# Topology Discovery for Large Ethernet Networks

Bruce Lowekamp
Computer Science Dept.
College of William and Mary
Williamsburg, VA

lowekamp@cs.wm.edu

David R. O'Hallaron
School of Computer Science &
Dept. of Electrical and
Computer Engineering
Carnegie Mellon University
Pittsburgh, PA

droh@cs.cmu.edu

Thomas R. Gross
Computer Science Dept.
ETH Zurich
Zurich, Switzerland

trg@inf.ethz.ch

## ABSTRACT

Accurate network topology information is important for both network management and application performance prediction. Most topology discovery research has focused on wide-area networks and examined topology only at the IP router level, ignoring the need for LAN topology information. Recent work has demonstrated that bridged Ethernet topology can be determined using standard SNMP MIBs; however, these algorithms require each bridge to learn about all other bridges in the network. Our approach to Ethernet topology discovery can determine the connection between a pair of the bridges that share forwarding entries for only three hosts. This minimal knowledge requirement significantly expands the size of the network that can be discovered. We have implemented the new algorithm, and it has accurately determined the topology of several different networks using a variety of hardware and network configurations. Our implementation requires access to only one endpoint to perform the queries needed for topology discovery.

## 1. INTRODUCTION

Both network management and performance analysis benefit from network topology knowledge. Although network managers are responsible for maintaining the network, inevitable mistakes or the actions of other people may leave them without a true picture of their network's topology. With access to an automatically derived topology of their network, network managers are better able to react to and prevent problems. They can not only observe the exact location of a problem and trace it back to the source, but also analyze the use of the network under normal operations. This knowledge allows them to anticipate problems and plan for them before services are impacted. For planning parallel applications, topology knowledge allows the effects of link-sharing on an application's performance to be predicted in advance and nodes to be selected with appropriate network connections to match the application's needs. While the bottlenecks on LANs are generally not as severe as those on WANs, reliance on commodity components and the emergence of grid-based computing has increased the need for network-aware applications that can determine the network support available to them.

Although network topology information, especially at the LAN level, is important for both the management and use of networks, it is very difficult to obtain such information. The majority of network-management tools rely only on IP-level discovery of routers and require the network manager to enter level 2 devices such as Ethernet switches manually, without providing facilities for topology discovery. Cisco, Intel, and other hardware providers have designed their own proprietary topology discovery protocols, but these are of little use in a heterogeneous environment. More recently, the IETF has acknowledged the importance of LAN topology by designating an SNMP MIB to describe topology, but it failed to define a protocol for obtaining that topology [2]. Peregrine's Infratools software claims to support automatic topology discovery, but it is based on proprietary technology to which we do not have access, nor can we evaluate its effectiveness in the same variety of network environments.

The complexity of performing Ethernet topology discovery arises from the inherent transparency of Ethernet bridge hardware. Endpoints are unaware of the presence of bridges in the network. The bridges themselves only communicate with their neighbors in the limited exchanges of the spanning tree protocol, and that is not used in all environments. The only state maintained by bridges is their forwarding database, which is used to direct incoming traffic to the appropriate destination port. Fortunately, this information is sufficient for topology discovery, and because it is available through a standard SNMP MIB, it is portable enough to enable automatic topology discovery on almost all Ethernet bridges.

Recently, Bertsekas et al. described an algorithm for performing topology discovery using information available from Ethernet bridges [3]. Their algorithm obtains good accuracy, but requires that each bridge have a forwarding entry for all other bridges in the network. Because bridges do not normally communicate with each other, and because obtaining complete information from forwarding databases can be a challenging process, this requirement makes it difficult to obtain sufficient information to derive the topology and requires more work and greater access to machines on the network as the number of bridges grows.

In this paper, we describe a new algorithm that can perform automatic topology discovery without requiring complete knowledge. In fact, our algorithm requires only forwarding entries for three machines to be shared between each pair of bridges, when those machines are appropriately placed. Using this approach, we have derived the topology of Ethernet networks with up to 2000 nodes and 50 bridges while using only one machine to send queries and pings. Because our algorithm has no requirement for complete in-

formation, it can determine the topology of very large networks, from which it is typically more difficult to obtain complete information.

We begin by describing related topology discovery work. Section 3 describes the Remos system for which the topology discovery algorithm was originally developed. Section 4 describes the direct approach with the completeness requirement, how it is applied to networks, and its shortcomings. Section 5 describes our new approach, which derives topology using minimal knowledge from the forwarding databases. This section includes both a general discussion and formal proofs of the theorem itself and the minimum knowledge requirement. Section 6 describes details related to implementing topology discovery using our algorithm. Section 7 describes some of the challenges of implementing the topology discovery algorithm on real networks. Section 8 describes the results obtained using our implementation on several different networks.

## 2. RELATED WORK

The Internet has been the focus of most work on topology discovery. Because it forms the bottleneck for nearly all wide-area applications, server selection and placement generally make Internet topology discovery and performance measurement an interesting and rewarding problem. The most common technique for determining WAN topology is to use benchmarks to measure bandwidth across the topology. Topology-d was an early project that used measurements to build a minimal spanning tree view of the network [15]. Current work, such as the IDMaps project [9] and the work of Theilmann and Rothermel [18] build network distance maps, a more flexible representation of the network's topology. These projects have studied the best types of benchmarks as well as placement of the sites running the benchmarks for accurate topology determination.

A lower level approach has been to use the hop-by-hop feedback provided by tools such as traceroute. The Mercator project has explored this technique to group IP addresses by network topology to produce an Internet map [8]. Although many modern routers no longer respond to traceroute packets, they have achieved good results on the modern Internet. Skitter has been developed by CAIDA to combine traceroute and benchmark-based analysis [4]. Octopus combined SNMP routing information, traceroute, measurements, and heuristics to determine network topology [17].

Although there has been a significant amount of research studying the importance of topology in task placement for LANs [1, 5, 7, 10], there has been less work on the automatic determination of LAN topology than WAN topology. A number of projects have looked at discovering the topology between IP routers, but because the most interesting portions of LAN topology are generally formed by level 2 devices, these projects have been unable to address the majority of LAN topology issues.

ECO demonstrated that benchmarks could be used to discover LAN topology on older, shared Ethernet networks, but these techniques are of little value on modern switched networks [11]. More recent work by Shao, Berman, and Wolski has focused on using benchmark measurements to determine functional differences between a central server and a collection of machines across a network, which they refer to as effective network views [16]. By experimenting with different combinations of machines, their technique groups machines into clusters with good accuracy. However, for many purposes in parallel computing and network management, this view of the network's structure is insufficient.
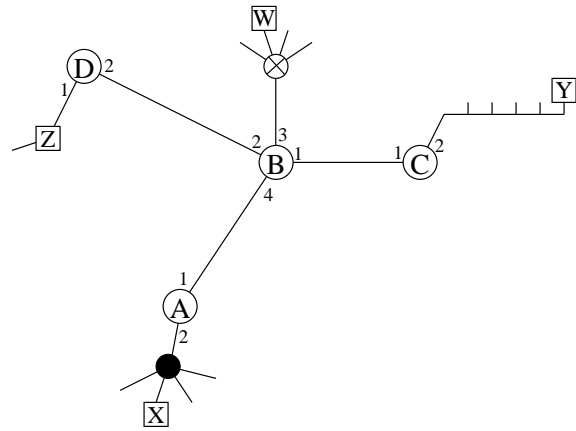


Figure 1: A sample bridged Ethernet network. A, B, C, and D are bridges. W, X, and Y are hosts. Z is a router. The port numbers of the bridges are indicated next to their respective links. A hub is shown connected to A, a non-SNMP enabled switch to B, and a shared segment to C.

## 3. REMOS

The topology discovery algorithm was originally developed as part of Remos, a system for providing resource monitoring information to network-aware applications [12]. In that framework, this algorithm is used to implement the Bridge Collector, which is responsible for discovering the topology of an Ethernet LAN. That topology is then given to the SNMP Collector, which is responsible for monitoring network utilization and for determining routed (level 3) topology. The resulting network topology and measurements are then available to applications through the Remos API. Remos provides different collectors for WANs and LANs [14], simple predictors of future network utilization [13], and also incorporates software for host load prediction [6].

## 4. TOPOLOGY DISCOVERY WITH DIRECT CONNECTIONS

Our goal was to design an algorithm that determines the physical topology of the Ethernet network used to connect a set of hosts. To accomplish this goal, we chose to derive the network topology using information obtained from the bridges themselves with SNMP.

A bridged network containing the nodes $N$ can be divided into a set of bridges, $B$, and endpoints, $E$. Figure 1 shows a sample network. As shown in this figure, $B = \{A, B, C, D\}$. $E$ may contain both hosts, $\{W, X, Y\}$, and routers, $\{Z\}$, which are identical to hosts for the purpose of the Ethernet bridging algorithm. Modern Ethernet networks are generally built with "switches," which are essentially bridges with many ports, and the term can be used interchangeably with "bridges." Bridged Ethernet networks may also include the following: hubs, which serve to connect several machines off one port of a bridge via shared Ethernet, as shown connected to A; shared segments, shown connected to C; and "dumb" switches, which do not speak SNMP, shown connected to B.

Bridges connect Ethernet segments together in an acyclic network. If cycles are present in the physical connections, the bridges use the Ethernet spanning tree algorithm to select an acyclic subset of those connections. Once the topology has been selected, bridges learn the location of machines on the network by monitoring traffic on each of their ports. When a new node sends a message that appears on that port, the bridge adds it to the list of nodes found

off that port. Later, when the bridge receives a message bound for that node, it knows to forward the message to that port. The database that maps addresses to ports is known as the forwarding database (FDB). For a bridge, $C$, we denote the forwarding set for port $x$ as $F_C^x$. In Figure 1, $F_C^1 = \{A, B, D, W, X, Z\}$. This set changes as new addresses are learned, as old addresses expire, or as machines are physically moved to part of the network connected to a different port. $F_a^x$ is said to be *complete* if it contains a forwarding entry for each member of $N$ found off that port. The example given for $F_C^1$ is complete.

The first step in discovering a network's topology is determining how a pair of bridges is connected. Two nodes are referred to as *directly connected* if there are no other nodes between them. In particular, if the packets that bridge $A$ sends on port $x$ are received by port $y$ on bridge $B$ without going through any other device, bridges $A$ and $B$ are directly connected via ports $x$ and $y$, respectively. In Figure 1, A and B are directly connected by ports 1 and 4, as are B and D by ports 2 and 2. Furthermore, two bridges are *simply connected* by ports $x$ and $y$ if they find each other off those ports, but there may be other nodes in between. A and C are simply connected by ports 1 and 1. Directly connected nodes, such as A and B, are also simply connected by ports 1 and 4.

At a high level, this algorithm begins by determining all entries in the FDBs, ensuring they are complete. It then selects a single bridge and determines the bridges that are directly connected to each port. The Ethernet topology is then filled in by traversal. At the heart of this algorithm is the direct connection theorem, which is used to determine when two bridges are directly connected.

THEOREM 4.1 (DIRECT CONNECTION THEOREM). *Assume that $F_i^x$ and $F_j^y$ are complete. Two bridges $i$ and $j$ are directly connected via the link connected to port $x$ on $i$ and port $y$ on $j$ if and only if $F_i^x \cap F_j^y = \emptyset$ and $F_i^x \cup F_j^y = N$.*

The proof is omitted here. Interested readers are referred to Breitbart et al. [3] for proof of a similar theorem.

## 4.1 Shared segments

For a simple switched Ethernet, Theorem 4.1 is sufficient, but most networks do not use direct connections to connect all nodes to one another. For example, shared segments are commonly used at the leaves of networks, such as the hub attached to A in Figure 1 or the shared segment attached to C. These typical uses of shared segments do not cause problems for Theorem 4.1. However, it is also possible to create shared segments internal to the topology of the bridged Ethernet (between bridges in the topology). These interior shared segments cause problems and generally occur in two situations:

1. A hub is used to connect two bridges with other hosts or bridges. Because hubs do not participate in the bridging algorithm, this creates a shared network segment between the bridges. The reality is that a properly designed network would never contain such a hub, but the real world does not guarantee that all networks will be properly designed. We have also encountered a shared segment deliberately placed between switches to facilitate traffic monitoring on the external network connection.

2. A bridge exists in the network from which the program cannot obtain a forwarding database. This situation can occur either when the program is not informed of the bridge's existence or when SNMP access is denied to that bridge. Because SNMP protection generally consists of a simple allowed/denied list of IP addresses, this situation can easily

occur and did, in fact, occur on a number of occasions during the development of the topology discovery system.

Both of these cases cause problems because they form networks where there are no direct connections between bridges. Consider Figure 1 if B is replaced with a hub or if SNMP access is unavailable to B. In that case, there is no direct connection that can be established between A, C, and D.

Because Theorem 4.1 detects only direct connections and not bridges connected with shared segments, a new rule is required. To develop this rule, first let $a(b)$ be the port of bridge $a$ that address $b$ is found off of. Let $S_B$ be the set of bridges connected to the shared segment. (Note that $|S_B| > 1$.) Let $S_E$ be the endpoints attached to the shared segment. The entire shared segment is denoted $S = S_B \cup S_E$.

THEOREM 4.2 (SHARED SEGMENT THEOREM). *$S$ consists of a shared segment between the bridges in $S_B$ if and only if $\forall a \in S_B, \forall b, c \in S : a(b) = a(c)$ and all forwarding databases are complete.*

In other words, on a shared segment, all bridges must find all members of the shared segment on the same port. This theorem defines a shared segment and can be used to verify that a set of nodes forms a shared segment.

**Proof** $\Leftarrow$: Assume that $\forall a \in S_B, \forall b, c \in S : a(b) = a(c)$, but the members of $S_B$ are not connected with a shared segment. If $S$ is not shared, then there exists a switch in $S_B$ that has elements of $S$ on different ports, by the definition of a shared segment. More formally, and by applying the completeness requirement, $\exists a \in S_B, \exists b, c \in S : a(b) \neq a(c)$. This contradicts the original assumption. Therefore the bridges are connected with a shared segment if $\forall a \in S_B, \forall b, c \in S : a(b) = a(c)$.

$\Rightarrow$: Assume that the members of $S_B$ are connected via a shared segment containing $S_E$. By definition of a shared segment, no bridges in $S_B$ may have elements of $S$ on two different ports, otherwise, by the acyclicity requirement, the segment would be switched, rather than shared. Therefore, every member of $S_B$ has all members of $S$ on the same port. Stated formally, $\forall a \in S_B, \forall b, c \in S : a(b) = a(c)$ if the bridges are connected with a shared segment. $\blacksquare$

## 4.2 Limitations of the direct connection theorem

The most significant shortcoming of the direct connection theorem is the completeness requirement. Requiring entries for all, or nearly all, of the machines in an Ethernet is infeasible for any moderately sized network. First, many bridges are connected to the network with "out-of-band" ports, which do not participate in the main network, but are instead on a second network used for administration and are especially helpful when the primary network is down. These bridges may not participate in the topology of the network they are forming. Secondly, to generate entries in the forwarding databases, all machines involved in the topology must be generating traffic that is seen by all bridges in the network. This is unlikely to occur naturally and difficult to do automatically without accounts on every machine in the network.

As the network grows larger, the problem of obtaining complete forwarding sets becomes even more challenging. In a large network with hundreds or thousands of machines, more and more of the machines will be down or unresponsive at any given time, whether due to hardware failure, software failure, being rebooted, or other problems associated with maintaining a large network. Capturing a complete, consistent picture in a real-world dynamic network environment becomes less and less likely as the network grows.

Even in situations where sufficient traffic can be generated, there are still challenges. Although bridges supply their forwarding databases using SNMP, that interface appears to be a direct link into a dynamic data structure on several bridges we encountered. As that data structure changed, the traversal ordering of the entries changed, sometimes resulting in duplicate entries being received and other times skipping large portions of the FDB. These databases can require several attempts to obtain entries for most of the addresses in the network.

Although the direct connection approach is still usable with appropriate modifications and heuristic assumptions, the difficulties in applying it to real-world large Ethernet networks motivated the development of a new approach that does not require complete information.

## 4.3 Related work

The algorithm described in this section is very closely related to work previously published by Breitbart et al. from Lucent [3]. There are, however, significant differences between the two approaches that are worth discussing. The researchers from Lucent focused on Ethernet networks with multiple IP subnets and VLANs. They found that by extending their algorithm with information available from non-standard vendor MIBs, their algorithm could handle topology discovery on networks using VLANs.

On the other hand, their algorithms do not address the problems of shared segments appearing between bridges, whereas our experience with academic and research networks required support for shared segments in the algorithm. There were also a number of practical challenges that our algorithm had to deal with to support the Ethernet LANs at CMU that were apparently not a problem for the networks at Lucent. The unusual network configurations encountered by our group and the Lucent group generally seem to correspond to the nonstandard aspects of the networks we each sought to support.

The most significant difference between the approaches is that the Lucent group focused on forwarding sets containing only the addresses of bridges and routers in the network, whereas our approach also uses the addresses of endpoints on the network. There is a tradeoff here—although there are many more hosts, which make it harder to obtain complete databases, the penalty for missing a single entry is less severe because there are so many entries. Although using only forwarding entries for bridges reduces the number of entries needed for complete sets, they still experienced difficulties obtaining complete sets. They developed two approaches to improve the completeness of their FDBs.

The first technique was to use "spoofed" ICMP-echo packets to force bridges to communicate with each other. In our network, we found that approximately half of the bridges replied to these packets by putting the requested IP return address in the reply packet, but sending that packet to the Ethernet MAC address of the machine that originally sent it, rather than to the MAC belonging to the "spoofed" return address, thus defeating the purpose of the exercise. In our experiments, therefore, all traffic was creating using endpoints.

The Lucent group's second approach was to relax the completeness requirement by requiring only a user-defined fraction of the entries to be present in the forwarding databases for them to be called complete. This approach works well in situations where the majority of entries are present, with a small number of entries missing on each machine.

Although the Lucent group found that their techniques for addressing the completeness requirement resolved the difficulties for their network, we sought to develop an algorithm that avoids the
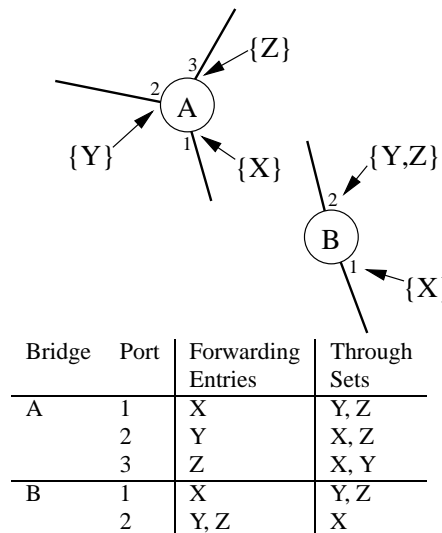


| Bridge | Port | Forwarding Entries | Through Sets |
|--------|------|--------------------|--------------|
| A | 1 | X | Y, Z |
|   | 2 | Y | X, Z |
|   | 3 | Z | X, Y |
| B | 1 | X | Y, Z |
|   | 2 | Y, Z | X |

**Figure 2: Example of two bridges for which contradictions can be used to determine the connections.**

challenges of obtaining complete information, and instead makes more efficient use of the incomplete information typically available from bridges.
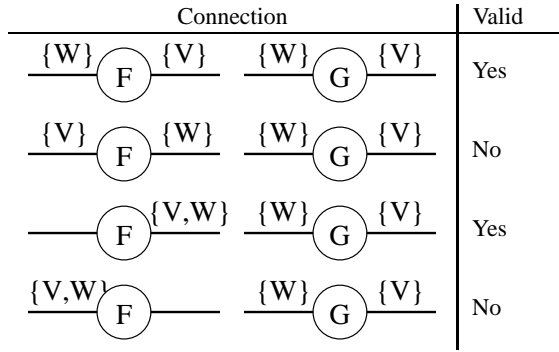
## 5. TOPOLOGY DISCOVERY WITH INCOMPLETE KNOWLEDGE

Because of the difficulties associated with obtaining complete forwarding databases for entire Ethernet LANs, we decided to pursue an alternative approach to topology discovery. Rather than proceeding with the goal to prove that two nodes are directly connected, consider the approach of proving that bridge $a$ is not simply connected to port $x$ of $b$, denoted $b_x$. Because this is a proof by contradiction, only a single counterexample is needed to demonstrate the contradiction.

For the remainder of this paper, the word "connection" will usually refer to a simple connection, rather than a direct connection. When a distinction is necessary, it will be made clear that a direct connection is being discussed.

Figure 2 shows an example where contradictions can be used to determine how two bridges are connected. The motivating concept here is to hypothesize that two bridges are simply connected by a particular pair of ports. Remember that there may be nodes in between, as this is not a direct connection. If all entries in the forwarding database are consistent with this connection, it may exist. If any one entry is inconsistent, then that connection is impossible.

Before considering how to connect the two bridges in Figure 2, consider the four hypothetical examples of valid and invalid connections shown in Figure 3. The only situation where a contradiction can be found in making a connection is where each of the two bridges have a forwarding entry for the same address on ports other than the connecting ports. In other words, they claim the same machine is in two different places on the network. In Figure 3, this rule excludes the second and fourth examples, where the two bridges forward the same address in opposite directions. Anytime there are not directly conflicting forwarding entries, the connection may be valid. Note that in the third example of Figure 3, node W is between bridges A and B. Because this rule establishes a simple connection and not a direct connection, this is perfectly valid.

| Connection | | Valid |
|---|---|---|
| {W} F {V} | {W} G {V} | Yes |
| {V} F {W} | {W} G {V} | No |
| F {V,W} | {W} G {V} | Yes |
| {V,W} F | {W} G {V} | No |

**Figure 3: Examples of valid and invalid connections between two bridges with different forwarding entries on bridge F. In each case, the forwarding entries are examined to see whether those two edges can form a valid simple connection. Note that the only cases where there are conflicts occur when the bridges map the same address in opposite directions.**

| Ports A | B | Mapping | Conflict |
|---|---|---|---|
| 1 | 1 | {Z} / {X} A — {X} B {Y,Z} / {Y} | Y and Z |
| 3 | 1 | {Y} / {Z} A — {X} B {Y,Z} / {X} | Y |
| 2 | 2 | {Z} / {Y} A — {Y,Z} B {X} / {X} | X |
| 2 | 1 | {Z} / {Y} A — {X} B {Y,Z} / {X} | Z |
| 1 | 2 | {Z} / {X} A — {Y,Z} B {X} / {Y} | **NONE** |
| 3 | 2 | {Y} / {Z} A — {Y,Z} B {X} / {X} | X |

**Figure 4: How contradictions can be used to eliminate impossible connections from the bridges in Figure 2.**

Figure 4 shows how contradictions can be used to determine the only valid connection between the bridges in Figure 2. All six possible connections between the bridges are shown. In five of the six connections, a contradiction of the two bridges forwarding the same address in opposite directions was found. Only the valid connection is left.

The most important observation about the application of this simple rule is that there is no requirement for complete knowledge. Instead of requiring complete knowledge, there is merely the minimal requirement of having enough information to rule out all but one possible connection. We will define and prove this minimum knowledge requirement. In the example in Figure 4, the entries for these three addresses meet the minimum knowledge requirement. There may be many other nodes in the network, and there may be entries for many other nodes on one or both of bridges A and B, but as long as the minimum information is captured as it is in this example, the correct connection will be determined.

To simplify the presentation of the algorithm, We will first introduce additional notation. The "through sets" indicated in Figure 2 contain the addresses that are forwarded through each port—in other words, the addresses that are on other ports, for which the bridge will forward packets through itself. The through sets for each port are, in a sense, the complement of $F_i^x$ and are denoted $T_i^x$.

After determining through sets for each port, it is simple to apply this information to determine the ports that connect two bridges. To see if ports $x$ and $y$ of bridges $a$ and $b$ are connected, calculate $T_a^x \cap T_b^y$. If there are any addresses in common, then the ports cannot be connected because a single node cannot be in two different directions on an Ethernet, which is required to be acyclic. If the intersection is empty, they can be connected.

By simple iteration, it is possible to map each bridge to the port it appears off every other bridge. Applying this technique to each bridge allows the complete topology of the network to be determined.

This approach has several advantages over the algorithm based on the Direct Connection Theorem:

- Rather than relying only on information about one port, information from all ports is combined for each mapping question. This is especially helpful for ports with few machines connected to them, because it allows data to be aggregated without requiring that it be complete.

- Incomplete information is tolerated much more easily.

- Shared segments are naturally determined from the primary theorem, rather than relying on a special case to resolve conflicts when the direct connection theorem fails.

- The approach offers positive rejection: if insufficient information is available to perform a mapping, that error is detectable and distinguishable from a case where a mapping is performed with incomplete, although sufficient, data.

## 5.1 Rigorous presentation

THEOREM 5.1 (SIMPLE CONNECTION THEOREM).
Let $a, b \in B$. Suppose there exists exactly one pair of ports $a_x$ and $b_y$ such that $T_a^x \cap T_b^y = \emptyset$. Then $a_x$ and $b_y$ are connected. Furthermore, if $a_x$ and $b_y$ are connected, then $T_a^x \cap T_b^y = \emptyset$.

Remark: Note that $a_x$ and $b_y$ being connected does not imply that $T_a^x \cap T_b^y = \emptyset$ for only one pair $x$ and $y$. In other words, there are
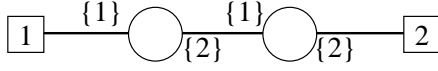
**Figure 5: This network consists of two bridges and two hosts, labeled 1 and 2. The connection of the two bridges cannot be determined because there is insufficient information. Observe that the bridges can be placed in either position, as they are indistinguishable using their forwarding databases.**



**Figure 6: Examples of the three required shared entries for rule 3 of the Minimum Knowledge Requirement. (a) illustrates a shared entry between $a_x$ and $b_i$, which is insufficient to determine $x$. (b) adds a shared entry with $b_j$, which determines $x$, but not $y$. Note that although $i \neq j$, $y$ can equal $i$ or $j$. (c) illustrates the shared entry needed to determine $y$ once $x$ is determined. Note that $k \neq x$.**

indeterminate solutions where insufficient information is present. See Figure 5 for an example.

**Proof** $\Rightarrow$: Assume that $a_x$ and $b_y$ are connected. Partition the network into three partitions, let $N_S$ be the partition between $a_x$ and $b_y$, let $N_a$ be the partition containing $a$, and let $N_b$ be the partition containing $b$. $T_a^x \subseteq N_a$ and $T_b^y \subseteq N_b$. Because $N_a$ and $N_b$ are a partitioning of $N$, $T_a^x \cap T_b^y = \emptyset$. Therefore, $T_a^x \cap T_b^y = \emptyset$ if $a_x$ and $b_y$ are connected.

$\Leftarrow$: Assume that $T_a^x \cap T_b^y = \emptyset$ for only one pair $x$ and $y$, but $a_x$ and $b_y$ are not connected. Because $a$ and $b$ belong to a connected Ethernet, they must be connected by some pair of ports. Let $a_i$ and $b_j$ be the true ports connecting the bridges. From the first half of the proof, $T_a^i \cap T_b^j = \emptyset$. However, this contradicts the assumption that the intersection is empty for only one pair of ports. Therefore, by contradiction, $a_x$ and $b_y$ are connected if $T_a^x \cap T_b^y = \emptyset$ for only one pair $x$ and $y$. $\blacksquare$

The requirement of Theorem 5.1 that the intersection is empty for only one pair $x$ and $y$ is the *minimum knowledge requirement*. It serves to prevent trivial solutions to $T_a^x \cap T_b^y = \emptyset$ that exist only because the forwarding databases have insufficient information or because the network topology is structurally indeterminate. Consider the network shown in Figure 5. These two bridges can be arranged in either order because the knowledge they give is insufficient to determine an ordering between themselves using only the forwarding entries for the hosts shown.

The minimum knowledge requirement can be met by any pair of bridges that meet the following rule:
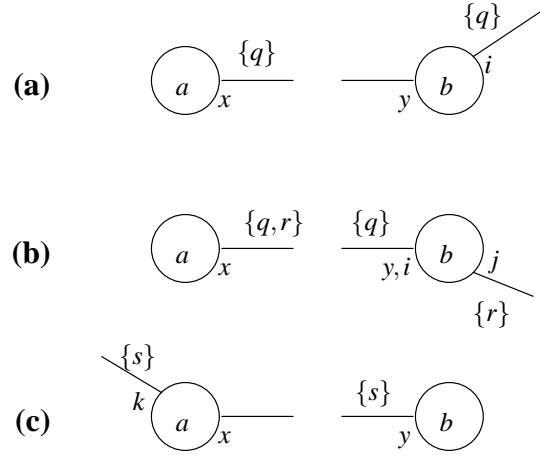
LEMMA 5.2 (MINIMUM KNOWLEDGE REQUIREMENT).
*The ports $x$ and $y$ that connect bridges $a$ and $b$ are uniquely determined if and only if at least one of these conditions is met:*

1. *Each bridge has an entry for the other's address in its FDB, not including out-of-band ports; or*

2. *Bridge $a$ has an entry for $b$ in $F_a^x$ and $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$; or*

3. *Forwarding entries for three nodes are shared between $a$ and $b$, divided among at least two ports on one of $a$ or $b$ and three ports on the other bridge. $x$ and $y$ must be included in those ports. Formally, $\exists i, j, i \neq j : (F_a^x \cap F_b^i \neq \emptyset \wedge F_a^x \cap F_b^j \neq \emptyset)$, and $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$.*

**Proof** The first condition is trivial—if each bridge has an entry for the other in their FDBs, then the entries directly indicate which ports are used to connect the bridges, and a contradiction is generated whenever the wrong pair of ports is considered for connection. If that option is not available, one of the other two must be met.

For the second condition, port $x$ is uniquely determined to be the port of $a$ connected to $b$ because of the explicit FDB entry. Port $y$ on $b$ is uniquely determined because there is an entry shared between $b_y$ and a port on $a$ other than $a_x$. If a connection is tried using any port on $b$ other than $y$, these two entries will cause a contradiction
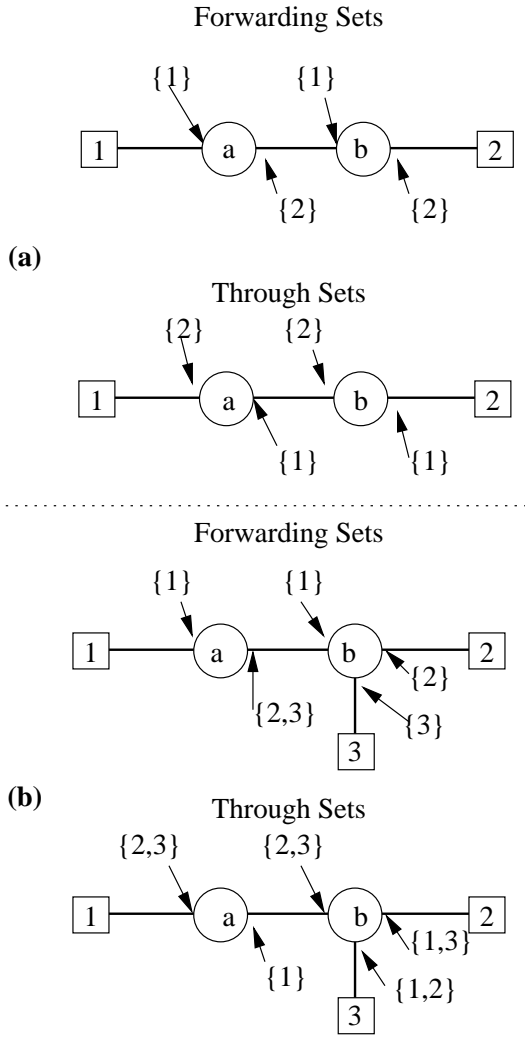
by pointing in different directions for the same entry. This is the important condition required by $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$.

Proving the second half of rule two formally: $\Leftarrow$: Assume $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$. Let $c \in F_b^y \cap F_a^k$. Because $k \neq x$, $c \in T_a^x$. Also, $\forall l \neq y : c \in T_b^l$. Because $c$ is in both sets, $T_b^l \cap T_a^x \neq \emptyset$. Therefore, if $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$, $y$ is uniquely defined.

$\Rightarrow$: Assume $y$ is uniquely defined, but $\forall k \neq x : F_b^y \cap F_a^k = \emptyset$. Because $x$ and $y$ are connected, $T_b^y \cap T_a^x = \emptyset$. By definition of through sets, $\forall l \neq y, \forall k \neq x : F_b^l \cap F_a^k = \emptyset$. Combined with the initial assumption, $\forall l, \forall k \neq x : F_b^l \cap F_a^k = \emptyset$. Again by the definition of through sets, $\forall l : T_b^l \cap T_a^x = \emptyset$. This contradicts the assumption that $y$ is uniquely defined. Therefore if $y$ is uniquely defined, $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$.

The third, and fully general condition is an extension of the logic used for condition two to using intersections to make the unique determination of both ports $x$ and $y$. The third case requires that $F_a^x$ must have members also found in two ports of $b$, denoted $F_b^i$ and $F_b^j$. This condition is sufficient to uniquely determine $x$. To see why an entry must be shared with two ports, whereas for condition two, a shared entry with only one port was required, consider the case where there is an entry shared between port $a_x$ and port $b_i$, but no entries shared with any port other than $b_i$. This situation is illustrated in Figure 6a. In this case, $a_x$ can be connected to any port on $b$ without creating a conflict. However, $b_i$ can also be connected to any port on $a$ without creating a conflict, $q$ will be forwarded from $b$ through $b_i$ to $a$ and on through $a_x$. Because $b_i$ can be connected to any port on $a$, this single shared entry does not uniquely define $x$.

Next, consider the added requirement that there is an additional entry shared with $b_j$, as illustrated in Figure 6b. In this case, if $a_x$ is connected to $b$, the connection is valid. However, if any port of $a$ other than $a_x$ is used for the connection, a conflict is created. If $b_i$ is connected to a port on $a$ other than $a_x$, $r$ will create a conflict because it is forwarded in opposite directions. Likewise, a conflict will be created if $b_j$ is connected to a port other than $a_x$, as $q$ will now be forwarded in opposite directions. If a port on $b$ other than

Forwarding Sets

$\{1\}$      $\{1\}$

1 — a — b — 2

$\{2\}$      $\{2\}$

**(a)**

Through Sets

$\{2\}$      $\{2\}$

1 — a — b — 2

$\{1\}$      $\{1\}$

Forwarding Sets

$\{1\}$      $\{1\}$

1 — a — b — 2

$\{2,3\}$    $\{2\}$   $\{3\}$

3

**(b)**

Through Sets

$\{2,3\}$      $\{2,3\}$

1 — a — b — 2

$\{1\}$   $\{1,3\}$   $\{1,2\}$

3

**Figure 7: Two network graphs are shown with forwarding and through sets indicated. Network (a) does not provide sufficient information to meet the minimum knowledge requirement. Network (b), however, with one additional host, meets that requirement.**

$b_i$ or $b_j$ is connected to a port other than $a_x$, then both $q$ and $r$ will present conflicts. Therefore, we see that these two shared entries uniquely determine $a_x$ as the port used to connect $a$ to $b$, although $y$ is still not determined, as any port of $b$ can be connected to $a_x$ without generating a conflict.

Once $x$ is uniquely determined, $y$ is determined with the same rule as used for condition two. Figure 6c illustrates this rule. Because $k \neq x$, the shared entry $s$ between $a_k$ and $b_y$ forces $y$ to be the port connected to $a$, otherwise a conflict would be creating by mapping $s$ in opposite directions.

Consider the example of an indeterminate network shown in Figure 7. In network (a), there is not a unique solution to the mapping problem. The bridges share two entries, but they are symmetric and can be placed in either order. Network (b), however, resolves the indeterminism. By adding a second machine off of one of the bridges which is shared in both bridges' FDBs, the network is uniquely determined. This second machine gives $a$ shared entries with two of $b$'s ports. It is no longer possible to reorder these two bridges be-

cause the only port on the left bridge that can be connected to the right one is the one with both entries. Any other connection will cause a contradiction.

The formal proof of condition three begins by proving that $\exists i,j, i \neq j : (F_a^x \cap F_b^i \neq \emptyset \wedge F_a^x \cap F_b^j \neq \emptyset)$ uniquely determines $x$.

$\Leftarrow$: Assume $\exists i,j, i \neq j : (F_a^x \cap F_b^i \neq \emptyset \wedge F_a^x \cap F_b^j \neq \emptyset)$. This implies that $\forall z : F_a^x \cap T_b^z \neq \emptyset$ because either $F_b^i$ or $F_b^j$ will be in $T_b^z$. Accordingly, $\forall l \neq x, \forall z : T_a^l \cap T_b^z \neq \emptyset$. Therefore, $x$ is uniquely determined if $F_a^x \cap F_b^i \neq \emptyset$ and $F_a^x \cap F_b^j \neq \emptyset$.

$\Rightarrow$: Assume that $x$ is uniquely determined, but that $\forall i \neq j : F_a^x \cap F_b^i = \emptyset \vee F_a^x \cap F_b^j = \emptyset$. In other words, there is at most one $i$ such that $F_a^x \cap F_b^i \neq \emptyset$. Consider two cases:

- First, that there is no set $F_b^i$ such that $F_a^x \cap F_b^i \neq \emptyset$. We know that $T_a^x \cap T_b^y = \emptyset$. Combining these two, $\forall l, \forall i : F_a^l \cap F_b^i = \emptyset$. Therefore, there are no common members, which contradicts the initial assumption that $x$ is uniquely determined.

- The second case is that there exists only one $i$ such that $F_a^x \cap F_b^i \neq \emptyset$. Note that network (a) in Figure 7 meets this condition, but is indeterminate. By example, this is a contradiction.

Therefore, if $x$ is uniquely determined, $\exists i,j, i \neq j : (F_a^x \cap F_b^i \neq \emptyset \wedge F_a^x \cap F_b^j \neq \emptyset)$.

Once $x$ is uniquely determined, the proof of rule 2 proves $y$ is uniquely determined if and only if $\exists k \neq x : F_b^y \cap F_a^k \neq \emptyset$. ∎

Therefore, it has been shown that Lemma 5.2 represents the minimum knowledge required for Theorem 5.1 to determine the simple connection between two bridges in a network.

## 5.2 Practicality

The next question to be asked is whether Lemma 5.2 is a realistic expectation for bridged Ethernet networks to meet. After all, the motivation for pursuing this technique is that it is much easier to satisfy the minimum knowledge requirement than to require the FDBs to be complete.

Consider the four snapshots of two bridges shown in Figure 8. Imagine that the real position of these two bridges has bridge A internal to the topology and bridge B positioned as a leaf bridge, connecting only to one other bridge, A, with the remainder of its ports connected to endpoints.

In Figure 8(a), only one host has an entry shared between the two bridges. This mapping is obviously indeterminate. In Figure 8(b) the new node 2 might correspond to the querying node sending pings to node 1 while probing the FDBs. However, for the same reasons as Figure 5, this mapping is still indeterminate.

Now, suppose that node 1 has communicated with some other host on the network, for instance a nameserver. If that node is also found on any port of the upper bridge other than the port with 2 in its forwarding set, such as in Figure 8(c), then the mapping is determinate. In many cases, it is possible to force a machine to contact a nameserver by connecting to its FTP daemon, for instance.

In most cases, networks aren't designed with only one endpoint connected to a bridge. There is no reason to purchase a bridge in that case. Almost all bridges have 4, 16, 24, or even more ports used for connections to machines. A bridge with just two ports connecting machines to the network will satisfy Lemma 5.2. Consider Figure 8(d). The only entries in the FDBs are for endpoints on two different ports of bridge B and for the querying machine sending them pings. Networks (c) and (d) are two minimal examples—a single machine on a bridge that has talked to more than one other
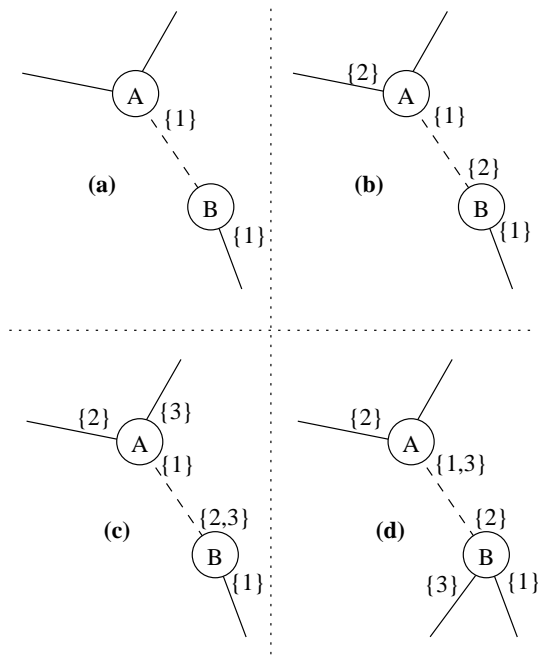
**Figure 8: These snapshots of two bridges within a network demonstrate different examples of indeterminate and determinate FDBs. The dashed line is the physical connection that the algorithm is attempting to determine. The forwarding sets are shown for each port. (a) and (b) give examples of indeterminate networks. (c) and (d) are both determinate, although for different reasons.**

machine (host, server, or router) or a bridge connected to two machines. Practically every part of an Ethernet will meet at least one of these criteria. However, because bridges are generally only installed in a network to be used, which means there are machines divided among several ports, most components will provide information well beyond the minimal requirements.

### 5.3  Specialization for traversal

An important specialization of Lemma 5.2 can be used in most cases. If the network's topology is determined by traversal from a designated root, then the first mapping step resolves the ports forming simple connections between each bridge and the root bridge. In a sense, this step divides the bridges into sets, each set containing the bridges found off one of the ports of the root bridge. It also identifies the port of each of the child bridges that forms a simple connection with the root bridge. From that point in the algorithm, the "root port" of each bridge is fixed. This knowledge can be exploited by the algorithm. In fact, being able to fix the root port of a bridge satisfies the first half of rule 3 in Lemma 5.2. This reduces the requirement for subsequent mappings to only one entry shared between the connecting port and another port on the child bridge.

This specialization is easy to exploit. The root is picked to maximize the chances that it meets the minimum knowledge requirement. The only complexity in the traversal is that each bridge must be tested to see if it is the next hop in the traversal, essentially trying to find the bridge directly connected to each port of the current root. Determining which bridge is the next hop is easy, because performing the mapping with an incorrect bridge will either indicate that there are bridges between itself and the root bridge (these bridges will be connected to the incorrect choice's root port), or it

will actually cause a conflict because a bridge will have to be connected to it on a port other than its root port, indicating that this isn't the true topology.

In summary, the minimum knowledge requirement is very easy to satisfy. Any bridge that is installed and used to connect more than one machine to the network will meet the requirement. Furthermore, even if the topology has little information, once the root port has been determined, only one entry is needed to meet the requirement.

## 6.  IMPLEMENTATION

The algorithm has been implemented in the Remos Bridge Collector and tested on a variety of networks. The implementation begins with the set of bridges on the network and downloads the entire FDB from each bridge. The actual topology derivation is done by traversal from a designated root bridge. After obtaining the FDBs, the root bridge is selected with the goal of maximizing the distribution of nodes across different ports.

The core operation of the algorithm is determining the port of the root to which each bridge is connected. The simple connection theorem is applied to each pair of bridges to determine the port of the root to which it is connected (and the port of the child bridge that is connected to the root) and the child is placed in a set connected to that port. The algorithm is then called recursively for the set of bridges attached to each port of the root. The bridge that is directly connected to the previous root bridge is found, and the mapping begins again. Shared segments are discussed in the following section.

After the bridge topology is derived, the endpoints in the network are mapped to their location in the network. That information is already present in the forwarding databases, and in general endpoint location is easily determined. In some situations, incomplete databases make it difficult to determine an endpoint's exact location, but pinging endpoints that are not located using the FDBs obtained earlier and querying the bridges where more information is needed has always provided the required information to locate a functioning endpoint.

### 6.1  Virtual switches

In Section 4.1, we described two situations where a direct connection could not be established between two bridges in an otherwise complete network. The principal challenge in these situations is the creation of a shared Ethernet segment, rather than the modern point-to-point connections. These situations are:

1. A hub is used to connect two bridges with other hosts or bridges. Because hubs do not participate in the bridging algorithm, this creates a shared network segment between the bridges.

2. A bridge exists that the algorithm either was not informed about, or to which SNMP access is denied. Because SNMP security generally consists of a simple list of allowed or denied IP addresses, this situation can easily occur.

The direct connection theorem required a special case to handle these possibilities. However, using the simple connection theorem no special case is required. This is because the theorem is used to map bridges to the port to which they are connected, rather than finding direct connections. In the course of the traversal, at each step the algorithm selects the next bridge in the tree. In the ideal case, a next hop will be found and will have no other bridges along its root port, therefore it is directly connected to the previous root. A virtual switch corresponding to one of the above conditions is inserted when no single bridge is found to be the next hop. Instead,

all bridges map at least one other bridge or host along their root ports, without causing any conflicts that would indicate that the previous root choice was incorrect.

To determine which bridges are connected directly to the virtual switch and which are their children, take the intersection of the nodes that each bridge maps to its root port. If a node appears in this set, it indicates that every bridge believes that node is between itself and the root bridge. The only way this situation can occur is for a shared segment to exist that connects multiple nodes to the root bridge via a broadcast medium. The nodes that do not appear in the intersection set are on the other side of one of the bridges connected to the shared segment, and the traversal is continued for each of the bridges connected to the shared segment to resolve the topology connecting these remaining nodes.

The elegance of this solution is one of the appealing aspects of this technique. Rather than requiring a special case, virtual switches are naturally determined using the base algorithm. In fact, the intersection set is calculated as the algorithm searches for the next bridge, so when no directly connected bridge is found, the nodes involved in the shared segment are already determined.

# 7. REAL WORLD EXPERIENCE

Practical experience has shown that the algorithms work as expected when used on a bridged Ethernet with bridges that properly support the BRIDGE-MIB. While these standards covers a large percentage of deployed networks, there are a number of networks that require somewhat modified approaches.

## 7.1 Administrative access

The administrative complication is primarily accessibility. Typically, SNMP access is only allowed from machines on the local network, and it is usually impossible to make SNMP queries to network components on an ISP's network. Security and privacy are the two primary reasons for this. Security is actually a technical concern; because the original designers of SNMP were unable to agree on a workable security protocol, there is little security in many implementations, therefore a minimal security level is achieved by restricting access to local hosts. ISP's are generally concerned about privacy, not wishing to divulge information about the congestion levels of their services. Furthermore, because SNMP queries can be expensive, no one wants to open their network up to excessive load or even denial-of-service attacks with SNMP. Remos combines network-based data with benchmark-based data to provide performance predictions in environments where direct network queries are only available for portions of the network.

## 7.2 SNMP compatibility

Although RFCs describe the behavior of SNMP implementations, the standards and their implementations have not resulted in the different manufacturers providing consistent interfaces. For instance, the forwarding databases in Ethernet bridges are particularly troublesome. Some allow queries to be made for the forwarding port of a specific address. Other implementations are designed only for traversal, requiring the same query to be reformulated as a query for the subsequent entry from the numerically preceding address. Furthermore, some bridges remove the forwarding database if queries are made to it too rapidly, apparently as a security measure.

Additionally, some SNMP implementations are non-compliant. Two Xylan bridges failed to correctly implement the BRIDGE-MIB's port to interface table, providing numbers that were incorrect. While a patch was easily applied to the Bridge Collector that allowed this table to be replaced with a configuration file, discovering these incompatibilities takes time and a thorough understanding of how to interpret the data in the MIB.

## 7.3 VLANs

In theory, VLANs should be somewhat trivial to manage, as each can be treated as a separate bridged Ethernet. In the Remos framework, the connection between VLANs would likely be addressed by other components than the Bridge Collector. In practice, differences in how VLANs are expressed in SNMP MIBs make topology discovery of VLANs more challenging for the Bridge Collector.

A VLAN constructed using Cisco switches required the VLAN number to be appended to the community name to obtain forwarding information about that particular VLAN. A VLAN constructed with the Xylan switches, however, did not result in any observable change in MIB-2 or the BRIDGE-MIB compared to the same network constructed with a single IP network without VLANs. It is unclear how this switch might report trunked VLAN links, as none were present in that network

Both trunked VLAN links and running multiple IP subnets over the same bridged Ethernet raise the issue of addressing the relationship between physical and logical topology. Remos does not currently address this well. The Bridge Collector can determine the topology of a single bridged Ethernet with multiple IP subnets, as the IP subnets do not affect the bridging protocol, but making use of this information for network management or resource prediction purposes requires more work. The Lucent group addressed some challenging aspects of this problem.

## 7.4 Hubs and dumb switches

Ideally, Ethernet networks are constructed with each desktop connected to a high-end switch, which is connected to a backbone with gigabit bandwidth. In reality, financial constraints and legacy hardware typically provide a lower-capacity network. As in Figure 1, hubs and shared segments are frequently used to connect several machines to a single switched port, and might typically be deployed in an office with multiple desktops. Hubs and shared segments used in this manner do not present any problems to the topology discovery algorithm. Because they are typically used in a limited physical space, such as an office, they do not present any network management difficulties. From a performance measurement perspective, they are easily dealt with because, by virtue of the broadcast medium, the bridge's port connected to the shared segment receives all data transmitted on that segment and provides a simple measurement of utilization.

Dumb switches, which are Ethernet switches that do not speak SNMP, present a different set of problems. They are similarly indetectable, as they do not speak SNMP and do not have an IP address. However, unlike hubs, because they do not form a shared broadcast medium, utilization measurement is more difficult, as the closest SNMP-enabled switch only sees traffic between the hosts and the rest of the network. If truly end-to-end utilization measurements are needed, then the necessary measurement can be obtained from the hosts themselves, if SNMP-enabled. From a network management perspective, dumb switches are not necessarily more difficult than hubs. However, the increasing availability of cheap 24-port dumb switches may result in a number of hierarchical networks of dumb switches, presenting significant problems for both network management and utilization measurement.

The presence of a hub, dumb switch, or an intelligent switch to which SNMP access could not be obtained in the internal portion of the network—between other SNMP-enabled switches—may present a more serious challenge. Fortunately, if only a single switch is missed, all links can be successfully monitored from their

other end. If adjacent switches are missed, however, their absence becomes more significant. In general, the presence of a shared segment in the interior of a network should be regarded with suspicion.

## 7.5 Dynamic networks

The topology of an Ethernet is nearly always regarded as being static—machines are rarely moved from one location to another. However, as the size of a network grows and as the Bridge Collector is left running longer, machines moving across the network must be accounted for. Wireless networks such as WaveLAN make dynamic topology updates essential, as mobile hosts may change their location, and corresponding base station, quite rapidly.

Fortunately, verifying the location of a host is as simple as checking the leaf bridge at which it was last found and verifying its presence. In the event that a node moves, sending a ping to it and then tracing it to its new location has only a minimal cost. The frequency at which nodes' locations are verified can be adjusted according to the frequency at which nodes move and the accuracy desired. A logical extension would be to identify mobile hosts and verify their location at a much higher rate than wired hosts.

Rearrangement of bridges is somewhat harder to detect, requiring the verification of several entries. However, changes in bridge topology are expected extremely rarely, so the slight added expense of verifying their location is not significant. When a bridge is added, deleted, or moves the topology discovery algorithm is simply rerun.

Finally, movement of a host during the discovery phase, when the bridges' forwarding databases are being downloaded, can cause conflicts that prevent the topology from being discovered. Our current approach is to simply rerun the discovery phase when such a conflict is found. However, in an environment with many mobile nodes, it may be necessary to improve the discovery phase by verifying the entries for a particular node across all of the bridges' forwarding databases to ensure that the node did not move during discovery.

## 8. RESULTS

The topology discovery algorithm has been implemented and tested in a variety of networks. The largest network was the CMU CS Department network, which contains almost 2000 hosts and 50 bridges. It has also been tested on departmental networks at BBN, ETH Zurich, and William and Mary, as well as on testbed networks at CMU, William and Mary, S/TDC, and two labs at NSWC as part of the HiPer-D project. The hardware used to implement these networks has included bridges from Cisco, Intel, 3Com, Asanté, Netgear, Xylan, Linksys, and others. Two of the testbeds used different types of VLAN technology, and one testbed also utilized ATM LAN emulation. In all cases, the algorithm and implementation were able to correctly determine the topology of the Ethernet LAN.

Although correctness of the topology is the most important criterion when judging the performance of this algorithm, its time performance may also be important in some applications. The majority of the execution time is spent downloading the FDBs from the bridges. This is partially due to the slow speed of SNMP but most significantly due to a desire not to swamp the bridges with queries. As java implementations have improved, the time to determine the topology of large networks has dropped significantly. Figure 9 shows the time required to calculate the topologies for a network with varying numbers of hosts and bridges included. Because the topology of bridged networks changes rather slowly, this performance is quite acceptable. Furthermore, detecting changes in the topology and updating the location of endpoints, which may
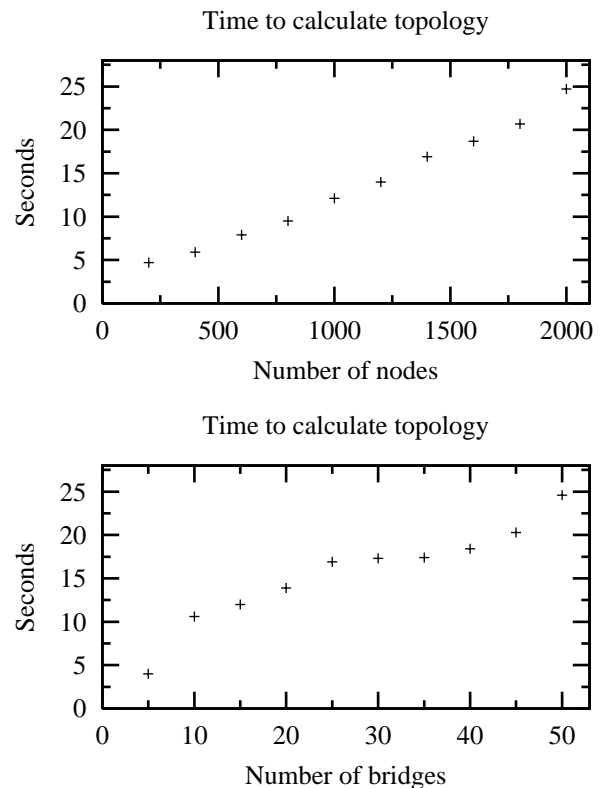


**Figure 9: The performance of the topology discovery algorithm running on a 566Mhz Pentium III. The number of hosts and bridges were varied by removing the entries for hosts or bridges from a previously collected data file.**

move more often than bridges, can be done without recomputing the entire topology.

Figure 10 shows the bridge topology discovered in the CMU CS Department network. This topology was verified to the extent possible as correct by the departmental network manager. There was no actual record of where hosts are attached to the network, but we discovered no errors in the placement of the machines we verified.

### 8.1 Practical considerations

Our research has demonstrated that SNMP, already supported by almost all of the current networking infrastructure, is sufficient for obtaining the information needed to determine topology and predict performance directly from the most commonly-used networks used today. Although it is not an ideal interface for this purpose, it allows the network-based approach to performance prediction to be explored and utilized on existing networks. Demonstrating the value of this approach by using it in real systems and applications should result in the development of more appropriate interfaces for network components. However, both administrative and technical considerations must be addressed to provide a better interface for performance prediction purposes.

One of the motivations for this work is the lack of standardization between the various bridge vendors on support for topology discovery. While both Cisco and Intel, two major bridge vendors, support topology discovery, they use incompatible proprietary techniques for doing so. In September 2000, the IETF adopted the Physical Topology MIB as a standard MIB [2]. While an encouraging devel-
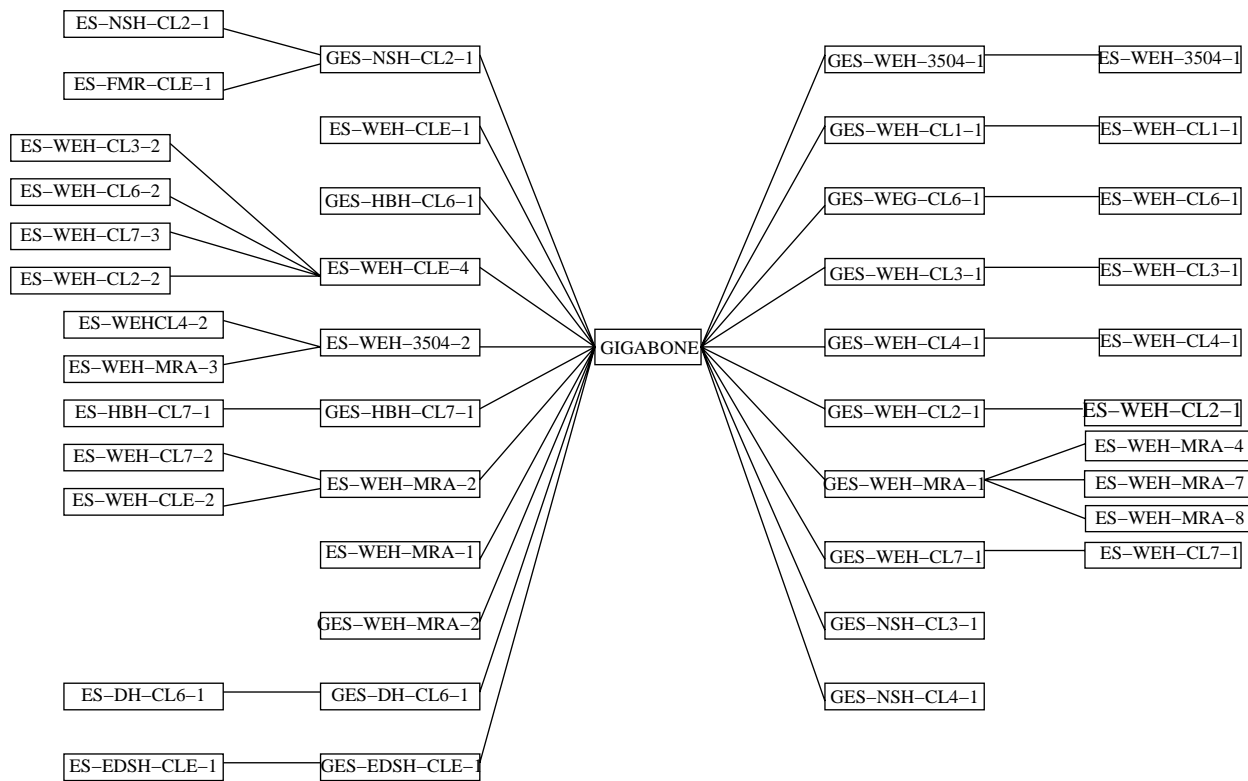
**Figure 10: Topology of the CMU CS Department primary bridged Ethernet on May 29, 2000. 1831 endpoints are located on this topology and omitted for clarity.**

opment, the earlier components of the RFC that specified a protocol to be used for topology discovery were removed from this RFC, and the current RFC does not impose an Internet standard for a discovery protocol—it merely reserves this portion of the MIB space. These shortcomings make it unlikely that automatic topology discovery will soon be a portable standard in commodity components.

## 9.  CONCLUSIONS

We have described an algorithm that can derive the topology used in a bridged Ethernet network using only information available from standard SNMP MIBs, while performing queries from a single machine. The primary contribution of the algorithms described in this paper is the ability to perform accurate topology discovery using knowledge of only a few endpoints, whereas previously knowledge of the entire network was required from the bridges. The minimum knowledge requirement allows topology discovery to be performed on very large networks with many bridges. The combined challenges of forcing the bridges to learn forwarding information for all of the bridges and then obtaining that information out of the bridges using sometimes unreliable implementations of the Bridge MIB were previously a major impediment to performing automatic discovery of Ethernet topology.

The information obtained through topology discovery is useful to both network managers and to the developers and users of parallel applications. Our implementation of the algorithm is currently the easiest way our network manager has to locate machines that have been inappropriately moved on the network. We have also used the information obtained through this program as the first step in obtaining link-level performance prediction used for scheduling applications in distributed environments. We hope that the useful-

ness of the topology information will encourage the development of a true standard protocol for topology discovery.

The implementation is available for download as part of the Remos system, available at:

`http://www.cs.cmu.edu/~remos/`

## 10.  ACKNOWLEDGEMENTS

## 11.  REFERENCES

[1] I. Ahmad, Y. Kwok, and M. Wu. Analysis, Evaluation, and Comparison of Algorithms for Scheduling Task Graphs on Parallel Processors. In *Proceedings of the Second International Symposium on Parallel Architectures, Algorithms, and Networks*, pages 207–213, June 1996.

[2] A. Bierman and K. Jones. Physical topology MIB. RFC2922, September 2000.

[3] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz. Topology discovery in heterogeneous IP networks. In *Proceedings of INFOCOM 2000*, March 2000.

[4] CAIDA. Skitter. http://www.caida.org/TOOLS/measurement/skitter/.

[5] P. E. Crandall and M. J. Quinn. A partitioning advisory system for networked data-parallel processing. *Concurrency: Practice and Experience*, 7(5):479–495, August 1995.

[6] P. A. Dinda and D. R. O'Hallaron. An evaluation of linear models for host load prediction. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, August 1999.

[7] S. Figueira and F. Berman. Modeling the effects of contention on the performance of heterogeneous applications. In *Proceedings of the Fifth IEEE International Symposium on High Performance Distributed Computing (HPDC 5)*, pages 392–401, Syracuse, NY, August 1996.

[8] R. Govindan and H. Tangmunarunkit. Heuristics for internet map discovery. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[9] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang. On the placement of internet instrumentation. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[10] P. K. K. Loh, W. J. Hsu, C. Wentong, and N. Sriskanthan. How Network Topology Affects Dynamic Load Balancing. *IEEE Parallel and Distributed Technology*, pages 25–35, Fall 1996.

[11] B. Lowekamp and A. Beguelin. ECO: Efficient collective operations for communication on heterogeneous networks. In *Proceedings of the 10th International Parallel Processing Symposium (IPPS'96)*, pages 399–405. IEEE, April 1996.

[12] B. Lowekamp, N. Miller, D. Sutherland, T. Gross, P. Steenkiste, and J. Subhlok. A resource query interface for network-aware applications. *Cluster Computing*, 2(2):139–151, 1999.

[13] B. Lowekamp, D. O'Hallaron, and T. Gross. Direct queries for discovering network resource properties in a distributed environment. In *Proceedings of the 8th IEEE International Symposium on High Performance Distributed Computing (HPDC)*, pages 38–46. IEEE Computer Society, August 1999.

[14] N. Miller and P. Steenkiste. Collecting network status information for network-aware applications. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.

[15] K. Obraczka and G. Gheorghiu. The performance of a service for network-aware applications. In *Proceedings of the ACM Sigmetrics SPDT'98*, 1998.

[16] G. Shao, F. Berman, and R. Wolski. Using effective network views to promote distributed application performance. In *Proceedings of the 1999 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'99)*, 1999.

[17] R. Siamwalla, R. Sharma, and S. Keshav. Discovering internet topology. http://www.cs.cornell.edu/skeshav/papers/discovery.pdf, July 1998.

[18] W. Theilmann and K. Rothermel. Dynamic distance maps of the internet. In *IEEE INFOCOM 2000*, Tel Aviv, Israel, March 2000.