Topology Discovery in Heterogeneous IP Networks

Yuri Breitbart, Minos Garofalakis, Cliff Martin, Rajeev Rastogi, S. Seshadri, Avi Silberschatz
Information Sciences Research Center
Bell Labs, Lucent Technologies
600 Mountain Avenue
Murray Hill, NJ 07974
{yuri,minos,cliff,rastogi,seshadri,avi}@bell-labs.com

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 (a) 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 (b) 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 a 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 in time that is roughly quadratic in the number of network elements.**

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 will 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 [1], [2]. Furthermore, a full physical map of the network enables a proactive analysis of the impact of link and device failures. Early identification of single points of failure that could disrupt a large fraction of the user community 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) feature an IP mapping functionality for automatically discovering routers and subnets and generating a *network layer* (i.e., ISO *layer-3*) topology showing the router-to-router interconnections and router interface-to-subnet relationships. Building a layer-3 topology is relatively easy because routers must be explicitly aware of their neighbors in order to perform their basic function. Therefore, standard routing information is adequate to capture and represent layer-3 connectivity. 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) that comprise each subnet. 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 at each element and the diversity of elements and protocols present in today's multi-vendor IP networks. The combination of these 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 algo-

rithm 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 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.

A. Related Work

SNMP-based algorithms for automatically discovering network layer (i.e., layer-3) topology are featured in many common network management tools, such as HP's OpenView (`www.openview.hp.com`) and IBM's Tivoli (`www.tivoli.com`). Other commercially available tools for discovering layer-3 network topology using SNMP include Actualit's Optimal Surveyor (`www.actualit.com`) and the Dartmouth Intermapper (`intermapper.dartmouth.edu`). In recent work, Siamwalla et al. [3] propose heuristics for inferring layer-3 topology that employ only basic IP primitives (e.g., `ping` and `traceroute`).

Recognizing the importance of layer-2 topology, a number of vendors have recently developed proprietary tools and protocols for discovering physical network connectivity. 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 MIBs and are not useful on a heterogeneous network comprising elements from multiple vendors.

The only tool available on the market that claims to automatically discover physical topology in heterogeneous networks is Loran Technologies' Kinnetics network manager (`www.loran.com`). The details of their discovery algorithm are proprietary and not available to us at the time of this writing.

B. Our Contributions

In this paper, we develop novel, practical algorithmic solutions for the problem of discovering physical topology in heterogeneous IP networks. The practicality of our algorithms stems from the fact that they rely solely on standard information routinely collected in the SNMP *Management Information Bases* (MIBs) [4], [5] of elements and they require no modifications to the operating system software running on elements or hosts. More specifically, our topology discovery tool only utilizes information from the *address forwarding tables* of elements capturing the set of Medium Access Control (MAC – i.e., layer-2) addresses that are reachable from each element interface. 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 (global) physical network 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 [6] 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 global network topology works perfectly when (a) each *switched domain* (i.e., collection of switched subnets connected to the "outside world" through one or more layer-3 routers) consists of a *single* switched subnet, and (b) 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 invisible 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 two distinct techniques to handle this problem. Our first technique is based on generating extra network traffic across switches (using the IP `ping` mechanism) to ensure that the address forwarding tables are adequately populated. Our second method extends our topology discovery algorithm so that interconnection decisions are made based on incomplete information by employing some reasonable approximations. 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 the two 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.)

We have implemented and run all our topology discovery algorithms on Lucent's research network at Murray Hill. Preliminary results look very encouraging and we are in the process of conducting more extensive experiments.

C. 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. Finally, Section VII 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 the *administrative domain*. We model the administrative domain communication network as an undirected graph N. The nodes in the network correspond to network elements that can be one of two types: *routers* and *switches*[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. Figure 1 shows the graph corresponding to an example administrative domain. In Figure 1, R1, R2, and R3 are routers, while S1 through S5 are switches forming two distinct switched domains ({ S1, S2, S3 } and { S4, S5 }). We define a *subnet* as a maximal set of IP addresses such that any two machines 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 Figure 1, the switched domain $\{$ S4, S5 $\}$ contains only one subnet while the switched domain $\{$ S1, S2, S3 $\}$ consists of two subnets, one containing S1 and S3, and 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.

Switches within a switched domain typically employ the *spanning tree protocol* to determine unique forwarding paths for each switch [7]. Our topology discovery algorithm is based on the MAC addresses learned using backward learning [7] on interfaces that are part of the switched domain spanning tree.

[1]The algorithms in this paper can be used to discover topology of hosts and other network elements such as hubs. However, for the sake of simplicity of exposition, we do not consider hosts and hubs.

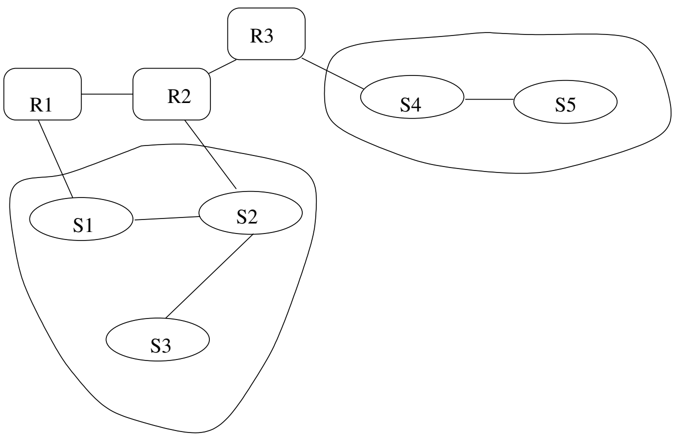

Fig. 1. Network graph for a typical administrative domain.

Therefore, it follows that we will not discover edges between interfaces that are not active (i.e., are eliminated by the spanning tree protocol). 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 j^{th} 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 *address forwarding table* 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 frames received at S_{ij}. In the remainder of the paper, the address set A_{ij} is restricted to MAC addresses of switches and routers only. We say A_{ij} is *complete* if A_{ij} contains the MAC addresses of *all* switches and routers 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 Figure 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 forwards it to R1 and, finally, R1 forwards the frame to S3 with the source MAC address being that of R1 (even though the frame will pass through S2).

III. Single Subnet Switched Domains

In this section, we describe a topology discovery algorithm under the following assumptions: (i) each switched domain contains exactly one subnet, (ii) no VLANS are present in the administrative domain, and (iii) the address forwarding tables are complete. We first briefly describe how we discover the set of switches and routers in the administrative domain which form the nodes of N. We then describe our algorithm for discovering the 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 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 [6] in R. Figure 2 outlines our algorithm for discovering the set of routers in the administrative domain.

```
Procedure FindRouters(R1)
/* R1 is the IP address of some known */
/* router in the administrative domain */
begin
routerSet := {R1}
routersVisited := φ
while routerSet ≠ φ do {
    choose a router R from routerSet
    routerSet := routerSet - {R}
    if {R} ∈ routersVisited
        continue
    routersVisited := routersVisited ⋃ {R}
    NH(R) := next hops for R for some destination
    routerSet := routerSet ⋃ NH(R)
}
end
```

Fig. 2. Finding 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 obtaining the IP address of an interface of R using the `ipAddrTable` in MIB-II. D is then computed by enumerating the set of IP addresses in the subnet corresponding to the IP address of an interface. This enumeration will take into account the subnet masks and the IP address formats. Once D is computed, for each IP address in D, we determine whether it is a switch by checking for the presence of the Bridge MIB [8]. Actually, both routers and switches contain the Bridge MIB and, therefore, we use the value of the `ipForwarding` variable to determine if an IP address belongs to a switch or a router. If `ipForwarding` is equal to 1, then the element in question is a router, otherwise it is a switch.

At this juncture, we have discovered the set of routers and switches in the administrative domain, i.e., the nodes of N. We next describe how to discover the interconnections between switches and routers.

A. Discovering the edges in N

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

Lemma III.1: Interfaces S_{ij} and S_{kl} are connected to each other if and only if $A_{ij} \bigcup A_{kl} = \mathcal{U}$ and $A_{ij} \bigcap A_{kl} = \phi$.

Proof: If S_{ij} and S_{kl} are connected to each other clearly, $A_{ij} \bigcap A_{kl} = \phi$. Further, since the A_{ij}'s are complete, $A_{ij} \bigcup A_{kl} = \mathcal{U}$.

To prove the other direction, assume $A_{ij} \bigcup A_{kl} = \mathcal{U}$ and $A_{ij} \bigcap A_{kl} = \phi$. Let if possible, S_{ij} and S_{kl} not be connected to each other. Let P be the path from S_i to S_j in the spanning tree. Recall that we assume all the A_{ij}s are complete. There are three 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 can not be the case that $A_{ij} \bigcap A_{kl} = \phi$.
2. P contains exactly one of S_{ij} or S_{kl}: In this case, once again it can not be the case that $A_{ij} \bigcap A_{kl} = \phi$.
3. P contains neither S_{ij} nor S_{kl}: In this case, clearly $A_{ij} \bigcup A_{kl} \neq \mathcal{U}$ since it will not contain both S_i and S_k.

□

Lemma III.1 gives us the basis for a simple algorithm to discover 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. Before, we describe the condition, we need a definition.

Definition III.1: 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 III.1 is a leaf interface. We can now state a necessary and sufficient condition for a router to be connected to a switch.

Lemma III.2: 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.

Figure 3 gives the pseudo-code for the edge discovery algorithm based on Lemmas III.1 and III.2.

```
Procedure FindInterConnections(S1, S2, ... Sn, R1, R2, ..., Rm)
/* S1, S2, ..., Sn are the switches of a subnet S */
/* R1, R2, ..., Rm are the routers of the subnet S */
begin
for each switch Si do
    for each interface j of Si do {
        If Sij has already been matched
            continue
        else {
            If Aij ⋃ Akl = U and Aij ⋂ Akl = φ
                Match Sij with Skl
                /* Sij and Skl are connected */
        }
    }
for each router Rk do
    for each switch Si do
        for each interface j of Si do
            If Sij is not matched and Aij contains Rk
                Match Sij with Rk
                /* Sij and Rk are connected */
end
```

Fig. 3. Interconnection between switches and routers.

IV. Multiple Subnet Switched Domains

As described in the previous section, for switched domains containing a single subnet, the topology of switches is easy to

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

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.

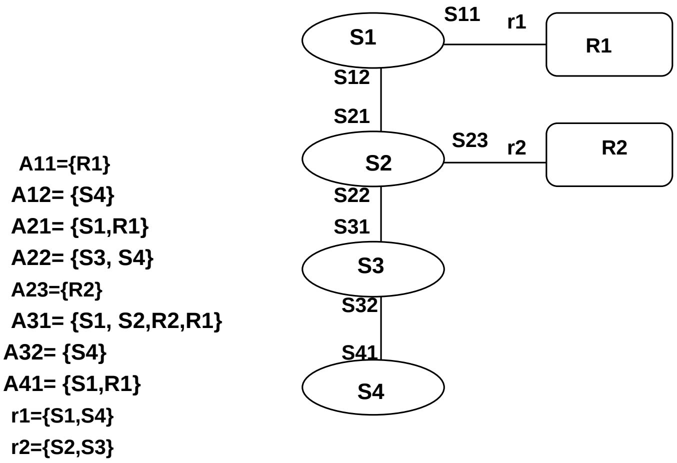

Fig. 4. Network containing multiple subnets.

Unfortunately, the assumption that switches in a switched domain are always from a single subnet may not always hold. For example, consider the network depicted in Figure 4. 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 to switch S_1 are routed through R_2.) 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 switches 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 switches 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 switches, based only on address forwarding information.

Consider the two distinct network topologies depicted in Figure 5. 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 Figure 5(a) and S_3 is connected to S_1 in Figure 5(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 information, we restrict ourselves to finding a minimal set of candidate topologies that contains the actual network topology.

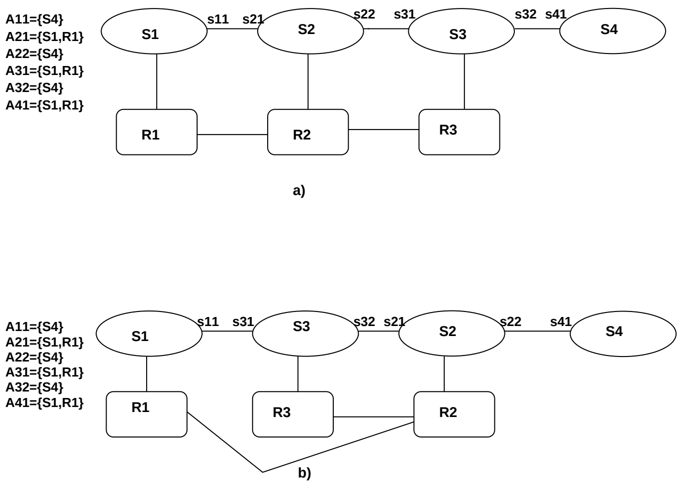

Fig. 5. Networks for which unique topology cannot be determined.

A. Properties of Switched Domains Containing Multiple Subnets

As mentioned earlier, the approach we adopt to discovering the topology of switched domains containing multiple subnets is to rule out interfaces that cannot be connected. In the following lemmas, we identify the conditions under which two interfaces cannot be matched. The lemmas make use of the following property for switched domains containing multiple subnets: Suppose S_i and S_k are two switches from different subnets; then, A_{ij} contains S_k if and only if there is a 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 union $A_{ij} \cup A_{kl}$.

Lemma IV.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 matched.

Proof: Suppose to the contrary that switch 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 and 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 non empty intersections they cannot be matched. □

Lemma IV.2: Let t be a subnet that contains at least two switches 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 matched.

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 switch 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 IV.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 matched.

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 $U_{ijkl} = U_{ijpt}$. 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 switch 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 the following contradiction: $S_k \in A_{kl}$. □

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 subsection in order to eliminate pairs of interfaces that cannot be matched. 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 IV.1, IV.2, and IV.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 switch S_p from a different subnet such that $U_{ijkl} = U_{ijpt}$ and $A_{ij} \cap A_{pt} = \phi$.

For all such pairs of potentially matching interfaces S_{ij} and S_{kl} satisfying the above conditions, we refer to unions U_{ijkl} as *valid unions*. For a valid unions 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 represent a superset of the actual connections in the network. Also note that, since between any pair of switches there can be at most one direct active connection, once we have connected an interface of S_i with an interface of S_k, all other valid unions containing both S_i and S_k can be eliminated.

Thus, the 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 (1) delete all valid unions containing S_{ij} from U except for U_{ijkl}, and (2) delete all valid unions U_{ixky}, $x \neq j, y \neq l$.
3. For every 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 Figure 5) 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 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 Figure 4, our algorithm for multiple subnets will in fact identify the correct network topology.

Example IV.1: Consider the network depicted in Figure 4. 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 matched with r_1. Similarly, S_{23} is matched with R_2. The remaining sets of addresses A_{ij} are listed below.

A_{12}	S_4
A_{21}	S_1, R_1
A_{22}	S_3, S_4
A_{31}	S_1, S_2, R_1, R_2
A_{32}	S_4
A_{41}	S_1, R_1

Valid unions are as follows:

U_{1221}	S_1, S_4, R_1
U_{2231}	$S_1, S_2, S_3, S_4, R_1, R_2$
U_{3241}	S_1, S_4, R_1

Note that $U_{1231} = \{S_1, S_2, S_4, R_1, R_2\}$ is not a valid union (due to Lemma IV.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 IV.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 matched with S_{21}, S_{22} is matched with S_{31} and S_{32} is matched with S_{41}. □

C. Characterization of Identified Topologies

In this section, we characterize a broad class of networks for which the algorithm developed in the previous section is guaranteed to identify the *unique* physical topology. We refer to this class of networks as *ordered* networks (formally defined below). We also define a set A of addresses to be *legal* if, for any subnet t, A contains either *all* or *none* of the addresses in t.

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

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 $< S_{ij}, S_{kl} >$. 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 $< S_{ij}, S_{kl} >$, the address table A_{ij} contains all the addresses in the subtree rooted at S_k. The second property, by requiring that roots and parents of 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 Figure 4 is an ordered network. To see this, consider the network arranged as the tree with switch S_2 as the root, as shown in Figure 6. 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 switches S_1 and S_4 constitute a pair of legal subtrees (since they contain all the addresses in subnet 1), and the switches themselves as well as their parents (S_2 and S_3) belong to the same subnet.

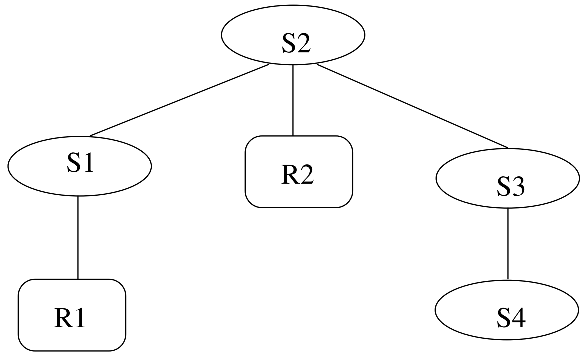

Fig. 6. Example of ordered network.

Theorem IV.1: The topology discovery algorithm presented in Section IV identifies the accurate physical topology for ordered network graphs.

Proof: For ordered networks, for any connection $< S_{ij}, S_{kl} >$, 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 contained 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 switch connections $< S_{ij}, S_{kl} >$ and $< S_{pq}, S_{uv} >$, $A_{ij} \neq A_{pq}$ and $A_{kl} \neq A_{uv}$. As a result, for the connection $< S_{ij}, S_{kl} >$, there can exist at most one other connection $< S_{pq}, S_{uv} >$ 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. 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 need to show that the valid unions U_{ijkl} and U_{pquv}, however, 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 S_{kl}. However, since $S_{kl} = S_{pq}$, S_i must be in the subtree rooted at S_u. This would mean that S_k is in the subtree rooted at S_u, and so $S_k \in S_{pq}$, which is impossible since $S_{kl} = S_{pq}$. □

We must note that our topology discovery algorithm can find the accurate topology for networks that may not be ordered. Figure 7 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 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 subclass of the class of networks for which our algorithm identifies the unique physical topology.

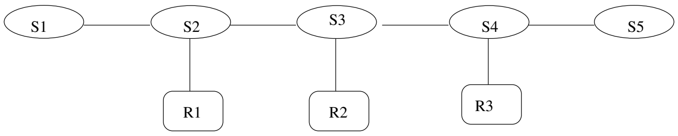

Fig. 7. Example of network that is not ordered.

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 VLANs.

A. Dealing With The Completeness Requirement

We have assumed thus far 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 is highly unlikely to be true. 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 switches. Therefore, unless a switch constantly receives packets from a source at intervals smaller than the aging interval (which is typically 5 minutes), the switch may delete the entry corresponding to that source. Thus, the A_{ij}'s may not be complete.

We present two complementary solutions to the above problem. The first solution attempts to keep the A_{ij}'s as complete as possible while the second attempts to handle minor deviations from completeness. These two together ensure that our algorithms work in practice as borne out by our experiments with the Bell Laboratories research network.

In our first solution, we try to ensure that the A_{ij}'s are as complete as possible by generating constant traffic between any pair of switches in the switched domain and, consequently, not allowing the address forwarding table entries to age. The mechanism we use to generate traffic from a node X to a 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 will cause X to respond to the Echo Request to Y. Making this work requires a minor modification of publicly available code for generating ICMP messages in order to build the appropriate IP header for the ICMP Echo Request.[3]

Our second solution handles minor deviations from completeness by choosing to match S_{ij} with the interface S_{kl} such that $A_{ij} \bigcap A_{kl} = \phi$ and $A_{ij} \bigcup A_{kl}$ contains either no switches from each subnet or a reasonably large fraction of the switches in the subnet (this fraction can be user-defined).

B. Handling VLANs

Virtual LANs (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 individually for each VLAN to generate the spanning tree for the VLAN. We only need to be careful to restrict ourselves to the universe of addresses consisting of only MAC addresses in the VLAN. Even though standard SNMP MIBs do not provide information on address forwarding tables for individual VLANs, this information can be collected using proprietary MIBs (for example, the Prominet MIB for Cajun Switches).

Our next example demonstrates that, even in the presence multiple subnets and VLANs in a switched domain, and in the absence of specific information on forwarding tables for each VLAN, our topology discovery algorithm (Section IV) can identify the correct topology.

Example V.1: Consider the network depicted in Figure 8. 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,

[3]A potential problem with this approach arises when switches in the network are connected through "out-of-band" interfaces since, in that case, 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* in the same subnet as "out-of-band"-connected switches, to ensure that the "in-band" interfaces are used.

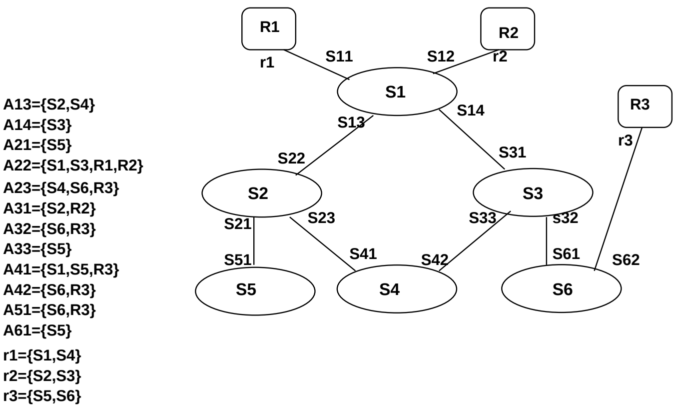

Fig. 8. Network containing VLANs and multiple subnets.

and router R_3 belong to subnet 3. In addition, there are 3 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 Figure 8.

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 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}	S_5, S_6, R_3
U_{2151}	S_5, S_6, R_3
U_{2341}	S_1, S_4, S_5, S_6, R_1, R_3
U_{3261}	S_5, S_6, R_3
U_{3342}	S_5, S_6, R_3
U_{3351}	S_5, S_6, R_3
U_{4261}	S_5, S_6, 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. Experiments

We have implemented the topology discovery algorithms presented in this paper and 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.

A. Implementation

We begin with a brief description of our implementation. A basic issue that we needed to resolve was how to ensure the completeness of the address sets in the elements' address forwarding tables. In our implementation, we addressed this issue by appropriately modifying a standard `ping` program in a way that allows our tool to submit `ping`s from a given source address to a given destination address. The modified `ping` program (termed `mping`) uses the raw socket option and, consequently, requires root privileges to run. To obtain complete address sets for each interface of each network element, we executed `mping`s across any pair of network elements in the set of elements in the input administrative domain. (For network elements connected through an "out-of-band" (management) interface, we employed `mping`s between *hosts* attached to these elements to populate the "in-band" address forwarding tables.)

Even with such exhaustive `mping`ing, however, we did discover situations where our implementation could not achieve complete address sets for certain interfaces. An example scenario occurs when the destination network element simply interchanges the source and destination MAC addresses to acknowledge an `mping` (as an optimization), instead of calling ARP to generate a MAC address for the acknowledgement message. In our experience with the system, however, such optimizations were not employed very often. Furthermore, even when such situations did occur and the address sets were incomplete, we employed appropriate approximations to deal with deviations from completeness (Section V-A). More specifically, our tool chose as valid interface unions those for which (a) the address set intersection was empty, and (b) the address set union U is closest (among all other such unions) to the union of the subnets containing the nodes in U. We found this approximation technique to work very well in practice.

Figure 9 depicts a high-level view of our implementation architecture. Users can submit a set of subnets for which a topology map is to be generated. All network elements located in one of the specified subnets are selected as input for the `mping`ing process which works by generating `mping`s across all network element pairs. Then, after some randomly selected time interval, the input generation program is run to produce the address sets for all element interfaces. Finally, our topology generation algorithm is executed on the collected address sets.

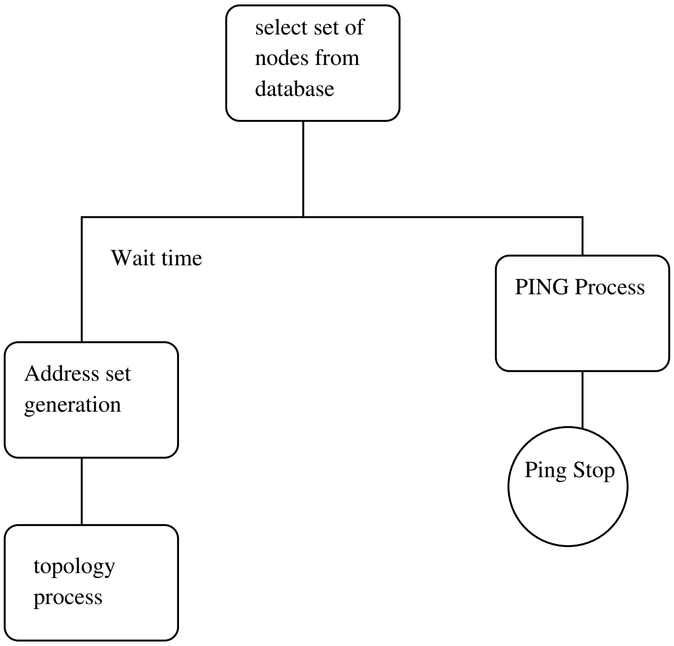

Fig. 9. Implementation architecture.

B. Results

A primary goal of the experimental study with our topology discovery tool was to ascertain the correctness and accuracy of our algorithms in a *real-life* networking installation. 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 *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. Figures 10 and 11 depict the topology maps of two multi-subnet networks discovered by our system. (Since these maps depict parts of Lucent's proprietary research network, we are using generic names for the network elements.)

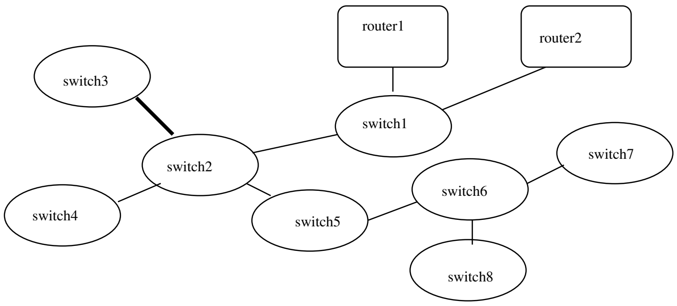

Fig. 10. Network with two subnets.

In Figure 10, 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 $switch_2$, $switch_7$, and $switch_8$ were not complete. Nevertheless, the approximation heuristics employed by our tool were sufficient to discover the correct network topology. We should also note that the connection between switches

$switch_2$ and $switch_3$ was in fact missing from the network administrators' manual map.

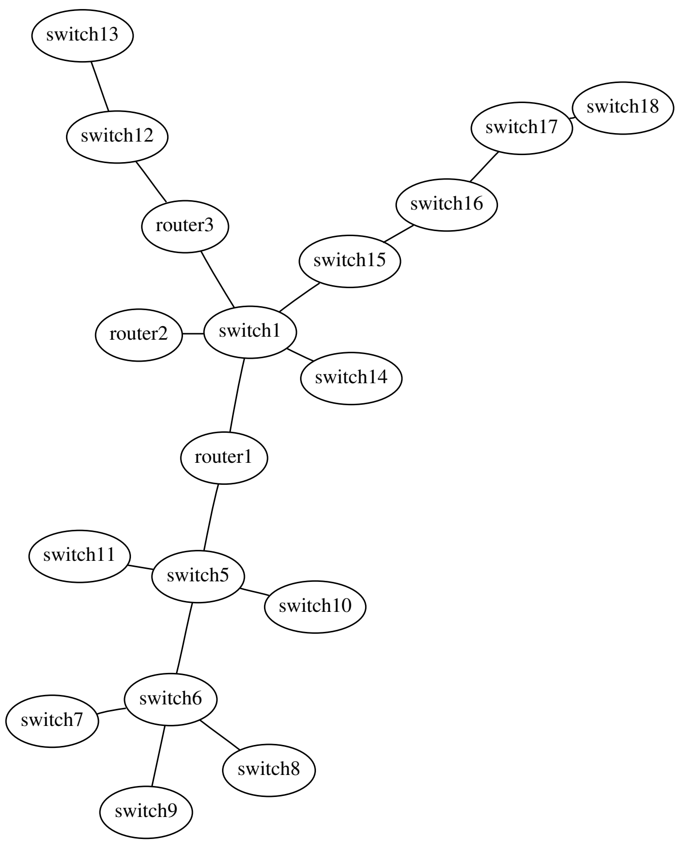

Fig. 11. Network with more than two subnets.

Figure 11 depicts the physical topology map discovered by our tool for a network with six distinct subnets composed as follows:

- **Subnet-1:** $switch_1$.
- **Subnet-2:** $switch_5$, $switch_{11}$, $switch_{10}$, and $router_1$.
- **Subnet-3:** $switch_6$, $switch_7$, $switch_8$, and $switch_9$.
- **Subnet-4:** $switch_{15}$, $switch_{16}$, $switch_{17}$, and $switch_{18}$.
- **Subnet-5:** $switch_{14}$ and $router_2$.
- **Subnet-6:** $switch_{12}$, $switch_{13}$, and $router_3$.

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. In general, we have discovered that our algorithm is sufficiently fast for all practical purposes; for example, it requires approximately 32 seconds to generate a topology map for a network with 100 elements. Furthermore, we found the running time of our algorithm to be roughly quadratic in the number of elements in the input network. We are currently in the process of optimizing the implementation to further improve the running time requirements of our tools.

VII. Conclusions

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 (a) 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 (b) 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 global layer-2 topology. Our algorithms can handle switched domains comprising one or more subnets and can be readily extended to deal with incomplete information and VLANs. Preliminary experimental results from an implementation of our topology discovery tools on Lucent's research network clearly validate our methodology, demonstrating the accuracy and practicality of the proposed algorithms. We are currently in the process of optimizing our implementation and conducting more extensive experimental tests, and hope to be able to report more detailed performance results in the near future.

References

[1] Irene Katzela and Mischa Schwarz, ""Schemes for Fault Identification in Communication Networks"," *IEEE/ACM Transactions on Networking*, vol. 3, no. 6, pp. 753–764, Dec. 1995.

[2] S. Yemini, S. Kliger, E. Mozes, Y. Yemini, and D. Ohsie, ""High Speed & Robust Event Correlation"," *IEEE Communications Magazine*, May 1996.

[3] R. Siamwalla, R. Sharma, and S. Keshav, ""Discovering Internet Topology"," Unpublished Manuscript (available from `http://www.cs.cornell.edu/skeshav/`), July 1998.

[4] J. Case, M. Fedor, M. Schoffstall, and J. Davin, ""A Simple Network Management Protocol (SNMP)"," Internet RFC-1157 (available from `http://www.ietf.org/rfc/`), May 1990.

[5] William Stallings, *"SNMP, SNMPv2, SNMPv3, and RMON 1 and 2"*, Addison-Wesley Longman, Inc., 1999, (Third Edition).

[6] K. McCloghrie and M. Rose, ""Management Information Base for Network Management of TCP/IP-based internets: MIB-II"," Internet RFC-1213 (available from `http://www.ietf.org/rfc/`), Mar. 1991.

[7] Andrew S. Tanenbaum, *"Computer Networks"*, Prentice Hall PTR (ECS Professional), 1996, (Third Edition).

[8] E. Decker, P. Langille, A. Rijsinghani, and K. McCloghrie, ""Definitions of Managed Objects for Bridges"," Internet RFC-1493 (available from `http://www.ietf.org/rfc/`), July 1993.