cisco) [2]. Unfortunately, Cisco is, to the best of our knowledge, the only vendor to implement the proposed physical MIB design.

Loran Network Systems has recently released a software tool for discovering layer-2 topology in heterogeneous networks. Briefly, their approach is based on trying to statistically map (i.e., correlate) the traffic patterns observed at the ports of different elements in the underlying network, and probabilistically inferring connections for ports with similar traffic characteristics [21], [8]. Their approach relies on statistical correlation, so it can only infer element connections with some (high) probability; furthermore, it is not at all clear if or how their proposed method would work in the presence of interconnections between network elements belonging to different IP subnets. Shao *et al.* [22] also propose correlating benchmark measurements to determine the functional differences between a central server and a collection of machines across a LAN (referred to as *effective network views*); however, for network-management purposes, such an application-level view of the underlying network structure is often insufficient.

In the earlier, conference version of this paper [3], we have proposed the *first* algorithm that relies solely on standard SNMP information to discover the layer-2 topology of a heterogeneous, multi-subnet network. In a nutshell, our algorithm employs a set of sufficient conditions that guarantee the existence of physical connections between elements organized in multiple different subnets. In a follow-up paper, Lowekamp *et al.* [18] have extended the methodology proposed in [3] by proposing new techniques to deal with incomplete information as well as "dumb" network elements that cannot provide SNMP information (e.g., hubs). Their algorithms, however, are designed to work only in the much simpler case of *a single subnet* and can easily be shown to fail when multiple subnets are present [1].

Our Contributions: We develop novel, practical algorithmic solutions for the problem of discovering physical topology in heterogeneous IP networks comprising multiple IP subnets and different types of network elements. The practicality of our algorithms stems from the fact that they rely solely on standard information routinely collected in the SNMP MIBs [6], [24] of elements and they require no modifications to the operating system software running on elements or hosts. More specifically, our topology discovery tool utilizes information either from the *address forwarding tables* (AFTs) of elements capturing the set of medium access control (MAC—i.e., layer-2) addresses that are reachable from each element interface, or (in the absence of AFT data) from the elements' `NetToMedia` tables. The main algorithmic challenge that our tool faces is how to "stitch" such (local) information together to identify interconnected router/switch interfaces and come up with a complete physical LAN topology. The issue of heterogeneity comes into play when trying to access the address forwarding information of elements from different vendors. Even though international standards bodies have proposed a *standard* MIB design [19] with uniformly defined and named variables for collecting address forwarding data, this design is often not adhered to in commercially available elements. As a consequence, our tool often needs to gather the necessary information by accessing and interpreting MIB variables stored in vendor-specific private MIBs or custom-designed files.

Our algorithm for stitching local address forwarding information together into a complete network topology works perfectly when 1) each *switched domain* (i.e., collection of switched IP subnets connected to the "outside world" through one or more layer-3 routers) consists of a *single* switched subnet, and 2) the element address forwarding tables are *complete*; that is, they contain the full set of MAC addresses in the subnet reachable from each element interface. Unfortunately, these conditions are rarely satisfied in modern IP networks, thus forcing our solutions to deal with a number of complications that arise in practice.

- Switched domains usually comprise *multiple subnets* with elements of different subnets often directly connected to each other. This introduces serious problems, since it means that an element can be completely transparent to its direct physical neighbor(s). In fact, we prove that this situation gives rise to scenarios under which no algorithm using only address forwarding information can identify a unique physical topology. We do, however, propose an engineering solution that extends our approach to multiple subnets and identifies a small set of candidate network graphs which is guaranteed to contain the correct topology. Furthermore, we provide a succinct characterization of a broad class of networks for which our algorithm is guaranteed to *uniquely* identify the accurate physical topology.
- Element address forwarding tables typically employ an *aging* mechanism to evict infrequent destination MAC addresses from the address cache; thus, the sets of MAC addresses found in these tables are not necessarily complete. We develop several different techniques to handle this problem. Our first set of techniques tries to guarantee that the address forwarding information collected is reasonably complete. This can be accomplished by either 1) generating extra network traffic across switches (using the IP `ping` mechanism) to ensure that element AFTs are adequately populated, or 2) collecting element AFT information continuously (e.g., at regular intervals) to ensure that the majority of relevant MAC addresses are seen. Our second set of techniques extends our topology discovery algorithm so that interconnection decisions are made based on *incomplete information* by employing some reasonable approximations and heuristics. Note that, since it is very unlikely to guarantee the completeness of address forwarding information without an inordinate amount of extra traffic, a hybrid of these techniques is likely to work best in practice.
- *Virtual LANs* (VLANs) allow IP network managers to completely break the linkage between the physical and logical network by grouping the interfaces of the same physical network element into different subnets. Our topology discovery algorithm can readily handle VLANs if the VLAN interface groupings are known. (This information is available in most proprietary MIBs.)

Our topology-discovery algorithms are at the heart of the ***NetInventory*** network management system that has been implemented and extensively tested over Lucent's research network at Murray Hill, NJ. The results have been very encouraging and, in fact, ***NetInventory*** is now an integral part of Lucent's network-management product portfolio.

Organization: The remainder of this paper is organized as follows. Section II reviews necessary background information and presents our system model. In Section III, we develop our algorithm for discovering the physical topology of a single subnet. Section IV then extends our algorithm to handle multiple subnets in a switched domain and identifies a broad class of networks for which our algorithm is guaranteed to discover the unique physical topology. In Section V, we discuss how our solution can be extended to deal with incomplete information and VLANs. Sections VI and VII give an architectural overview of the ***NetInventory*** network-management system and present some experimental results. Finally, Section VIII concludes the paper.

II. Background and System Model

In this section, we present necessary background information and the system model that we adopt for the network topology discovery problem. We refer to the domain over which topology discovery is to be performed as an *administrative domain*. We model an administrative domain network as an undirected graph N. The nodes in the network correspond to network elements that can be one of three types: *routers*, *switches*, and *hosts*.[1] A direct physical connection between a pair of interfaces belonging to different network elements is modeled as an edge between the corresponding nodes in N. The goal of our algorithms is to discover the nodes and edges of N as accurately as possible.

We define a *switched domain* to be the maximal set S of switches such that there is a path in N between every pair of switches in S, involving only switches in S. Fig. 1 shows the graph corresponding to an example administrative domain. In Fig. 1, R1, R2, and R3 are routers, while S1 through S5 are switches forming two distinct switched domains ($\{\mathrm{S1}, \mathrm{S2}, \mathrm{S3}\}$ and $\{\mathrm{S4}, \mathrm{S5}\}$). We define an *IP subnet* as a maximal set of IP addresses such that any two nodes within a subnet can communicate (at layer-3 or above) with each other without involving a router. Typically, every network element within an administrative domain is identified with a single IP address and a subnet mask that defines the IP address space corresponding to the element's subnet. For example, IP address 135.104.46.1 along with mask 255.255.255.0 identifies a subnet of network elements with IP addresses of the form 135.104.46.x, where x is any integer between 1 and 254. Note that a switched domain can comprise multiple different subnets and communication across these different subnets must go through a router. For example, in Fig. 1, the switched domain $\{\mathrm{S4}, \mathrm{S5}\}$ contains only one subnet while the switched domain $\{\mathrm{S1}, \mathrm{S2}, \mathrm{S3}\}$ consists of two subnets, one containing S1 and S3, and the other one containing S2. Therefore, a packet from S1 to S2 will have to be routed through R1 and R2, despite the existence of a direct physical connection between S1 and S2.

[1]To simplify the exposition, we do not explicitly consider unmanaged network elements (e.g., hubs) in the remainder of this paper; the detailed treatment of hubs in ***NetInventory*** is discussed in [1].

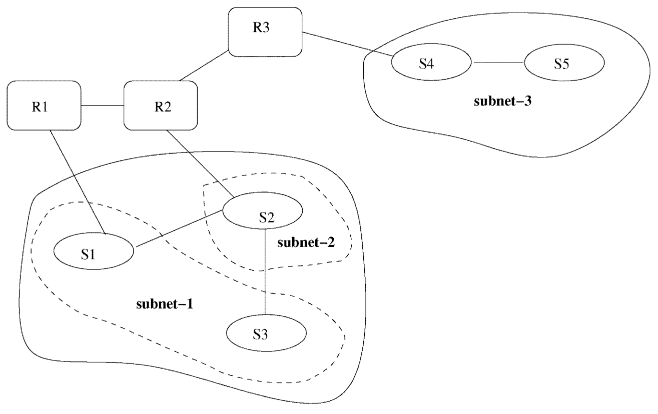

Fig. 1. Network graph for a typical administrative domain.

Switches within a switched domain typically employ the *spanning tree protocol* to determine unique forwarding paths for each switch [20], [25]. Our topology discovery algorithm is based on the MAC addresses learned using backward learning [20], [25] on interfaces that are part of the switched domain spanning tree. Therefore, it follows that we do not discover edges between interfaces that are not active (i.e., are eliminated by the spanning tree protocol) at the time of network discovery. In the remainder of the paper, we use N to refer to the administrative domain graph with all such inactive edges removed. We also assume that the structure of N remains stable during the course of topology discovery.

We denote the jth interface of a switch S_i by S_{ij}. For each interface S_{ij}, the set of addresses that have been learned (by backward learning) on that interface is referred to as the AFT corresponding to S_{ij} and is denoted by A_{ij}. Therefore, A_{ij} is the set of MAC addresses that have been seen as source addresses on packet frames received at S_{ij}. We say that A_{ij} is *complete* if A_{ij} contains the MAC addresses of *all* network nodes from which frames can be received at S_{ij}. If the switched domain comprises only one subnet, then A_{ij} corresponds to the set of nodes in N that are reachable from S_i via the interface S_{ij} by a path in the switched domain spanning tree. In the case of multiple subnets, however, the above is not necessarily true. For example, in Fig. 1, S3 will never receive a frame from S2 with S2 as the source MAC address. The reason is that, if S2 has to communicate with S3, then the packet from S2 is first sent to R2, which in turn forward it to R1 and, finally, R1 forward the packet to S3 with the source MAC address being that of R1 (even though the packet passes through S2).

III. Single Subnet Switched Domains

In this section, we describe a topology discovery algorithm under the following assumptions: 1) each switched domain contains exactly one subnet; 2) no VLANS are present in the administrative domain; and 3) the address forwarding tables are complete. (Of course, as already discussed in Section II, we also assume that the topology of the network does not change during

```
Procedure FindRouters($R_1$)
/* $R_1$ is the IP address of some known router in the administrative domain */
begin
1.  routerSet := $\{R_1\}$ , routersVisited := $\phi$
2.  while routerSet $\neq \phi$ do {
3.      choose a router $R$ from routerSet
4.      routerSet := routerSet - $\{R\}$
5.      if ( $R \in$ routersVisited ) then continue
6.      routersVisited := routersVisited $\bigcup \{R\}$
7.      $NH(R)$ := next hops for $R$ for some destination
8.      routerSet := routerSet $\bigcup NH(R)$
9.  }
end
```

Fig. 2. Discovering routers in an administrative domain.

the discovery process.) We first briefly describe how we discover the set of network elements in N and then describe our algorithm for discovering the (active) edges of N.

The basic idea behind discovering the set of routers in the administrative domain is to repeatedly find the neighboring routers of the currently known routers until no new routers are discovered. We assume that we know the IP address of at least one router, say, R_1, in the administrative domain to bootstrap this process[2]. The neighboring routers of a router R are the set of routers that are next hops for some destination in the `ipRouteTable` in MIB-II [19] in R. Fig. 2 outlines our algorithm for discovering the set of routers in the administrative domain.

The switches in the administrative domain are identified by first discovering, for each interface of a router R, the subnet that it is directly connected to or, equivalently, the set of IP addresses D to which it can perform direct delivery. This is obtained by first getting the IP address of an interface of R using the `ipAddrTable` in MIB-II. The set D is then computed by enumerating the set of IP addresses in the subnet corresponding to the IP address of an interface. This enumeration takes into account the subnet masks and the IP address formats. For each IP address, we use the element MIB to obtain a MAC address (from the `ipNetToMediaTable`), a subnet, a system name, and the number of ports. Once D is computed, for each IP address in D, we determine whether it corresponds to a switch by checking for the presence of the Bridge MIB [9]. Actually, both routers and switches may contain the Bridge MIB and, therefore, we use the value of the `ipForwarding` and the `system.sysServices` variables to determine if a node is a switch or a router. If `ipForwarding` is not 1 or the second bit of `system.sysServices` is set, then the element in question is a switch, otherwise it is a router.

At this juncture, we have essentially discovered the set of all network elements in the administrative domain N. We next describe how to discover the active interconnections between such elements.

A. Discovering Spanning-Tree Edges

We discover the edges of N, one switched domain (in this case, one subnet) at a time. Let $\mathcal{U}$ be the set of MAC addresses corresponding to the switches and the routers of a subnet S. We begin the description of our edge discovery algorithm with a

[2]We assume N is a connected graph; otherwise, we need to know the identity of one router in each connected component.

```
Procedure FindInterConnections($S_1, S_2, \ldots S_n$, $R_1, R_2, \ldots, R_m$)
/* $S_1, S_2, \ldots, S_n$ are the switches of subnet $S$ */
/* $R_1, R_2, \ldots, R_m$ are the routers of subnet $S$ */
begin
1.  for each switch $S_i$ do
2.      for each interface $j$ of $S_i$ do {
3.          if ( $S_{ij}$ has already been matched ) then continue
4.          else if ( $A_{ij} \bigcup A_{kl} = \mathcal{U}$ and $A_{ij} \bigcap A_{kl} = \phi$ ) then
5.              match $S_{ij}$ with $S_{kl}$        /* $S_{ij}$ and $S_{kl}$ are connected */
6.      }
7.  for each router $R_k$, switch $S_i$ do
8.      for each interface $j$ of $S_i$ do
9.          if ( $S_{ij}$ is not matched and $A_{ij}$ contains $R_k$ ) then
10.             match $S_{ij}$ with $R_k$        /* $S_{ij}$ and $R_k$ are connected */
end
```

Fig. 3. Discovering switch and router interconnections.

lemma that establishes a necessary and sufficient condition for an interface of a switch element to be connected to an interface of another switch.

Lemma 3.1: Interfaces S_{ij} and S_{kl} are directly connected to each other if and only if $A_{ij} \cup A_{kl} = \mathcal{U}$ and $A_{ij} \cap A_{kl} = \phi$. ■

Proof: If S_{ij} and S_{kl} are directly connected to each other then, clearly, $A_{ij} \cap A_{kl} = \phi$. Further, since all address forwarding tables are complete, $A_{ij} \cup A_{kl} = \mathcal{U}$.

To prove the other direction, assume that $A_{ij} \cup A_{kl} = \mathcal{U}$ and $A_{ij} \cap A_{kl} = \phi$, but S_{ij} and S_{kl} are not directly connected. Since the active spanning tree of the switched domain does not contain loops, A_{ij} (A_{kl}) does not contain S_i (respectively, S_k). As a consequence, A_{ij} contains S_k and A_{kl} contains S_i. Let P be the path between S_i and S_k in the spanning tree. There are two cases to consider.

1) *P contains both S_{ij} and S_{kl}.* In this case, there exists another switch S_m in P and, therefore, it cannot be the case that $A_{ij} \cap A_{kl} = \phi$.
2) *P contains either exactly one of S_{ij} and S_{kl}*, or neither of the two. In this case, since nodes S_i and S_k are connected and, furthermore, $S_i(S_k)$ is reachable from S_{kl} (respectively, S_{ij}) (since $S_i \in A_{kl}, S_k \in A_{ij}$), we have found two distinct paths (i.e., a loop) in the spanning tree of the switched domain between S_i and S_k. This is again a contradiction. ■

Lemma 3.1 gives us the basis for a simple algorithm to discover direct connections between switches. However, we still need to discover connections between routers and switches. We now describe the condition for a router to be connected to a switch.

Definition 3.2: A leaf interface of a switch S_i is an interface that is not connected to an interface of any other switch. ■

Clearly, an interface S_{ij} for which there does not exist another interface S_{kl}, such that A_{ij} and A_{kl} satisfy the conditions specified in Lemma 3.1 is a leaf interface. We can now state a necessary and sufficient condition for a router to be connected to a switch.

Lemma 3.3: A router R is connected to an interface S_{ij} if and only if S_{ij} is a leaf interface and A_{ij} contains the MAC address of R. ■

Fig. 3 gives the pseudo-code for the single-subnet, edge-discovery algorithm based on Lemmas 3.1 and 3.3 (termed FindInterConnections).

```
Procedure FindLeafConnections($S_1, S_2, \ldots S_n$ )
/* $S_1, S_2, \ldots, S_n$ are the nodes of a subnet $N$ */
begin
1.  $Current := \{S_1, S_2, \ldots, S_n\}$
2.  while ( $Current$ is not empty ) do {
3.      find AFT $A_{ij}$ with minimal number of entries
4.      if ( $A_{ij} = \{S_t\}$ ) then {        /* $A_{ij}$ contains a single entry */
5.          create a connection between $S_i$ and $S_t$
6.          eliminate node $S_t$ from all remaining AFTs
7.          continue
8.      }
9.      else {
10.         let $A_{ij} = \{S_{t_1}, \ldots, S_{t_k}\}$
11.         create a hub and use it to connect all $S_i, S_{t_1}, \ldots, S_{t_k}$
12.         eliminate $S_{t_1}, \ldots, S_{t_k}$ from all remaining AFTs
13.     }
14. }
end
```

Fig. 4. Alternative algorithm for single-subnet discovery.

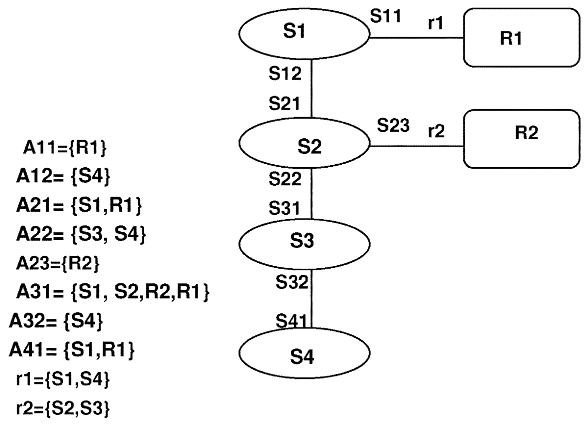

Fig. 5. Example network containing multiple subnets.

The FindInterConnections procedure can discover connections between switches and routers as well as hosts. In the remainder of this section, we discuss a faster, simple alternative algorithm (termed FindLeafConnections) to discover the topology of a single-subnet network that, in fact, can also deal with the case of unmanaged hubs. Our FindLeafConnections algorithm relies on the following lemma.

Lemma 3.4: Let A_{kl} be an address forwarding table of minimal cardinality among all other address forwarding tables in N. Then, every element in A_{kl} has a single interface in the switched domain spanning tree. ■

Proof: Let A_{kl} be the address forwarding table that contains the smallest number of MAC addresses among all interface AFTs in N. Let S_t be a network node (whose address is) contained in A_{kl}. We claim that S_t has a single interface S_{ti} in the spanning tree. Assume, to the contrary, that S_t has at least two such interfaces; consequently, S_t cannot be a leaf node. Thus, there exists a node S_p that is attached to S_t through an interface other than S_{ti}. But, in this case, A_{kl} is clearly not minimal, since there exists a port of S_t whose address forwarding table contains a proper subset of the MAC addresses in A_{kl}, that does not contain S_t. Thus, S_t can only have a single interface in the spanning tree. This completes the proof. ■

The pseudo-code for procedure FindLeafConnections is depicted in Fig. 4. Briefly, the algorithm starts with finding an address forwarding table A_{ij} of smallest cardinality. If such a table contains a single element, then a connection is created and, since the discovered single-port node has already been connected, it is excluded from every other AFT that it belongs to. If the number of elements in this minimal AFT A_{ij} exceeds one, then all single-port elements along with the minimal-AFT port are connected by a hub, and all such single-port elements are excluded from every other AFT that they are discovered in. (Note that, if S_{ij} is the final port of the S_i node to be connected, then S_i itself can also be excluded.) Further, if there already exists a hub connection containing at least one of these elements, then the two hubs are merged into a single hub.

IV. Multiple Subnet Switched Domains

As described in the previous section, for switched domains containing a single subnet, network-element interconnections are fairly easy to determine. Interfaces S_{ij} and S_{kl} are connected if and only if the union $A_{ij} \cup A_{kl}$ contains all the nodes in the subnet and the intersection $A_{ij} \cap A_{kl}$ is empty.

Unfortunately, the assumption that nodes in a switched domain are always from a single subnet may not always hold. As an example, consider the network depicted in Fig. 5. Switches S_1 and S_4 belong to subnet 1, while S_2 and S_3 belong to subnet 2. The algorithm from the previous section will not be able to connect interfaces S_{21} to S_{12}, as it should. The reason for this is that switches S_2 and S_3 do not show up in the address forwarding table of switch S_1. (Since S_2 and S_3 belong to a different subnet, frames originating at S_2 and S_3 are actually routed to switch S_1 through the R_2 router.) Even if we were to consider a modification of the previous algorithm in which two interfaces are connected if the union of their address forwarding tables includes all the nodes in some subnet, the method would still not work. Since $A_{12} \cup A_{21}, A_{12} \cup A_{31}$, and $A_{12} \cup A_{41}$ contain all the nodes in subnet 1, the modified algorithm would find that interfaces S_{21}, S_{31}, and S_{41} are all valid candidates for connecting to S_{12}, which violates the condition that the interface matching must be one-to-one.

In this section, we extend our solution for single subnets with additional rules to account for cases when our algorithm finds multiple interfaces that are potential candidates for connecting to a single interface. The rules exploit properties of the spanning tree algorithm and enable us to narrow down the choice of interfaces that can be connected to a given interface. We must note, however, that the rules may not always be able to pinpoint the exact topology of a network (although our expectation is that such cases will be rare). In fact, we can show that there are cases for which it is *impossible* to uniquely determine the topology of the network, based solely on address forwarding information.

Consider the two distinct network topologies depicted in Fig. 6. Switches S_1 and S_4 belong to a single subnet, while switches S_2 and S_3 both belong to different subnets. Clearly, the address forwarding tables for switches in both topologies are identical, even though switch S_2 is connected to S_1 in Fig. 6(a) and S_3 is connected to S_1 in Fig. 6(b). Thus, *any* algorithm that relies only on address forwarding table information cannot distinguish between the two topologies. Since it may be impossible to infer a unique topology based on the given

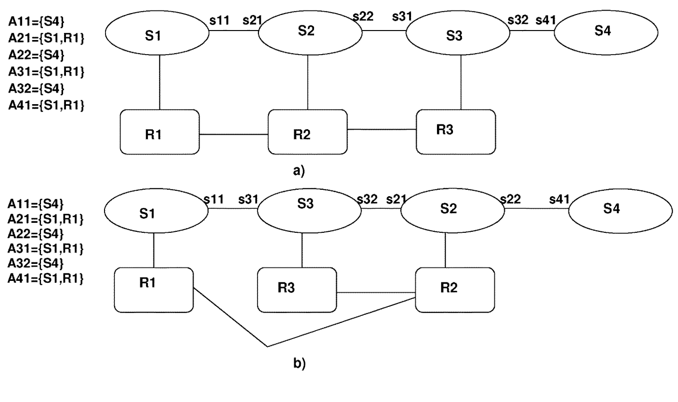

Fig. 6. Example indistinguishable network topologies.

information, we restrict ourselves to finding a minimal set of candidate topologies that contains the actual network topology.

A. Properties of Multi-Subnet Switched Domains

As mentioned earlier, the approach we adopt to discovering the topology of switched domains containing multiple subnets is to rule out interface connections that cannot exist. In the following lemmas, we identify the conditions under which two interfaces cannot be directly connected. The lemmas make use of the following property for switched domains containing multiple subnets: Suppose S_i and S_k are two nodes from different subnets; then, A_{ij} contains S_k if and only if there is a node S_p from the same subnet as S_k such that $S_p, \ldots, S_i, \ldots, S_k$ is a path in the spanning tree. Let U_{ijkl} denote the set of MAC addresses in the union $A_{ij} \cup A_{kl}$.

Lemma 4.1: Let S_{ij} and S_{kl} be different interfaces. If $A_{ij} \cap A_{kl} \neq \phi$, then interfaces S_{ij} and S_{kl} cannot be connected. ■

Proof: Suppose, to the contrary, that node S_p appears in both A_{ij} and A_{kl}, and interfaces S_{ij} and S_{kl} are connected. Then, there is a path from S_p to S_i via S_k and from S_p to S_k via S_i. Furthermore, each of these paths belongs to the spanning tree, which leads to a contradiction. Thus, if two interfaces have a nonempty intersection they cannot be connected. ■

Lemma 4.2: Let t be a subnet that contains at least two nodes S_p and S_q. If $A_{ij} \cap A_{kl} = \phi$ and U_{ijkl} contains either S_p or S_q but not both, then the interfaces S_{ij} and S_{kl} cannot be connected. ■

Proof: Suppose that S_{ij} and S_{kl} are connected. Without loss of generality, let $S_p \in A_{ij}$. Thus, there must be a path from S_p to S_i passing through S_k in the spanning tree. We consider two cases.

1) *The path from S_q to S_i in the spanning tree does not pass through S_k.* In this case, S_q will belong to A_{kl} since the path in the spanning tree from S_q to S_p will pass through S_i and S_k, and S_q and S_p belong to the same subnet t.
2) *The path from S_q to S_i in the spanning tree passes through S_k.* In this case, since S_p is in A_{ij}, there must be a node S_r such that $S_p, \ldots, S_k, S_i, \ldots, S_r$ is a path in the spanning tree and S_r also belongs to subnet t. Thus, it follows that $S_q, \ldots, S_k, S_i, \ldots, S_r$ will also be a path in the tree and S_q will belong to A_{ij} also.

Thus, we have shown above that *both* S_p and S_q must belong to U_{ijkl} if S_{ij} and S_{kl} are connected, and so the interfaces cannot be connected. ■

Lemma 4.3: Let $A_{ij} \cap A_{kl} = \phi$ and $A_{ij} \cap A_{pt} = \phi$. If $U_{ijkl} = U_{ijpt}$ and S_i and S_k belong to the same subnet which is different from that of S_p, then S_{ij} and S_{kl} cannot be connected. ■

Proof: Suppose S_{ij} and S_{kl} are connected. Note that $A_{kl} = A_{pt}$ since $A_{ij} \cap A_{kl} = \phi, A_{ij} \cap A_{pt} = \phi$ and $U_{ijkl} = U_{ij\,\mathrm{pt}}$. Also, since S_i and S_k are from the same subnet, $S_i \in A_{kl}$ and, thus, $S_i \in A_{pt}$. Thus, there must exist a node S_r belonging to the same subnet as S_i such that $S_i, S_k, \ldots, S_p, \ldots, S_r$ is a path in the spanning tree for the subnet. However, since S_i and S_k belong to the same subnet, this implies that $S_k \in A_{pt}$, which leads to $S_k \in A_{kl}$; this is clearly impossible, so we have a contradiction. ■

B. Topology Discovery Algorithm

Our topology discovery algorithm initially assumes that every candidate pair of interfaces is connected. It then applies the results of the lemmas presented in the previous section in order to eliminate pairs of interfaces that cannot be connected. Thus, finally, for every interface, we are left with a set of interfaces that the interface can be potentially connected to. This is output by our algorithm. Note that, if after excluding pairs of interfaces that cannot match, every interface matches only one other interface, then our algorithm computes the unique physical topology of the network.

From Lemmas 4.1, 4.2, and 4.3, it follows that for any pair of interfaces S_{ij} and S_{kl} to match, the following must hold.

1) $A_{ij} \cap A_{kl}$ is empty.
2) For every subnet t, either $A_{ij} \cup A_{kl}$ contains all nodes from subnet t or none of them.
3) If S_{ij} and S_{kl} belong to the same subnet, then there does not exist a node S_p from a different subnet such that $U_{ijkl} = U_{ij\,\mathrm{pt}}$ and $A_{ij} \cap A_{\mathrm{pt}} = \phi$.

For all such pairs of potentially matching interfaces S_{ij} and S_{kl} satisfying the above conditions, we refer to their AFT unions U_{ijkl} as *valid unions*. For a valid union U_{ijkl}, if S_{kl} does not occur in any other valid union, then we can conclude that S_{ij} is connected to S_{kl}. As a result, we can eliminate all other valid unions containing S_{ij}. This follows since the set of valid unions represents a superset of the actual connections in the network. Also note that, since between any pair of nodes there can be at most one active connection, once we have connected an interface of S_i with an interface of S_k, all other valid unions that could potentially create a loop in the underlying topology can be eliminated.

Thus, our topology discovery algorithm for matching interfaces is as follows.

1) Generate the initial set of valid unions U.
2) Repeat the following step until no further valid unions can be deleted from U.
2.1) If an interface S_{kl} occurs in only one valid union U_{ijkl} in U then create a connection between S_{ij} and S_{kl}, and (2.1.1) delete all valid unions containing S_{ij} from U (except for U_{ijkl}); and, (2.1.2) delete all valid unions

U_{ixpy}, where $(x,p,y) \neq (j,k,l)$ and S_p is any node that can reach node S_k using existing connections (including S_k itself).

2.2) If every interface S_{kl} occurs in more than two valid unions then select U_{ijkl}, such that the number of occurrences of S_{kl} in valid unions of U is minimal among all (node, interface) combinations. Otherwise, select an arbitrary U_{ijkl}. In either case, after the selection is fixed, repeat steps (2.1.1) and (2.1.2) above.

3) For every selected valid union U_{ijkl} in U, output "S_{ij} connected to S_{kl}".

The connections output by the topology discovery algorithm above are guaranteed to be a superset of the actual connections in the network. As we pointed out earlier, for certain networks (see Fig. 6) it is impossible to accurately compute the network topology. For such networks, our algorithm may not return a unique network topology; in other words, our algorithm may output multiple possible connections for an interface, only one of which is an actual connection (in the network). However, for most practical network topologies, we expect our algorithm to generate the precise topology information in which there is a one-to-one mapping between interface pairs. The question of what extra information (in addition to address forwarding information) is required to guarantee the discovery of a unique topology for arbitrary networks remains open. In the following example, we demonstrate that while the topology discovery algorithm for the single subnet case (Section III) cannot find the correct topology for the 2-subnet network in Fig. 5, our algorithm for multiple subnets will in fact identify the correct network topology.

Example 4.4: Consider the network depicted in Fig. 5. Switches S_1, S_4 and router R_1 belong to subnet 1 while switches S_2, S_3 and router R_2 belong to subnet 2. There is a single interface (S_{11}) that contains only R_1 and a single interface (S_{23}) that contains only R_2. Consequently, S_{11} is connected with r_1. Similarly, S_{23} is connected with R_2. The remaining address sets A_{ij} are as follows: $A_{12} = A_{32} = \{S_4\}, A_{21} = A_{41} = \{S_1, R_1\}, A_{22} = \{S_3, S_4\}$, and $A_{31} = \{S_1, S_2, R_1, R_2\}$.

The valid unions are: $U_{1221} = U_{3241} = \{S_1, S_4, R_1\}$, and $U_{2231} = \{S_1, S_2, S_3, S_4, R_1, R_2\}$. Note that $U_{1231} = \{S_1, S_2, S_4, R_1, R_2\}$ is not a valid union (due to Lemma 4.2) since it contains switch S_2 but not S_3 belonging to subnet 2. Furthermore, $U_{1241} = \{S_1, S_4, R_1\}$ is also eliminated (due to Lemma 4.3) since $U_{1241} = U_{1221}$ and switches S_1 and S_4 belong to the same subnet, while S_1 and S_2 belong to different subnets. Since every interface occurs only once in the above set of valid unions, S_{12} is connected with S_{21}, S_{22} is connected with S_{31}, and S_{32} is connected with S_{41}. ∎

C. Characterization of Identified Topologies

In this section, we characterize a broad class of switched domains for which the algorithm developed in the previous section is guaranteed to identify the *unique* physical topology. We refer to this class of switched domains as *ordered* networks (formally defined below). Note that, we view each switched domain as a tree of active forwarding paths as determined by the spanning tree protocol [20]. We define a set of addresses A to be *legal* if, for any subnet t, A contains either *all* or *none* of the addresses in t.

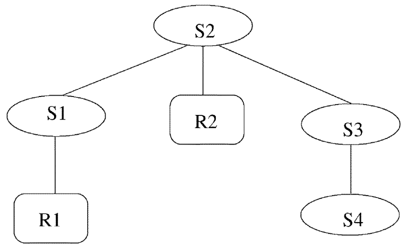

Fig. 7. Example ordered network.

Definition 4.5: A network is an *ordered network* if it can be arranged as a *rooted tree* that satisfies the following two properties: 1) For every subtree in the (rooted) network tree, for every subnet contained in it, there exists a node belonging to the subnet elsewhere in the network (not in the subtree); and 2) For any two subtrees rooted at nodes S_i and S_k in the (rooted) network tree, if the union of addresses in the two subtrees is legal, then: (a) the nodes S_i and S_k belong to the same subnet; and, (b) the parents of S_i and S_k in the tree belong to the same subnet (that is perhaps different from that containing S_i, S_k). ∎

Let us denote a connection between interfaces S_{ij} and S_{kl} such that S_i is a parent of S_k in the network tree by $\langle S_{ij}, S_{kl}\rangle$. We refer to a pair of subtrees as *legal subtrees* if the union of addresses in the subtrees is legal. The first property of ordered networks ensures that for a connection $\langle S_{ij}, S_{kl}\rangle$, the address table A_{ij} contains all the addresses in the subtree rooted at S_k. The second property, by requiring that roots and their parents for a pair of legal subtrees belong to the same subnet, guarantees that valid unions which do not correspond to matching connections are eliminated by our algorithm. Note that this requirement is not too restrictive, since most networks will most likely contain few pairs of legal subtrees. Furthermore, it is trivially satisfied in networks that do not contain pairs of legal subtrees or networks in which every subnet occurs in *more than two* distinct subtrees of the root.

The network depicted in Fig. 5 is an ordered network. To see this, consider the network arranged as a rooted tree with node S_2 as the root, as shown in Fig. 7. Note that for every subtree in the network tree, there is a node belonging to a subnet in the subtree elsewhere in the graph. For example, consider the subtree rooted at S_1. Node S_4 belongs to the same subnet as S_1 and is not contained in the subtree. Also, the network satisfies the second property of the ordered network definition. To see this, note that the subtrees rooted at nodes S_1 and S_4 constitute a pair of legal subtrees (since they contain all the addresses in subnet 1), and the nodes themselves as well as their parents (S_2 and S_3) belong to the same subnet.

Theorem 4.6: The topology discovery algorithm presented in Section IV-B identifies the accurate physical topology for ordered network graphs. ∎

Proof: For ordered networks, for any parent-child connection $\langle S_{ij}, S_{kl}\rangle$, it is the case that A_{ij} is the set of addresses that appear in the subtree rooted at S_k. Also, A_{kl} is the set of addresses belonging to subnets in S_k's subtree that are not con-

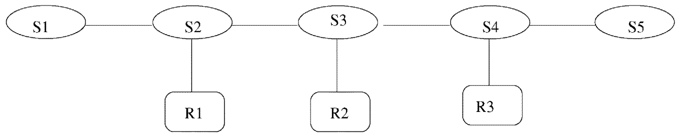

Fig. 8. Example nonordered network.

tained in A_{ij}. We refer to these addresses as the complement of A_{ij} and denote them by $\bar{A}_{ij}$. Note that $A_{ij} \cup \bar{A}_{ij}$ is legal. Thus, $A_{kl} = \bar{A}_{ij}$ and $A_{ij} = \bar{A}_{kl}$.

In an ordered network, for any distinct pair of parent-child node connections $\langle S_{ij}, S_{kl} \rangle$ and $\langle S_{pq}, S_{uv} \rangle, A_{ij} \neq A_{pq}$ and $A_{kl} \neq A_{uv}$. As a result, for the connection $\langle S_{ij}, S_{kl} \rangle$, there can exist at most one other connection $\langle S_{pq}, S_{uv} \rangle$ such that $A_{ij} = A_{uv}$ and $A_{kl} = A_{pq}$. In this case, the subtrees rooted at S_k and S_u constitute a pair of legal subtrees (since $A_{kl} = A_{pq} = \bar{A}_{uv}$). Furthermore, these connections can result in the following four valid unions that are all equal: $U_{ijkl}, U_{ijpq}, U_{pquv}$, and U_{kluv}. Of these, U_{ijpq} and U_{kluv} will be deleted since S_i and S_p belong to the same subnet, and S_k and S_u also belong to the same subnet (due to the second property of ordered networks). We still need to show, however, that the valid unions U_{ijkl} and U_{pquv} will not be deleted. For this, we need to show that S_i and S_k belong to different subnets (a similar argument can be used to show that S_p and S_u belong to different subnets). If S_i is in the same subnet as S_k, then S_i must belong to A_{kl}. However, since $A_{kl} = A_{pq}, S_i$ must be in the subtre rooted at S_u. This, however, would mean that S_k is also in the subtree rooted at S_u, and so $S_k \in A_{pq}$, which is impossible since $A_{kl} = A_{pq}$ and S_k cannot be in A_{kl}. ■

We must note that our topology discovery algorithm can determine the accurate topology for networks that may not be ordered. Fig. 8 depicts one such network. In the figure, S_1, S_3, S_5, and R_2 belong to subnet 1, S_2 and R_1 belong to subnet 2, and S_4 and R_3 belong to subnet 3. For every possible rooted network tree, one of subnets 2 or 3 will be entirely contained in a single subtree and so the network cannot be ordered. Our algorithm, however, will accurately discover the physical topology of the network. Thus, the class of ordered networks is actually a *proper subset* of the class of networks for which our algorithm identifies the unique physical topology. In other words, the "ordered network" condition is only *sufficient* for our algorithm to uniquely identify the layer-2 topology; the question of determining a characterization that is both necessary and sufficient for our techniques remains open.

V. Extensions

We now show how the algorithms that we presented in the previous subsections can be extended to handle incomplete address forwarding tables and Virtual LANs (VLANs).

A. Dealing With the Completeness Requirement

Thus far, we have assumed that each address forwarding table A_{ij} is complete, i.e., it consists of all MAC addresses reachable from S_i through the interface S_{ij}. In practice, however, this assumption is highly unlikely to hold. The reason for this is that, although the A_{ij}'s are learned based on the source addresses in frames received at the interface S_{ij}, these learned entries are aged (and removed) by the network elements. Therefore, unless an element constantly receives packets from a source address at intervals smaller than the aging interval (which is typically five minutes to five hours), the element may delete the entry corresponding to that source address. Thus, the A_{ij} address sets may not be complete.

We present two complementary sets of techniques to deal with the above problem. Our first set of techniques attempts to keep the A_{ij}'s as complete as possible either by creating extra traffic across nodes in a switched domain or by continuously collecting the relevant AFT information. Our second set of techniques employs approximations and heuristics to handle minor deviations from completeness. These two solutions together ensure that our algorithms work in practice as borne out by our experiments with the Bell Labs research network.

Our first solution tries to ensure that the A_{ij}'s are as complete as possible by generating extra traffic between pairs of nodes in the switched domain and, consequently, not allowing the address forwarding table entries to age. The mechanism we use to generate traffic from node X to node Y is to generate an ICMP (Echo Request) message from a network management station to X with the source address in the ICMP packet set to the IP address of Y. This essentially involves creating "spoofed" ICMP `ping` packets that look like they originate from node Y, while they actually originate from the management station. This will cause X to respond to the Echo Request to node Y. There are potential problems with this approach. First, when switches in the network are connected through "out-of-band" interfaces, the ICMP messages sent between such switches would not populate the address forwarding tables of "in-band" interfaces, as required by our algorithm. To deal with such scenarios, our implementation relies on sending ICMP messages between *hosts* and switches in the same subnet to ensure that the "in-band" interfaces are used. Second, for security concerns, network administrators often explicitly disable the ICMP `ping` command on some nodes, which means that our spoofed `pings` will not help in populating the switch address forwarding tables. Finally, even in the case that `ping` commands are enabled, the number of required messages raises a potential scalability concern as the size of the underlying switched domain grows. To address this concern, our implementation restricts `ping` message exchanges only to nodes belonging in the same subnet (which is typically not that large); furthermore, in the case of large subnets, we also subdivide the subnet into smaller segments and perform the AFT population in parallel for these segments.

A different technique used in ***NetInventory*** in an effort to obtain address-forwarding information that is as complete as possible is to query element AFTs continuously (i.e., at regular time intervals). Our data-collection program copies the address forwarding tables of the switches at regular intervals and does not age out the entries regularly like switches do. Experience shows that after one normal business day, the address forwarding tables are usually almost complete. Note that, in the (rare) event of a change in the spanning-tree topology for the underlying switched domain (e.g., due to a failure), this approach may result in false-positive connections, since some switch addresses could be legitimately dropped from element AFTs. To eliminate such false positives, ***NetInventory*** also employs a slow timeout mechanism for collected address-forwarding entries.

Our second set of techniques extends the valid-union algorithm described in Section IV-B to handle the cases in which the address forwarding tables are incomplete. More specifically, we modify Condition (2) in the valid-union definition as follows.

- For every subnet t, the union $A_{ij} \cup A_{kl}$ contains either no nodes from t, or all nodes from t that appear in the union of the address forwarding tables for *all* interfaces of S_i, S_j.

The above modification is guaranteed not to eliminate true connections, while it may also produce some false positives. Our experience with the ***NetInventory*** system shows that employing a combination of the above-described techniques guarantees the reliable and accurate discovery of layer-2 topology information.

B. Handling VLANs

VLANs define multiple spanning trees within a switched domain. A switch may belong to multiple VLANs and effectively maintains address forwarding tables for each VLAN that it is a part of. Frames belonging to a specific VLAN are then forwarded by a switch using the forwarding tables for the VLAN. If we have access to the address forwarding tables for each VLAN, then we can run our algorithms for each VLAN individually in order to generate the spanning tree for the VLAN. We only need to be careful to restrict ourselves to the universe of addresses comprising only MAC addresses within the VLAN.

The major difficulty in handling VLANs within the ***NetInventory*** topology discovery algorithms is discovering where the required VLAN information is stored in the element MIBs. Standard SNMP MIBs do not provide information on address forwarding tables for individual VLANs, but our experience shows that this information can be collected using proprietary MIBs (for example, the Prominet MIB for Cajun Switches). Thus, ***NetInventory*** needs to employ wrappers that are custom-designed for specific types of switches in order to extract VLAN information from the proprietary portions of their MIBs. The extensions necessary for dealing with VLAN topology discovery are currently being incorporated in the ***NetInventory*** architecture.

Our next example demonstrates that, even in the presence multiple subnets and VLANs in a switched domain, our topology discovery algorithm (Section IV-B) can still identify the correct topology.

Example 5.1: Consider the network depicted in Fig. 9. Switches S_1, S_4, and router R_1 belong to subnet 1; switches S_2, S_3, and router R_2 belong to subnet 2; switches S_5, S_6, and router R_3 belong to subnet 3. In addition, there are three VLANS, one for each subnet. The first VLAN consists of the path R_1, S_1, S_2, S_4, the second consists of the tree involving router R_2 and switches S_1, S_2 and S_3, and the third consists of the path $R_3, S_6, S_3, S_4, S_2, S_5$. The address forwarding tables for the interfaces without taking into account VLAN information are shown in Fig. 9.

There are single interfaces that contain only R_1, or R_2, or R_3. Consequently, r_1, r_2 and r_3 are matched respectively with S_{11}, S_{12}, and S_{62} and these interfaces are eliminated from further consideration. The set of valid unions

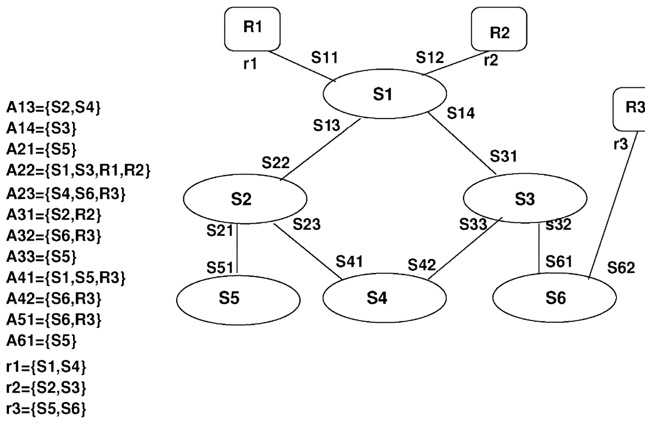

Fig. 9. Network containing VLANs and multiple subnets.

is as follows: $U_{1322} = \{S_1, S_2, S_3, S_4, R_1, R_2\}, U_{1431} = \{S_2, S_3, R_2\}, U_{2142} = U_{2151} = U_{3261} = U_{3342} = U_{3351} = U_{4261} = \{S_5, S_6, R_3\}$, and $U_{2341} = \{S_1, S_4, S_5, S_6, R_1, R_3\}$, The valid unions U_{1322}, U_{1431}, and U_{2341} all contain interfaces that appear only once in the set of unions. Consequently, S_{13}, S_{14}, and S_{23} are matched with S_{22}, S_{31}, and S_{41}, respectively. Thus, union U_{2142} is eliminated since S_{23} is already matched with S_{41}. Deletion of U_{2142} causes U_{2151} to be selected (since interface S_{21} appears only once). Thus, in the next iteration, U_{3351} is deleted. In the final iteration, among the remaining unions, since interfaces S_{32} and S_{33} occur only once, U_{3261} and U_{3342} are retained, while U_{4261} is eliminated. Thus, the final set of valid unions yields the actual topology of the network. ■

VI. Implementation

We have implemented the topology discovery algorithms presented in this paper in the context of the ***NetInventory*** system, which is designed as part of a network management tool suite. In this section, we describe the ***NetInventory*** conceptual and functional system architecture. The key features of ***NetInventory*** are as follows:

- automatic discovery of currently active network elements for a specific network or a given network segment;
- automatic discovery of (active) network interconnections at layer-3, layer-2, or layer-1 (i.e., hubs) of the OSI protocol hierarchy;[3]
- storing current as well as historic network information in the network management database, and using database techniques to access the topology information in an efficient manner;
- ensuring automatic database updates whenever the network goes through either management- or network-induced changes;
- easy system extensibility.

NetInventory automatically discovers and maintains layer-1, layer-2, and layer-3 network information, including (but not limited to) network topology for heterogeneous networks with multiple sets of subnets, and maintains a database consisting of

[3]The details of the layer-1 algorithms of ***NetInventory*** can be found in [1].

information about network elements and their interconnections by periodically updating its topology information as the network goes through changes. The frequency of data collection and the set of network addresses that the information should be collected from are specified by the user as configuration settings. Information in the database is timestamped to enable historical views of the network and report the network changes that have occurred in a given time frame. The ***NetInventory*** database also contains information about network elements at layer-1 (e.g., hubs) that do not have any IP address assigned to them. ***NetInventory*** comes with a graphical user interface and can be easily integrated with existing network management tools. The current version of ***NetInventory*** assumes that the topology of the underlying network remains stable during the discovery process; thus, it cannot effectively deal with node or link failures that may occur during network discovery.

A. NetInventory Conceptual System Architecture

In this section, we describe the ***NetInventory*** conceptual system architecture. ***NetInventory*** assumes that each node in the network has one of four basic types: *router, switch, hub*, and *host*. We assume that each switched domain is a tree and the network may include several switched domains, so that the entire network is not necessarily a tree. The ***NetInventory*** conceptual system architecture is depicted in Fig. 10.

The three basic components of ***NetInventory*** are *User Interface (GUI), Resource Discovery,* and *Topology Discovery*. The ***NetInventory*** database contains information about each network node and its interfaces that were active during the discovery process. It also contains information on interconnections among nodes. Database information on active network nodes is collected by the ***NetInventory*** Resource Discovery process. The process gathers the information in a consistent manner from network-element MIBs using SNMP commands and from other sources such as *domain name servers (DNS)*, *administrative files* maintained by network managers, and *system files*. Along with the collected information, the resource discovery process stores the time at which data collection was performed and the data source from which the information was extracted.

The data collection process accesses information from each node's MIBs. It is a well known fact that access to large MIB tables may significantly degrade node performance. This is especially noticeable for router nodes that have large routing and address conversion tables and for switch nodes that have large address forwarding tables. In order to reduce the performance impact of ***NetInventory*** on a network, each MIB entry returned by the nodes is processed immediately and used to determine what other MIB entries are needed. For example, the validity (`dot1dTpFdbStatus`) of an entry in the address forwarding table needs to be checked only if the corresponding MAC address (`dot1dTpFdbAddress`) belongs to the switched domains whose topology is being computed, and the corresponding port (`dot1dTpFdbPort`) is up. By using this kind of optimizations, the data traffic, and hence the CPU load on the routers and switches, required to collect topology information is reduced by almost 60% compared to standard, publically available SNMP utility packages that download the entire MIB before processing it.

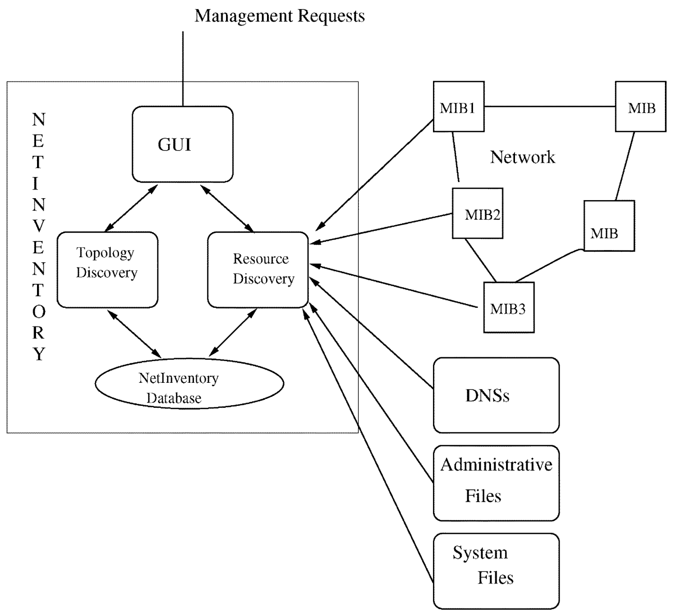

Fig. 10. ***NetInventory*** conceptual system architecture.

The topology discovery process generates the network map for a user-specified set of nodes. Requests are initiated by the network manager using the ***NetInventory*** user interface. The discovered map is also stored in the database along with the timestamp of map generation. Each map is stored as a set of links with each link indicating the two network elements that it connects and the ports through which they are connected. All the ***NetInventory*** code except for the GUI is written in platform-independent C++. We are also currently working on a Java version that will make our GUI implementation platform-independent.

B. NetInventory Functional System Architecture

The ***NetInventory*** system consists of five primary modules.

1) **Inventory**. This module generates and periodically updates the ***NetInventory*** database. There are two modes of operation:

   a) Inventory Creation: The network operator creates "profiles" in which a list of IP addresses and masks for the subnets being monitored are specified. For single nodes to be included, the mask `255.255.255.255` is used. All the inventory information is collected on the nodes, not just the portion pertaining to the specified subnets. For example, if a router is attached to eight subnets (hence, having eight IP addresses) but only one of the IP addresses is specified in the profile, the information regarding the other seven interfaces will still be collected. After the initial inventory creation, the operator may examine the inventory information and decide to add other subnets to the profile.

b) Inventory Maintenance: The information on nodes discovered in the inventory creation process will be refreshed periodically, with a refresh interval specified by the network operator. When the same node is specified in multiple profiles, its information will be refreshed according to the shortest interval only. For example, the operator may specify the entire subnet in one profile that is refreshed every hour, while using another profile to monitor the gateway router and the web server at five-minute intervals.

In addition to specifying a data-refresh interval, the pace of communication within each data collection cycle can also be configured by the operator at a level that does not impose significant load on the network. We use our own *pipelined* version of `ping` for node discovery; that is, the program does not wait for the response from the first IP address before sending out the next ICMP `ping` packet. The speed of our `ping` is only limited by the speed of the network interface at the management station. Therefore, to avoid flooding the network, we insert artificial delays between packets. Theoretically, we could ping more than 1.3 million nodes per second on a 100-Mb/s Ethernet connection but, in practice, we set the program to ping 160 nodes per second, in 16-node bursts, 100 ms apart.

2) **Topology**. This module computes the topology of the specified profile. It works in three phases:

a) Address Forwarding Table Population: Use the "spoofed" `ping`-packet techniques described in Section V-A to ensure that element AFTs are reasonably complete.

b) Data Collection: Collect information pertaining to the topology of the network as specified in the profiles. If an address forwarding table is incomplete, entries from the previous run may also be used.

c) Topology Generation: The topology of the specified profile is computed and stored in the database.

3) **Device Vendor Plug-in**. Not all vendors design their network elements to completely conform to all standards. For example, the Lucent Cajun switches use a different format to store the address forwarding table due to VLAN considerations. ***NetInventory*** allows special data collection actions to be taken based on the value of the `sysObjectID` MIB variable. This applies to both the Inventory and the Topology portions.

4) **Interface Type Plug-in**. ***NetInventory*** started with handling only Ethernet interfaces. However, we quickly recognized the need to compute the topology of network elements connected using other technologies, such as Fast Ethernet, ATM, FDDI, SONET, and Frame Relay interfaces. Our current ***NetInventory*** implementation supports all these different connection technologies.

5) **Protocol Plug-in**. Currently, ***NetInventory*** uses primarily SNMP to collect MIB information. However, the system is implemented in such a way that the node information can also be collected through other means, such as a `telnet` command-line interface, which is used extensively in the Telecom world.

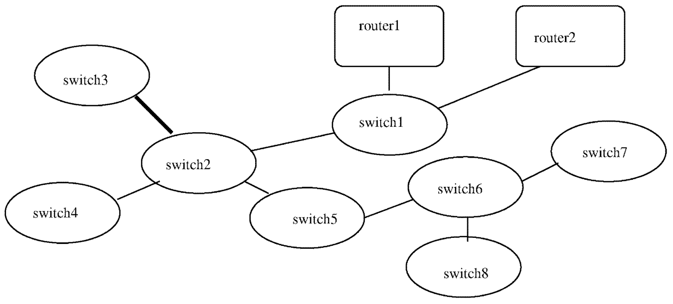

Fig. 11. Two-subnet network.

VII. Experimental Results

We have conducted several experiments using parts of Lucent's own research network. The main purpose of these experiments was twofold. First, we wanted to test the accuracy and correctness of our topology discovery tools in a real-life networking installation. Second, we wanted to verify the practicality of our tools by obtaining measurements for the running times of our algorithm for networks with multiple network elements that are distributed over several subnets. For the experiments presented in this section, we were mostly concerned with network elements that are either routers or switches.

Our results verified the robustness of our methodology for multi-subnet administrative domains, even in the presence of element interfaces with incomplete address sets. The topology maps generated by our tool were compared against the maps that were *manually* maintained by local network administrators. For all administrative domains tested, our tool generated the correct physical topology map. In fact, there were several cases in which our tool discovered element connections that were not present the network administrators' maps. In all such cases, the new interconnections discovered by our tool were indeed proven to be correct by a thorough check of the actual network topology. (The maps maintained manually by our network administrators did not include hosts, which made the task of comparing them against the output of ***NetInventory*** tedious, but manageable.) Figs. 11 and 12 depict the topology maps of two multi-subnet networks discovered by our ***NetInventory*** system. (Since these maps depict parts of Lucent's proprietary research network, we are using generic names for the network elements.)

In Fig. 11, switches switch_1 and switch_3 as well as router router_2 belong to the same subnet, while all remaining switches and router_1 belong to a different subnet. For our test runs, we found that the address sets that our algorithm collected for interfaces on switches $\mathrm{switch}_2, \mathrm{switch}_7$, and switch_8 were not complete. Nevertheless, ***NetInventory*** was able to discover the correct network topology using the supplemental approximation techniques described in Section V-A. We should also note that the connection between switches switch_2 and switch_3 was in fact missing from the network administrators' manual map.

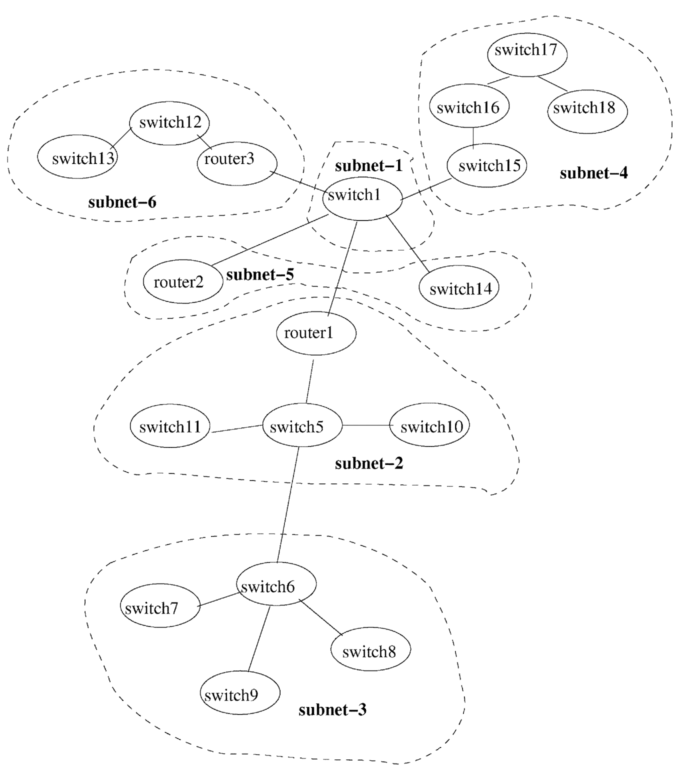

Fig. 12. Multi-subnet network.

TABLE I
NETINVENTORY PERFORMANCE RESULTS

No. of Subnets	No. of Routers/ Switches	No. of Nodes	Ping Time	Inventory Time	AFT Time	Topology Time
5	5	38	8.0	4.2	1	<1
8	12	74	5.9	23	4	<1
10	14	89	7.3	29	3	<1
12	23	241	7.7	40	9	<1
14	26	322	12.1	69	9	<1
20	32	522	18.6	229	19	<1
22	35	566	31.0	156	21	<1
28	40	672	7.5	176	22	<1
30	41	677	20.9	175	27	<1
32	43	691	24.6	186	25	1
36	44	717	31.4	349	26	1
40	49	736	34.5	207	28	<1
44	84	779	32.3	341	286	1
48	90	915	39.4	379	202	1
52	92	922	47.2	376	266	1
56	94	928	54.3	348	260	1
64	113	1221	51.2	492	259	2
72	243	1363	66.1	961	1437	5

Fig. 12 depicts the physical topology map discovered by ***NetInventory*** for a network with six distinct subnets composed as shown in the figure.

A second goal of our experimental study was to verify the practicality of our topology discovery algorithm, by measuring its running-time requirements for various network sizes. Table I shows the running time (in seconds) of different ***NetInventory*** components at various network sizes. (The "AFT Time" column in Table I indicates the time spent in order to populate switch AFTs.) For this experiment, our ***NetInventory*** implementation was compiled with Borland C++ Builder 5.0 and was run on a Pentium II 400-MHz computer with 288 MB of RAM and Windows NT 4.0 operating system. The maximum size of the working set during the entire experiment was 11 620 kB.

In general, we have discovered that our algorithm is sufficiently fast for all practical purposes. As can be seen in our numbers, most of the time in ***NetInventory*** is spent on data collection. Even in fairly large network configurations with more than 200 switches and 1000 nodes, running our topology algorithm took only five seconds.

VIII. CONCLUSION

Automatic discovery of physical topology information plays a crucial role in enhancing the manageability of modern IP networks. Despite the importance of the problem, earlier research and commercial network management tools have typically concentrated on either: 1) discovering logical (i.e., layer-3) topology, which implies that the connectivity of all layer-2 elements (e.g., switches and bridges) is ignored; or 2) proprietary solutions targeting specific product families. In this paper, we have developed novel, practical algorithms for discovering physical topology in heterogeneous IP networks. The practicality of our solution stems from the fact that it relies solely on local address forwarding information routinely collected in the SNMP MIBs of routers and switches. The main algorithmic challenge we have addressed is how to cleverly "stitch" that information together into a complete layer-2 LAN topology. Our algorithms can handle switched domains comprising one or more subnets and can be extended to deal with incomplete information and VLANs. We have implemented our algorithms in the ***NetInventory*** topology-discovery tool, which has been experimentally tested over Lucent's research network. The results clearly validate our methodology, demonstrating the accuracy and practicality of the proposed algorithms.

REFERENCES

[1] Y. Bejerano, Y. Breitbart, M. Garofalakis, and R. Rastogi, "Physical topology discovery for large multi-subnet networks," in *Proc. IEEE INFOCOM*, 2003, pp. 342–352.
[2] A. Bierman and K. Jones, "Physical topology MIB," IETF, Internet RFC-2922, Sept. 2000.
[3] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology discovery in heterogeneous IP networks," in *Proc. IEEE INFOCOM*, 2000, pp. 265–274.
[4] H. Burch and B. Cheswick, "Mapping the Internet," *IEEE Computer*, vol. 32, pp. 97–98, Apr. 1999.
[5] K. Calvert, M. B. Doar, and E. Zegura, "Modeling Internet topology," *IEEE Commun. Mag.*, vol. 35, pp. 160–163, June 1997.
[6] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," IETF, Internet RFC-1157, May 1990.
[7] P. E. Crandall and M. J. Quinn, "A partitioning advisory system for network data-parallel processing," *Concurrency: Practice and Experience*, vol. 7, no. 5, pp. 479–495, 1995.
[8] N. Dawes, D. Schenkel, and M. Slavitch, "Method of determining the topology of a network of objects," U.S. Patent 6,411,997, June 25, 2002.
[9] E. Decker, P. Langille, A. Rijsinghani, and K. McCloghrie, Definitions of managed objects for bridges, IETF, Internet RFC 1493, July 1993.
[10] N. G. Duffield, J. Horowitz, and F. Lo Presti, "Adaptive multicast topology inference," in *Proc. IEEE INFOCOM*, 2001, pp. 1636–1645.
[11] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On power-law relationships of the Internet topology," in *Proc. ACM SIGCOMM*, 1999, pp. 251–262.
[12] S. Figueira and F. Berman, "Modeling the effects of contention on the performance of heterogeneous applications," *Proc. IEEE Int. Symp. High-Performance Distributed Computing*, pp. 392–401, 1996.
[13] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proc. IEEE INFOCOM*, 2000, pp. 1371–1380.

[14] I. Katzela and M. Schwarz, "Schemes for fault identification in communication networks," *IEEE/ACM Trans. Networking*, vol. 3, pp. 753–764, Dec. 1995.
[15] MAS. Keshav, *An Engineering Approach to Computer Networking*. Reading: Addison-Wesley, 1997.
[16] J. Lee and G. de Veciana, "Resource and topology discovery for IP multicast using a fan-out decrement mechanism," in *Proc. IEEE INFOCOM*, 2001, pp. 1627–1635.
[17] P. K. K. Loh, W. J. Hsu, C. Wentong, and N. Sriskanthan, "How network topology affects dynamic load balancing," *IEEE Parallel and Distributed Technology*, vol. 4, pp. 25–35, Fall 1996.
[18] B. Lowekamp, D. R. O'Hallaron, and T. R. Gross, "Topology discovery for large Ethernet networks," in *Proc. ACM SIGCOMM*, 2001, pp. 237–248.
[19] K. McCloghrie and M. Rose, Management information base for network management of TCP/IP-based Internets: MIB-II, IETF, Internet RFC 1213, Mar. 1991.
[20] R. Perlman, *Interconnections: Bridges, Routers, Switches and Internetworking Protocols*. Reading, MA: Addison-Wesley, 1999.
[21] D. Schenkel, M. Slavitch, and N. Dawes, "Method of determining topology of a network of objects which compares the similarity of the traffic sequences/volumes of a pair of devices," U.S. Patent 5,926,462, July 20, 1999.
[22] G. Shao, F. Berman, and R. Wolski, "Using effective network views to promote distributed application performance," in *Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications*, 1999, pp. 2649–2656.
[23] R. Siamwalla, R. Sharma, and S. Keshav. (1998, July) Discovering Internet topology. [Online]. Available: http://www.cs.cornell.edu/skeshav/papers.html
[24] W. Stallings, *SNMP, SNMPv2, SNMPv3, and RMON 1 and 2*. Reading, MA: Addison-Wesley Longman, 1999.
[25] A. S. Tanenbaum, *Computer Networks*. Englewood Cliffs, NJ: Prentice-Hall PTR, 1996.
[26] K. Tutschku and H. Baier, "Characterizing network performance for enterprise networks," *Proc. Passive and Active Measurements (PAM) Workshop*, 2001.
[27] B. M. Waxman, "Routing of multipoint connections," *IEEE J. Select. Areas Commun.*, vol. 6, pp. 1617–1622, Dec. 1988.
[28] D. Wetherall, N. Spring, and R. Mahajan, "Measuring ISP topologies with Rocketfuel," in *Proc. ACM SIGCOMM*, 2002, pp. 133–146.
[29] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, "High speed and robust event correlation," *IEEE Commun. Mag.*, vol. 34, pp. 82–90, May 1996.
[30] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for Internet topology," *IEEE/ACM Trans. Networking*, vol. 5, pp. 770–783, Dec. 1997.

Yuri Breitbart received the D.Sc. degree in computer science from the Department of Computer Science at Israel Institute of Technology (Technion), Haifa, Israel.

He is the Ohio Board of Regents Distinguished Professor of Computer Science in the Department of Computer Science at Kent State University. He joined Kent State in January 2002 after spending several years at Bell Labs. From 1986 to 1996, he was a Professor in the Department of Computer Science, University of Kentucky, Lexington, and the Department Chair for the first seven years. Prior to 1986, he was leading the Database Research group first at ITT Research Center (1979–1981) and then at Amoco Production Company Research Center (1981–1986). He has held visiting positions at the Swiss Technological Institute (ETH) in Zurich and at HP Research Center in Palo Alto. He has been consulting for numerous companies, including IBM, Boeing, Amoco, HP, and Bell Labs. His research is in the area of distributed information system, network management systems and bio-informatics.

Dr. Breitbart is a Fellow of the Association for Computing Machinery (ACM) and a member of the IEEE Computer Society and ACM SIGMOD.

Minos Garofalakis (S'94–A'98–M'03) received the M.Sc. and Ph.D. degrees in computer science from the University of Wisconsin-Madison in 1994 and 1998, respectively.

He is a Member of Technical Staff with the Internet Management Research Department of Bell Labs, Lucent Technologies, Murray Hill, NJ. He joined Bell Labs in September 1998. His research interests lie in the areas of data streaming, approximate query processing, data mining, network management, and XML databases.

Dr. Garofalakis is a member of the Association for Computing Machinery (ACM). He has served on the program committees of several conferences in the database area, including ACM SIGMOD, VLDB, ACM SIGKDD, and IEEE ICDE.

Ben Jai received the Ph.D. in computer science from New York University, New York, in 1999, after developing software for 15 years.

He then worked on network research at Bell Labs, Lucent Technologies, Murray Hill, NJ, for four years, and joined Google, Mountain View, CA, in March 2003. He is currently working on the next-generation high-performance distributed computing infrastructure for Google.

Cliff Martin is a Distinguished Member of Technical Staff in the Computer Sciences Research Center of Bell Labs, Lucent Technologies, Murray Hill, NJ.

Rajeev Rastogi received the B.Tech. degree in computer science from the Indian Institute of Technology, Bombay, in 1988, and the M.Sc. and Ph.D. degrees in computer science from the University of Texas, Austin, in 1990 and 1993, respectively.

He is the Director of the Internet Management Research Department at Bell Labs, Lucent Technologies, Murray Hill, NJ. He joined Bell Labs in 1993 and became a Bell Labs Fellow in 2003. His research interests include database systems, network management, and knowledge discovery.

Avi Silberschatz (SM'93–F'00) is a Professor of Computer Science at Yale University, New Haven, CT. Prior to joining Yale, he was the Vice President of the Information Sciences Research Center at Bell Labs., Murray Hill, NJ. Prior to that, he held a chaired professorship in the Department of Computer Sciences, University of Texas at Austin. His research interests include operating systems, database systems, real-time systems, storage systems, network management, and distributed systems. He served as a member of the Biodiversity and Ecosystems Panel on President Clinton's Committee of Advisors on Science and Technology, as an advisor for the National Science Foundation, and as a consultant for several private industry companies. His writings have appeared in numerous ACM and IEEE publications and other professional conferences and journals. He is a coauthor of two well-known textbooks, *Operating System Concepts* (6th ed., New York: Wiley, 2002) and *Database System Concepts* (4th ed., New York: McGraw-Hill, 2001).

Prof. Silberschatz is a Fellow of the Association for Computing Machinery (ACM). He received the 2002 IEEE Taylor L. Booth Education Award, the 1998 ACM Karl V. Karlstrom Outstanding Educator Award, the 1997 ACM SIGMOD Contribution Award, and the IEEE Computer Society Outstanding Paper Award for the article "Capability Manager," which appeared in the IEEE Transactions on Software Engineering.