Layer-2 Path Discovery Using Spanning Tree MIBs

David T. Stott

Avaya Labs Research, Avaya Inc.

233 Mount Airy Road

Basking Ridge, NJ 07920

stott@research.avayalabs.com

March 7, 2002

Abstract

Layer-2 network topology discovery has become critical for areas of multimedia network management. For example, the topology can be used to help locate areas of congestion or to find the path between a pair of hosts (e.g., a pair experiencing poor VoIP performance). Little work is available for automatically finding the layer-2 topology of such networks. This paper presents an approach to find the topology based on tables for the spanning tree algorithm available through SNMP. The approach is being used as part of Avaya Inc.'s ExamiNet™, a tool for assessing IP telephony readiness in customer networks, and has been run on an example enterprise network.

1 Introduction

Knowledge of network topology including the path between endpoints, can play an important role in analyzing, engineering, and visualizing network performance. For example, ExamiNet™ [1] collects

performance variables from routing devices and end-to-end application measurements and needs the topology to identify which performance variables applly to interfaces on the path. This paper describes an algorithm for finding layer-2 paths using SNMP to collect information about switches' spanning trees.

Not long ago, the standard network layout used a separate switched network for each department and geographical location (e.g., a floor and wing of a building) and several layer-3 routers between the switched networks. The recent popularity of VLANs (virtual LANS) has resulted in an increase in the size of fast switched networks and a decrease in the importance of routers. Today, it is common to use a single switched network for an entire building or campus with a single edge-router for each switched network. This shift underscores the importance of the layer-2 topology in enterprise networks.

A solution to discovering the topology at layer-2 involves the spanning tree algorithm. Ethernet networks may fail if there is a loop in the switching devices. Loops may be useful for fault tolerance, however, by providing redundant paths. To allow loops in the network, the switches run a spanning tree algorithm between one another to identify any loops and block links to create a logic tree connecting the switches. The most common algorithm used is the IEEE 802.1D Spanning Tree Algorithm Protocol [2]. Some vendors use a variation of this algorithm to support using separate spanning trees for each VLAN (e.g., Avaya Inc.'s Cajun switches).

Simple Network Management Protocol (SNMP) [3] is a commonly used standard protocol for collecting network management information. SNMP works by running an SNMP agent on each network device (nearly all new network devices are delivered with an SNMP agent). A MIB (Management Information Base) [4] specifies a set of well-known objects that the client and agent upon. A client program sends SNMP queries to the agent to request a set of objects, and the agent replies with the values of the objects. SNMP can be used to learn information about the spanning trees from switches.

Topology is defined as set of devices and the connections between them along with the set of paths between endpoints. The layout of the network can be represented as a graph $G = (D, L)$ where D is a set of devices and L is a set of links. Let $I_{i,j}$ denote the jth interface of device D_i in D. Each link in L

is defined as a pair of interfaces that are connected by a direct communication link. A *path* is defined as the sequence of interfaces on devices (in and out) used to send data from host A to B. The path can be denoted in three ways, (a) a list of the interfaces, (b) a list of tuples where each device in the path has a tuple, (device, ingress interface, egress interface), or (c) a list of links where each link is represented by the interface pair $(I_{i,j}, I_{k,l})$ where $I_{i,j}$ is the egress interface, and $I_{k,l}$ is the ingress interface.

Figure 1 illustrates a small network consisting of three switches (S_1, S_2, and S_3) and two hosts (A and B). The switch interfaces that the packets traverse are also marked on the figure. For the network, $D = \{S_1, S_2, S_3, A, B\}, I = \{I_{A,0}, I_{1,1}, I_{1,2}, I_{2,1}, ...\}$, and $L = \{(I_{A,0}, I_{1,1}), (I_{1,2}, I_{2,1}), (I_{2,2}, I_{3,1}), (I_{3,2}, I_{B,0})\}$. The path from A to B is $I_{A,0}, I_{1,1}, I_{1,2}, I_{2,1}, I_{2,2}, I_{3,1}, I_{3,2}$, and $I_{B,0}$.

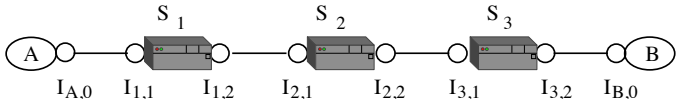


Figure 1: Example Path

It often makes sense to look at the topology or path at a particular network layer. For example, the *layer-2 topology* is the topology whose set of links are restricted to layer-2 links. The *layer-2 path* is the path containing only layer-2 devices between hosts that are directly connected at layer 3 (i.e., there is no layer-3 device in the path between the hosts). The *layer-3 path* is the path containing only layer-3 devices (e.g., hosts and routers) and only layer-3 links (i.e., a direct communication link at layer-3, which may involve a number of switches). And, the *layer-3 topology* is the topology whose links are layer-3 links (e.g., a number of layer-2 switches may be involved, but the network address does not change).

This paper describes related work in Section 2. Next, it provides background information on the spanning tree algorithm in Section 3. Next, it presents the topology discovery algorithm in Section 4. Then, it concludes with a summary in Section 5.

2 Related Work

Several approaches to finding layer-3 topologies have been proposed (e.g., [5, 6, 7, 8, 9]). Despite the importance of the layer-2 topology, little literature is available. One related approach [10] uses pattern matching on interface counters available through SNMP. Another approach to generate the layer-2 topology between switches was presented in [11] and improved upon in [12]. This approach operates by processing the forwarding tables obtained from each switch via SNMP. Both approaches start by collecting the forwarding tables from each switch in the network. The forwarding table caches entries for each physical address that associate the address with the port toward the host using the physical address. When a packet arrives at the switch, it looks up the destination address in the forwarding table to find the port it should use to forward the packet.

The original approach [11] assumes that all physical addresses are cached in every switch (this condition can be forced by sending ping messages between each pair of hosts). They define S_{ij} as the jth interface of the ith switch and A_{ij} as the set of physical address in the forwarding table for S_{ij}. They prove that two ports on two switches (S_{ij} and S_{kl}) are connected if and only if the union of A_{ij} and A_{kl} is the set of all physical addresses used and their $A_{ij} \cap A_{kl} = \emptyset$. The topology is generated by applying this test to each pair of switch ports. The assumption that every address must be cached can be somewhat relaxed by requiring $A_{ij} \cup A_{kl}$ to cover a given fraction of the physical addresses rather than all addresses in the network.

The second approach [12] uses a slightly different test to relax the assumption that every address is cached. They define the *forwarding entries* for a switch, F_i^j, to be the same as A_{ij} above, and the *through set* to be the $T_i^j = \cup F_i^k | k \neq j$. They define a simple connection as a pair of switch ports that connects two switches, possible with another switch between them. They prove that two ports form a simple connection if and only if $T_a^x \cap T_b^y = \emptyset$. After building the set of simple connections a set of rules are applied to handle cases such as a simple connection appearing because neither switch has an address

in common. Then, the topology is determined examining the set of simple connections from a single switch port and choosing the one that can be a direct connection without introducing any conflict to the forwarding table.

Some switch vendors (e.g., [13]) have produced commercial tools that use proprietary MIB extensions to generate the layer-2 topology in a network of consisting only of their products. A few commercial tools (e.g., [14, 15]) have recently added claims to support provide layer-2 topology discovery in heterogeneous networks. Since they use proprietary techniques, the approaches cannot be discussed here.

3 Spanning Tree Algorithm

This section provides a brief description of how switched bridges operate, what the spanning tree algorithm is, and how it works. Ethernet bridges are used to forward packets between network interface cards (NICs) on the same subnet. When a switched bridge (as opposed to hubs which broadcast packets on each port) receives a packet, it looks up the destination physical address in its *forwarding table* to find the port to which it needs to forward the packet.

The technique switches use to build the forwarding table works as follows. Initially, the bridge's forwarding table is empty. When the bridge receives a packet on a port from a new host, it updates its forwarding table by adding an entry with the host's physical address and the port. The bridge can only forward unicast messages to the addresses in the forwarding table; but it can send broadcast packets to all ports. In a typical IP network, before sending a unicast message, each host broadcasts an ARP (address resolution protocol) message to the subnet to find the physical address for a given IP address. When the host with the given IP address responds with a unicast ARP reply to the original host, the switches between the two hosts learn the location of each of the hosts.

This approach works well when the bridges are connected in a tree (i.e., the topology is loop-free). Should the switch topology contain a loop, it is possible for a packet to be forwarded through the loop

indefinitely. In practice, it is common to have loops in the physical switch topology. One reason for a loop is to provide fault-tolerance–if there are multiple paths in the network, it may be possible to tolerate a failure to a single switch. With VLANs, redundant paths allow load sharing on a per-VLAN basis. That is, if a network has loops, with multiple spanning trees, the links would only block for some, rather than all VLANs.

To detect loops in the topology, bridges run a spanning tree algorithm. In graph theory, a spanning tree is a tree connecting all the nodes in the graph. In networking terms, the nodes are switches and the edges are links. The links in the spanning tree may forward packets, but the links not in the spanning tree are in a blocking state and may not forward packets (unicast or broadcast).

The most common algorithm used is the IEEE 802.1D Spanning Tree Algorithm Protocol [2]. It defines these terms:

- Bridge ID, is an 8-octet identifier consisting of a 2-octet priority followed by the lowest 6-octet physical address assigned to the bridge.
- Designated Root, is the Bridge ID of the root bridge seen on the port.
- Designated Bridge, is the Bridge ID of the bridge connected to a port (or its own Bridge ID).
- Designated Port, is the Port ID of a port on the Designated Bridge.
- Path Cost, the cost assigned to a link.
- Root Path Cost, the sum of the Path Costs along the path to the root bridge.

Each bridge records the values for Bridge ID, Designated Root, Designated Bridge, Designated Port, Path Cost, and Root Path Cost for each port in the Spanning Tree Port Table. The values are updated by exchanging messages with its neighbors. The messages allow each bridge to find (a) the root bridge, and (b) the shortest path (i.e., the lowest cost path) to the root. The messages include the bridge's Bridge ID

and port ID, and the Designated Root and Root Path Cost values it has learned thus far to each neighbor. Upon receiving a message, the bridge learns (a) of a new Bridge Root if the neighbor's Bridge Root has a lower Bridge ID than the current Bridge Root, or (b) of a better path to the Bridge Root if the new Root Path Cost is the shortest among the bridge's ports. When the bridge updates its Designated Root ID or shortest path to root, it sends another message to its neighbors. The protocol converges when all bridges have the same Bridge Root and the spanning tree includes the shortest paths from each bridge to the Bridge Root.

Figure 2 gives an example network with loops in the physical topology. The results of the spanning tree algorithm for this network are given in Table 1. The two links shown with dashed lines are blocking. Consider Bridge_3 in the example. The Designated Root is the Bridge ID of Bridge_1 for all ports.

Port_1 on Bridge_3 is in the spanning tree and connects to the Bridge_1 (which is Bridge Root), because the Designated Bridge is equal to Bridge_1's Bridge ID. Designated Port 8 on Bridge_1 corresponds to Port_1 on Bridge_1 because that port has a Designated Port equal to 8 (the first entry in Table 1). Port_2 is blocking because it has a higher cost than the Port_1. Its Designated Bridge and Port that of Bridge_2, port_4. The Designated Bridge and Port for Port_3 and Port_4 are Bridge_3's own Bridge ID and Port ID because the link goes away from the root. Port_3 connects to Bridge_4, port_2, a child in the spanning tree. Because the link is away from the root, this can only be seen by examining Bridge_4's entries (namely, the entry for port_2). Port_4 may connect to a host, but does not connect to any other bridge in the spanning tree.

The IEEE 802.1D Spanning Tree Algorithm lacks support for VLANs. To support VLANs, vendors have introduced variations of the algorithm to include a VLAN tag. To have load-balancing across VLANs, it is common to have different spanning trees per VLAN. This requires using a different root election procedure. Most variations on the 802.1D algorithm are proprietary and non-interoperable.

The *dot1dStpPortTable* in the Bridge-MIB gives the results from the Spanning Tree Algorithm. Table 2 gives excerpts from port tables from six switches. The Bridge ID (i.e., the last 6 octets from the

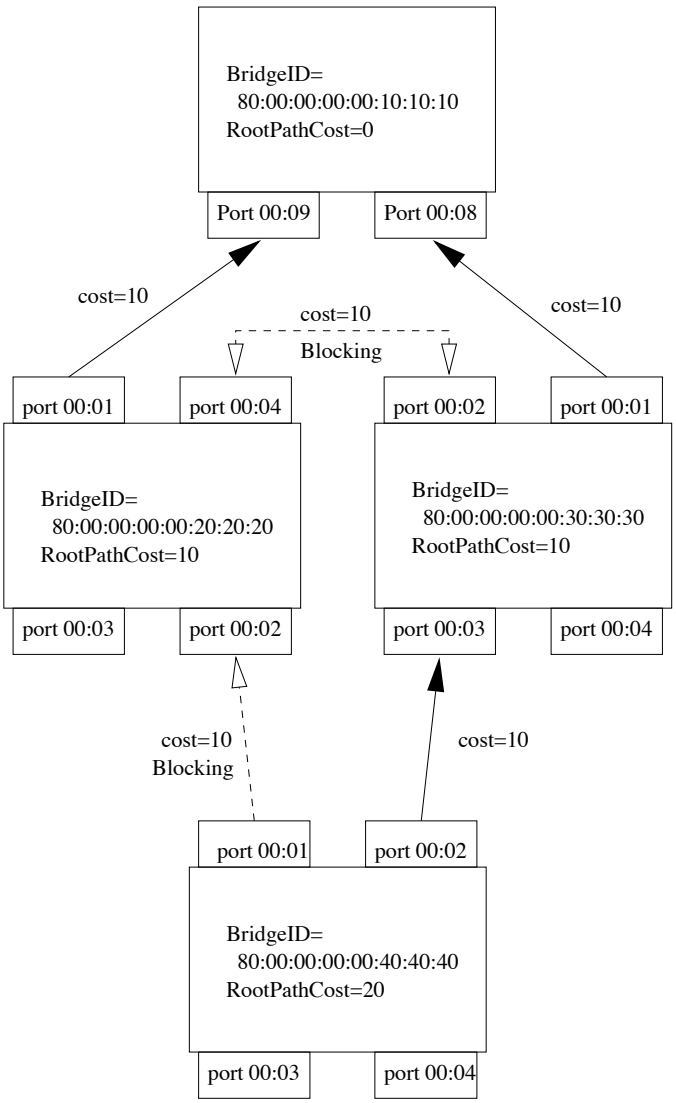


Figure 2: Example Network with Loops Physical Topology

dot1dBaseBridgeAddress variable) for each switch is included in the section headers. The first column, StpPort, identifies the entry's port (the *dot1dBasePortTable* can be used to map the port to its interface index, *ifIndex*). The next column, State, indicates if the port is: forwarding (part of the spanning tree), blocking (removed from the spanning tree to prevent a loop), disabled (the port is not in use or does not participate in the algorithm), building (a transient state while the algorithm is running), etc. The third column, DesignatedRoot, is the Bridge ID of the root of the spanning tree. The fourth column, DesignatedBridge is either the switches own Bridge ID or that of another switch directly connect to the

Table 1: Spanning Tree Results from Example Network

StpPort	State	Des. Root	Des. Bridge	Des. Port	Path Cost	Root Cost
Bridge 1 ID=80:00:00:00:00:10:10:10						
1	forwarding	...:10:10	...:10:10	8	10	0
2	forwarding	...:10:10	...:10:10	9	10	0
Bridge 2 ID=80:00:00:00:00:20:20:20						
1	forwarding	...:10:10	...:10:10	8	10	10
2	blocking	...:10:10	...:20:20	2	20	10
3	disabled	...:10:10	...:20:20	3	10	10
4	blocking	...:10:10	...:20:20	4	10	10
Bridge 3 ID=80:00:00:00:00:30:30:30						
1	forwarding	...:10:10	...:10:10	8	10	10
2	blocking	...:10:10	...:20:20	4	10	20
3	forwarding	...:10:10	...:30:30	3	10	10
4	forwarding	...:10:10	...:30:30	4	10	10
Bridge 4 ID=80:00:00:00:00:40:40:40						
1	blocking	...:10:10	...:20:20	2	20	30
2	forwarding	...:10:10	...:30:30	3	10	20
3	disabled	...:10:10	...:40:40	3	10	20
4	disabled	...:10:10	...:40:40	4	10	20

entry's port. The last column, DesignatedPort is the port ID of the DesignedBridge connected to the switch (that is, either the port ID of the entry's port on the same switch or the port ID on another switch connected to the entry's port). Several columns in the MIB table, such as PathCost and DesignatedCost, are not included in example because they are not used in this paper.

4 Topology Discovery Algorithm

This section describes the algorithm for determining the layer-2 topology from information available through SNMP. The primary goal is to find the path between two IP hosts including each SNMP-enabled device and interface in the order the packet takes. Thus, hubs or switches where SNMP is unavailable may be excluded from the topology. We initially restrict ourselves to common MIB objects though we expect to support some vendor-specific implementations. The algorithm will determine the exact topology for all SNMP-enabled switches that use standard MIB tables for the bridge information. The

Table 2: Sample dot1dStpPortTable Tables

Port	State	DesignatedRoot	DesignatedBridge	DesignatedPort
00:00:02:66:5E:80 switch_207				
8	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:66:5E:80	08:80
31	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:66:5E:80	1F:80
32	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:66:5E:80	20:80
73	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	39:80
00:00:02:6A:E4:80 switch_208				
10	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:6A:E4:80	0A:80
55	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:6A:E4:80	37:80
56	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:6A:E4:80	38:80
73	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:23:DF:80	39:80
00:00:02:66:47:80 switch_209				
1	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:66:47:80	01:80
47	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:66:47:80	2F:80
48	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:66:47:80	30:80
73	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	31:80
00:00:02:4A:58:80 switch_26				
8	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:4A:58:80	08:80
10	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:4A:58:80	0A:80
65	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:4A:58:80	41:80
73	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:23:DF:80	31:80
00:00:02:23:DF:80 switch_28				
5	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:23:DF:80	05:80
49	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:23:DF:80	31:80
57	forwarding	00:80:00:00:01:00:01:80	00:00:00:00:02:23:DF:80	39:80
91	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	5B:01
00:00:01:00:01:80 switch_29				
3	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	03:80
49	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	31:80
57	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	39:80
91	forwarding	00:80:00:00:01:00:01:80	00:80:00:00:01:00:01:80	5B:01

algorithm makes a few assumptions:

1. the network is in a valid configuration (e.g., there are no loops in the active topology),

2. all MIB tables from the switches are up-to-date (e.g., no topology changes occur while the algorithm is running, and there are no stale forwarding entries), and

3. switches use the Spanning Tree Algorithm.

The algorithm has three phases, data collection, direct link analysis, and endpoint analysis. The data collection phase includes (1) identifying all switches and routers, (2) collecting MIB tables pertaining to the spanning tree from the switches, and (3) collecting MIB tables from the router address translation. In the direct link analysis, the spanning tree is reconstructed from the collected tables. In the endpoint analysis, the switch directly connected to each host is discovered. To support VLANs, the direct link analysis and endpoint analysis phases must be repeated for each VLAN.

Because the switches are in a tree, it is easy to find the interfaces in the path between any two switches by traversing the tree between the switches. The endpoint analysis provides the switch and its interface that connects to each endpoint. Thus, it is easy to construct the path as the interface connected to the source endpoint, followed by those interfaces in the path between two switches, followed by the interface connected to the destination endpoint.

4.1 Data Collection Phase

Identify switches and routers: The first step is to identify the switches and routers in the network. This step is handled by the ExamiNet™discovery module [1]. The module scans a network by sending SNMP probe messages to every IP in a given range and collecting the responses from the network. Next, the module uses a heuristic to classify each device that responded to the probe as a router, switch, host, etc. based on the system OID or various MIB variables (e.g., *system.sysServices*, *ip.ipForwards*, *dot1d.bridgeType*). It also checks that each response is a valid IP address (i.e., not a broadcast or network address) and filters out responses where a single device responded to multiple IP addresses. In the current implementation, the results are stored in a database.

Collect MIB tables from switches: The Bridge-MIB [16] contains several objects describing the results of the Spanning Tree Algorithm. Table 3 lists the MIB objects collected from each switch. The contents of each table is described below.

Collect MIB tables from the router: Routers cache the mapping from physical address to IP address in

Table 3: MIB Objects for Bridges

MIB Object	Description
dot1dBridge.dot1dBase.dot1dBaseBridgeAddress	Bridge ID
dot1dBridge.dot1dBase.dot1dStpPortTable	results from Spanning Tree Algorithm
dot1dBridge.dot1dBase.dot1dBasePortTable	mapping from Bridge Port to ifIndex
dot1dBridge.dot1dTp.dot1dTpFdbTable	forwarding table
interfaces.ifTable	MIB-II interface table (per interface description and statistics)

the *ip.ipNetToMedia* MIB table. To send a packet to an endpoint, the router needs to learn this mapping (e.g., which it does through the ARP protocol in a traditional IP network). The mapping is stored in this table, and entries are flushed after a period of several minutes if no packets are sent to the device. Because the content of the table is temporal, the table may need to be read periodically to learn all mapping for every endpoint. Alternatively, packets could be sent to each IP address in the subnet to populate the ARP table shortly before collecting the MIB table.

4.2 Direct Link Analysis

The purpose of the direct link analysis is to reconstruct the spanning tree by finding all direct links between switches. The procedure examines MIB tables for each switch to identify the links between switches and the ports to each link.

For each entry in a switch's *dot1dStpPortTable* for which the Designated Bridge is different from the switch's own Bridge ID there is a direct link between the switch and the one identified by the Designated Bridge. If the link state for the entry is *forwarding*, the link is in the spanning tree. The entry contains the port number for the original switch and identifies the port on the neighboring switch as the Designated Port. Thus, each such entry completely describes the link between the two devices.

The direct link analysis algorithm is based on this principle. The algorithm scans each switch's *dot1dStpPortTable* for entries where the Designated Bridge is different than the switch's own Bridge ID and the link is in the forwarding state (as opposed to the blocking state). Of the two switches on

the link, the switch that produced such an entry is the one further from the bridge root in the spanning tree. The *StpPort* field gives the egress port (the switch's port table converts the port to an *ifIndex*). The *DesignatedBridge* field identifies the neighboring device, though the Designated Bridge ID needs to be converted to an IP address for our analysis. And, the *DesignatedPort* field identifies the port on the neighboring device, though switch port needs to be converted to an *ifIndex*.

The IP address of the neighboring switch can be determined from the Bridge ID. If the Bridge Address (the Bridge ID with the two-octet priority removed) matches any switch's bridge address table *dot1dBridge.dot1dBase.dot1dBaseBridgeAddress* MIB object, the switch that supplied the variable is the neighbor. If the neighboring switch does not use the bridge address MIB object, one approach is to find the Bridge Address in a switch's interface table. Another approach is to scan the switches' forwarding tables for an entry that matches the Bridge Address and whose *dot1dTpFdbStatus* column is *self* (meaning that the address belongs to the switch itself). Another approach is to check the router's *ip.ipNetToMedia* table to translate the Bridge Address to an IP address. In practice, at least one of these approaches will be successful if the switch is SNMP-enabled. For the results in this paper the approaches were attempted in the order they are presented.

Though most per-interface MIB objects use the *ifIndex* object to identify an interface, the Bridge MIB uses the bridge port ID (the ID used in the layer-2 messages). Converting the bridge port to an *ifIndex* should require only a lookup into the neighboring switch's *dot1dBridge.dot1dBase.dot1dBasePortTable*. The difficulty is that this table uses an integer representation for each number while the *dot1dStpPortTable* uses an ASN.1 [17] 'OCTET STRING' data type (which is a pair of 8 bit characters). In some vendor's implementations the conversion from the octet string to integer is more complicated than treating the string as a 16 bit integer. For example, one vendor sets the highest bit in octet string. Another vendor reverses the byte over after setting the highest bit. For example, a bridge ID of 10 could be represented as `00:0a`, `80:0a`, or `0a:80` depending on the vendor. Table 2 uses the third format. A more reliable approach is to build a bridge port to *ifIndex* conversion table from the *dot1dStpPortTable*. For each entry

in the *dot1dStpPortTable* where the Designated Bridge is the same as the device's Bridge ID, the Designated Port is the octet string representation of the port and the *dot1dStpPort* (the first column in Table 2) is the integer representation of the bridge port that is the same as the one used in *dot1dBasePortTable* table.

Figure 3 shows the topology for the example *dot1dStpPortTable* objects from Table 2. Each link in the spanning tree is marked with an arrow pointing toward the root of the spanning tree (switch_29). The last entry for each of the first five switches has a different Designated Bridge from it switch's bridge address. The algorithm identifies these entries as representing a link in the spanning tree. The sixth switch has no such entries because it is the root of the tree.

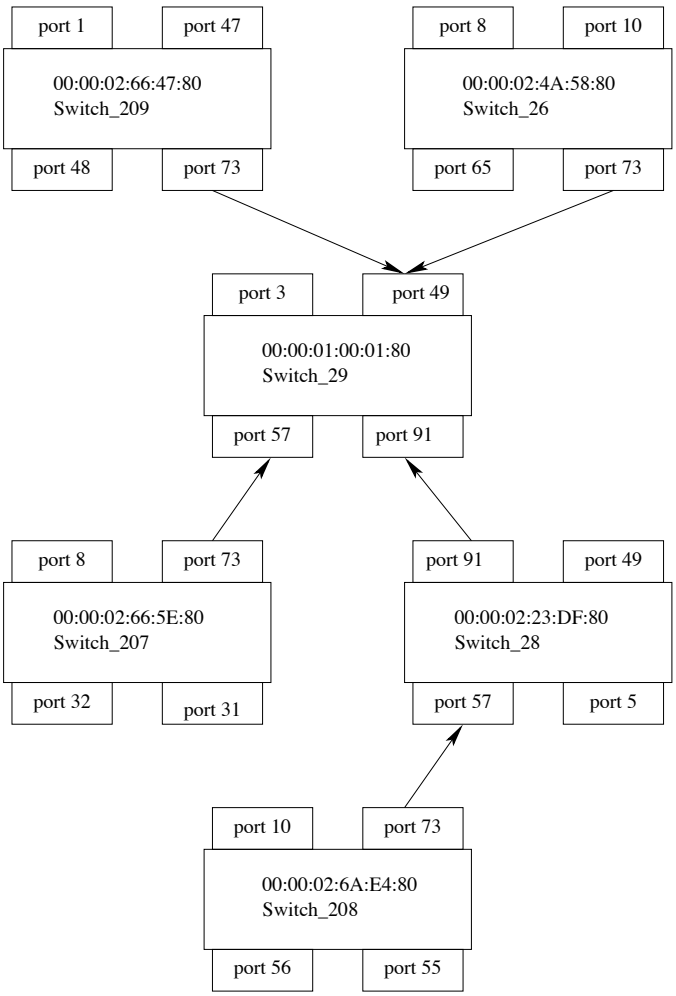


Figure 3: Example Switch Topology

The first switch, switch_207 has an entry for a trunk link on port 73. The neighboring switch is switch_29 because the switch_29's bridge address (00:00:01:00:01:80) matches the entry's Designated Bridge field. The port on switch_29 is 57 because the entry's Designated Port field (`39:80`) matches that of switch_29's entry for port 57. Similarly, port 73 on switch_208 connects to port 54 on switch_28. Port 73 on switch_209 connects to port 49 on switch_29. Port 73 on switch_26 connects to port 49 on switch_28. And, port 91 on switch_28 connects to port 91 on switch_29. Note that switch_26 and switch_209 connect to the same port in the spanning tree; this is because the three switches are connected by a hub or a transparent switch that does not participate in the spanning tree algorithm.

The *dot1dStpPortTable* table must be used to convert the port number to an *ifIndex* value. In this example, the vendor uses the same value for the port and *ifIndex*.

4.3 Endpoint Analysis

Two devices are said to be *directly connected* when the path between them does not include any other device in the spanning tree. A device, D_t, is directly connected to $I_{s,i}$, the ith port on device D_s, when D_t and D_s are directly connected and the path between them uses $I_{s,i}$. A *switch trunk* is a link between two directly connected switches. The switch ports to such a link are considered on the trunk.

A switch's forwarding table includes entries with a physical address and the port number the switch will use to forward packets to the device addressed by the physical address, M. We define $F(s, M)$ as the port, $I_{s,i}$, where the switch, D_s, that has a forwarding entry for M on that port or $\emptyset$ where there is no such entry in D_s's forwarding table. In a valid network configuration, each non-empty $F(s, M)$ is (a) the switch trunk from D_s in the spanning tree one hop closer to the host addressed by M, (b) a port directly connected to the host, or (c) both. The third case is when the trunk and the link to the host are shared or connected to a switch that does not participate in the spanning tree algorithm.

Lemma 4.1 provides a simple, efficient sufficiency test to discover the port directly connected to a device. That is, if any switch has an entry on a non-trunk interface for a physical address used by the

host, it must be directly connected to that port. In practice this sufficiency condition generally holds because switch trunks are seldom shared with hosts. When this condition does not hold, an approach using the spanning tree and the forwarding tables can be used.

Lemma 4.1 *If switch D_s has a forwarding entry for physical address M on port $I_{s,i}$ that is not on a switch trunk, the device addressed by M is directly connected to the switch.*

Proof By the definition of the switch trunk, D_s cannot connect to another switch. Thus, there can be no device in the spanning tree connected through $I_{s,i}$. Therefore, by definition, device addressed by M is directly connected to D_s on $I_{s,i}$. ■

Note that the inverse of Lemma 4.1 is not necessarily true. Two switches, D_s and D_t, and a host, D_h, could all be connected via a hub that does not participate in the spanning tree algorithm. In this case, the ports on each switch to the hub, $I_{s,i}$ and $I_{t,j}$, are directly connected because the path between them does not involve any other switch in the spanning tree. Since D_h is also directly connected to S_s and S_t, it is possible for a host to be directly connected to a switch via a switch trunk.

The first endpoint analysis approach is to apply Lemma 4.1. Since it is only a sufficiency test, a second approach will be needed to handle cases not covered by the first. Before describing the second approach, two important properties (Property 4.1 and Property 4.2) about the spanning tree and the forwarding tables should be noted. These properties (along with the lemma) can be used to define a region where the host may be located. The region is the minimal region that can be found using the spanning tree and the forwarding tables. Every switch with forwarding entries for M in the region must be on the boundary of the region. Thus, there can be no switch in the interior of the region with any information about the location of the host in its forwarding table. Hence, it is possible for the host to be located on any switch or link within the region.

Property 4.1 *If switch D_s has a valid forwarding entry $F(s, M)$ on a forward edge of the spanning tree (i.e., an edge away from the root), the host must be located in the subtree rooted at $I_{s,i} = F(s, M)$.*

```
LOCATE_HOST(S_root, M)
 1  Location ← NIL
 2  Region_Root ← NIL
 3  Region_Edge_List ← NIL
 4  if LOCATE_HOST_HELPER(S_root, M) ≠ NIL then
 5     OUTPUT("Host M is connected to switch
 6                Location.switch on port Location.port")
 7
 8  else
 9     OUTPUT("Host M is located in the region rooted at switch
10                Region_Root.switch port Region_Root.port")
11     for each (s, p) in Region_Edge_List do
12        OUTPUT( "The region is bounded by switch s port p" )

LOCATE_HOST_HELPER(S_cur, M)
 1  P ← F(S_cur, M)
 2  switch
 3     case IS_NOT_TRUNK(S_cur, P) :
 4          Location.switch ← S_cur
 5          Location.port ← P
 6          return TRUE
 7     case IS_TRUNK(P) and IS_FORWARD_EDGE(S_cur, P) :
 8          Region_Edge_List ← NIL
 9          Region_Root.switch ← S_cur
10          Region_Root.port ← P
11          for each S_next in descendant of S_cur on P do
12             if LOCATE_HOST_HELPER(S_next, M) then
13                return TRUE
14          return TRUE
15     case IS_TRUNK(P) and IS_BACKEDGE(S_cur, P) :
16          APPEND(Region_Edge_List, (S_cur, P))
17     case P = NIL :
18          for each S_next in descendant of S_cur do
19             if LOCATE_HOST_HELPER(S_next, M) then
20                return TRUE
21  return FALSE
```

Figure 4: Pseudo Code for Locating a Host

Property 4.2 *If switch D_s has a valid forwarding entry $F(s, M)$ on a back edge of the spanning tree (i.e., an edge toward the root), the host cannot be below D_s in the spanning tree.*

A tree search, such as the one in Figure 4, can be used to find the exact location of the host (i.e.,

its directly connected switch port(s)) or the region where the host could potentially be located. In this traversal, the forwarding port $F(s, M)$ for each switch can be in one of four cases:

1. If $F(s, M)$ is not-trunk port, the host must be directly connected to the port (by Lemma 4.1). Since the host has been located, the algorithm returns true to indicate that search is successful.

2. If $F(s, M)$ is a port on a forward link (one away from the root), the host must be located somewhere in the branch of the spanning tree under $I_{s,i} = F(s, M)$ (by Property 4.1). The algorithm marks this port as the root of the region where the host may be located. It needs to recursively check each of these branches. Then, the algorithm returns true to indicate that the search is complete, since the host must be located below D_s.

3. If $F(s, M)$ is a back edge (i.e., a link *toward* the root), the host cannot be located below $I_{s,i}$ (by Property 4.2). Hence, $I_{s,i}$ is the edge of the region where the host may be located, unless a forward edge is found later.

4. If $F(s, M) = \emptyset$, the search continues to each descendant of D_s as if they where directly connected to D_s's parent.

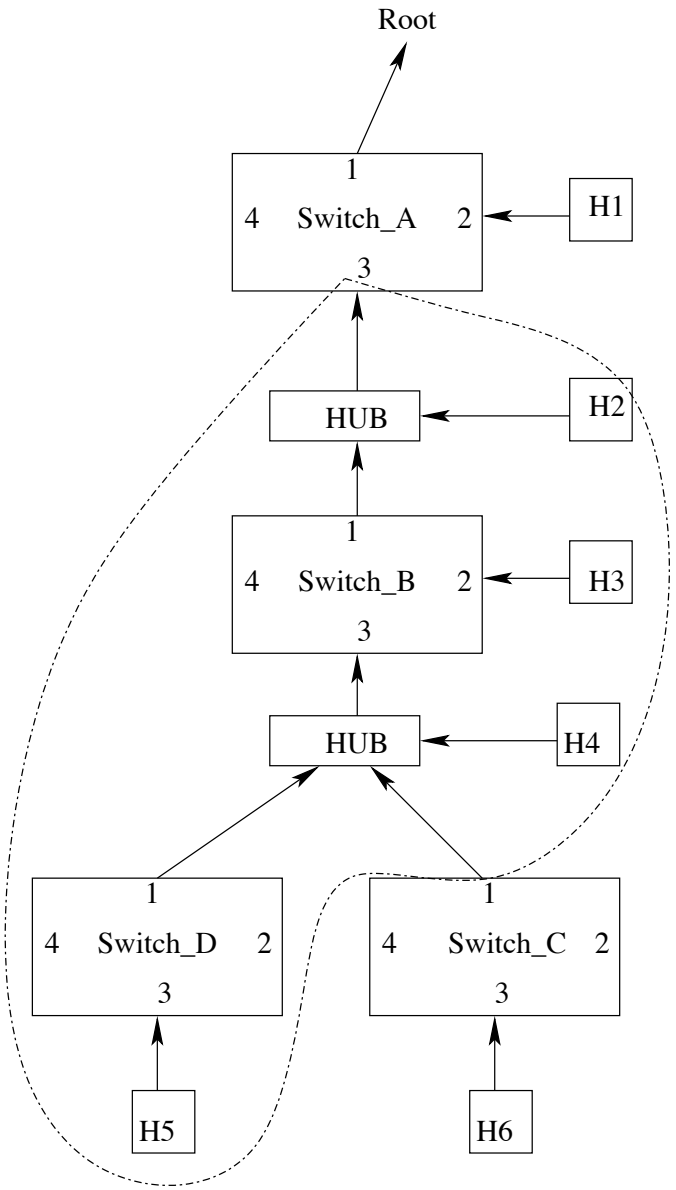
Figure 5 shows examples of each of the four cases in the tree search algorithm. The arrows show links toward the root of the tree. Hosts, labeled as H1-H6, are drawn though they would not have been located before running the algorithm. The two hubs do not participate in the spanning tree algorithm and are shown simply to explain the underlying network configuration. Relevant forwarding entries for the switches are shown to the right of the figure. An example of Case 1 is physical address M_a at $Switch_A$. Because $F(Switch_A, M_a) = Port2$, which is not a switch trunk, M_a must be located where $H1$ is shown; the search is complete. Examples of Case 2 are M_b, M_c, and M_d on $Switch_A$. Each of these addresses has a forwarding entry on $Port3$, a switch trunk away from the root. Thus, the hosts using these physical addresses must be located below $Port3$, and only those switches should

be traversed. An example of Case 3 is M_c at $Switch_C$. Because $F(Switch_C, M_c) = Port1$ is a trunk toward the root, the host must be located above $Port1$ on $Switch_C$. The search would not need to check any switch below $Switch_C$ if any existed. An example of Case 4 is M_c at $Switch_B$. Since $Switch_B$ has no forwarding entry for M_c, the search must check all of $Switch_B$ descendants (namely $Switch_C$ and $Switch_D$). If a descendant had a forward edge to the M_c (e.g., if $Switch_D$ has an entry for Case 2), no further descendants would to have be searched.

Here is how the algorithm would work for physical address M_c starting from $Switch_A$. At $Switch_A$, because the forwarding port, $F(Switch_A, M_C) = Port3$, is a forward edge in the spanning tree (Case 2), the host must be located below port $Port3$. The search continues to $Switch_A$'s descendants on $Port3$, namely $Switch_B$. Since $Switch_B$ has no forwarding entry for M_c (i.e., $F(Switch_B, M_C) = \emptyset$), all of $Switch_B$'s descendants must be searched (Case 4). Let $Switch_D$ be the first descendant. Because it has no forwarding entry for M_c, the search would continue with each of $Switch_D$'s descendants; but, since $Switch_D$ is a leaf node, the search returns to $Switch_B$. The next descendant from $Switch_B$ is $Switch_C$, which has a forwarding entry for M_c (namely, $F(Switch_C, M_C) = Port1$) on a back edge (Case 3). Thus, the host must be located above $Port1$. The search then returns to $Switch_B$, which has no more descendants. The search then returns to $Switch_A$, where it completes because $Switch_A$ is in Case 2. The resulting area where the device may be located is indicated with the dashed line, which includes $H2 - H5$.

The forwarding tables can also be used to help validate the results from the first approach. In fact, the entire set of forwarding entries for each physical address can be tested against the assumed results. Every entry (except those on ports directly connected to the host) should be in the spanning tree and should point toward the switch(es) directly connected to the host or the region.

The algorithm inputs the physical address of a host, but the user generally will only have the host's IP address. The router's *ip.ipNetToMedia* table can be used to find the IP address for the given physical address. If the host is a router, the path should include the port it uses to connect to the switch network,



Forwarding Tables

Switch	Address	Port
Switch_A	M_a	2
Switch_A	M_b	3
Switch_A	M_c	3
Switch_A	M_d	3
Switch_B	M_b	1
Switch_B	M_d	3
Switch_C	M_c	1

Figure 5: Example Network Demonstrating Four Cases in Algorithm

which is determined from the *ip.ipAddrTable* table. After learning the port to the switch network on the router, the physical address can also be found in its *interface.ifTable* table.

4.4 VLAN Support

In networks with VLANs we would like to find the spanning tree for a particular VLAN. Such devices generally have a proprietary (vendor specific) MIB that is a variation of the *dot1StpPortTable* but including a VLAN field. The process can be repeated using only those lines in the vendor specific *stpPortTable* whose VLAN field given VLAN ID. For example, Avaya Inc.'s Cajun switches use the *promBridgePortTable* table, which includes a VLAN identifier that can be mapped to the VLAN number using the *promVlanTable* table.

Without VLANs, a link will only appear in one switch's *StpPortTable*, in particular, the switch that is farther from the bridge root. With VLANs, the bridge root may change from one VLAN to another. Thus, the link could appear in both switches' *StpPortTable* tables. We assume that when VLANs are used, all switches share the same VLAN domain (i.e., there are no partitions in the switched domain with separately administered, isolated VLANs).

5 Conclusion

This paper describes an approach for finding the layer-2 topology of a network using SNMP queries to its network devices. The analysis first uses the spanning tree tables to find the links between switches. The spanning tree tables available through SNMP do not give the topology directly, but need to be processed to find the appropriate links and ports. Next, the analysis examines the location of endpoints (e.g., hosts) relative to the switches by examining the forwarding tables for each of the switches.

This paper does not provide experimental results for various reasons. The correctness of the results currently can only be verified by checking the physical wiring between devices. Only a prototype for the implementation exists; a robust version of the implementation is being developed. The performance is expected to vary with the size of the network. Noting these concerns, a few preliminary results can be given. The algorithms were tested on part of an enterprise network with over 200 routers and switches.

The link topology algorithm (generating a topology per VLAN) took about 33 seconds to run. The endpoint analysis ran on two of the larger switched networks using all the physical addresses cached in the routers connected to the switched networks (consisting of 400 and 214 hosts). The program took about 30 seconds to run on one network and 105 seconds on the other. These times do not include the time to collect the SNMP data from the network.

One area of future work is comparing this approach with the related approaches. For example, this approach, the pattern matching approach in [10], and the forwarding table based approach in [11, 12] all attempt to build a layer-2 topology.

References

[1] M. Bearden *et al.*, "Assessing network readiness for IP telephony," in *to appear in IEEE Int'l Conf. on Communications (ICC'2002)*, 2002.

[2] IEEE, *ANSI/IEEE Std. 802.1D: Part 3 Media Access Control (MAC) Bridges*, 1998 ed., 1998.

[3] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol (SNMP)," May 1990. RFC 1157.

[4] M. Rose and K. McCloghrie, "Concise MIB definitions," Mar. 1991. RFC 1212.

[5] D. T. Stott, "Snmp-based layer-3 path discovery," Tech. Rep. ALR-2002-005, Avaya Labs Research, Avaya Inc., Basking Ridge, NJ, 2002.

[6] R. Siamwalla, R. Sharma, and S. Keshav, "Discovering Internet topology." http://www.cs.cornell.edu/skeshav/papers/discovery.pdf, 1999.

[7] H. Burch and B. Cheswick, "Mapping the Internet," *IEEE Computer*, vol. 32, pp. 97–98, Apr. 1999.

[8] R. Govindan and H. Tangmunarunkit, "Heuristics for Internet map discovery," in *Proc. of the 2000 IEEE Computer and Communications Societies Conf. on Computer Communications (INFOCOM-00)*, (Los Alamitos, CA), IEEE, Mar. 26-30, 2000.

[9] B. Huffaker, M. Fomenkov, D. Moore, and k. claffy, "Macroscopic analyses of the infrastructure: Measurement and visualization of Internet connectivity and performance," in *Proc. of PAM2001–A workshop on Passive and Active Measurements*, (Amsterdam, Netherlands), Apr. 23-24, 2001.

[10] W. Zhao, J. J. Li, and D. T. Stott, "A method for heterogeneous network discovery." Internal Technical Report, Avaya Labs Research, Avaya Inc., Dec. 2001.

[11] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology discovery in heterogeneous IP networks," in *Proc. of the 2000 IEEE Computer and Communications Societies Conf. on Computer Communications (INFOCOM-00)*, (Los Alamitos, CA), pp. 265–274, IEEE, Mar. 26-30, 2000.

[12] B. Lowekamp, D. R. O'Hallaron, and T. R. Gross, "Topology discovery for large Ethernet networks," in *ACM SIGCOMM 2001*, (San Diego, CA), pp. 237–248, ACM, Aug. 27-31, 2001.

[13] Hewlett-Packard Co., *HP Toptools 5.5 User Guide*, 2001.

[14] Peregrine Systems, Inc., "InfraTools Network Discovery." http://www.peregrine.com/, 2001.

[15] Hewlett-Packard Co., "Discovering and mapping level 2 devices." http://www.openview.hp.com/library/papers/index.asp, Mar. 2001.

[16] E. Decker, P. Langille, A. Rijsinghani, and K. McCloghrie, "Definitions of managed objects for bridges," July 1993. RFC 1493.

[17] *Specification of Abstract Syntax Notation One (ASN.1)*, Dec. 1987.