Finding Ethernet-Type Network Topology is Not Easy

Hassan Gobjuka, Yuri Breitbart
Department of Computer Science
Kent State University
Kent, OH 44242
{hgobjuka,yuri}@cs.kent.edu

Abstract—**In this paper we investigate the problem of finding a layer-2 network topology when the information available from SNMP MIB is incomplete. We prove that finding a network topology in this case is NP-hard. We further prove that deciding whether the given information defines a unique network topology is a co-NP-hard problem. We show that if there is a single node r such that every other network node sees it, then the network topology can be restored in polynomial (in the number of network ports) time. Finally, we design a polynomial time heuristic algorithm to restore a topology when the information available from SNMP MIB is incomplete and conduct extensive experiments with it to determine how often the algorithm succeeds in finding topology. Our results indicate that our algorithm restores the network topology in close to 100% of all test cases.**

Keywords: Layer-2 Topology Discovery, Ethernet LANs, SNMP MIB, Switches, NP-hard.

I. Introduction

With the vast proliferation of VLANs, local networks may span large geographical distances. Modern local networks may contain hundreds of bridges and switches. In this network environment a network management of local area networks (i.e. network management at the second layer of the ISO hierarchy) becomes an important and challenging problem. Many network management tasks (such as performance analysis, root cause analysis, and fault identification, for example) critically depend on a knowledge of network connectivity. There are, however, very few network tools that enable network managers to maintain an accurate view of network connections. Without such tools there is a high probability of making wrong decisions either on adjusting network performance or on identifying network faults and network traffic bottlenecks.

Despite the importance of the network topology information, especially at the LAN level (layer-2 of the ISO hierarchy), there are significant difficulties in obtaining the topology information. Very few commercial network management platforms available on the market today offer general-purpose tools for automatic network topology discovery. Commercial tools that are currently on the market (such as HP's OpenView (openview.hp.com), IBM's Tivoli (tivoli.com), Cisco's Discovery Protocol (`www.cisco.com`), and Nortel's Discovery Protocol (`www.nortelnetworks.com`), for example) are based on proprietary information and often fail to capture many layer-2 connections in large Ethernet networks. There are several fundamental difficulties in obtaining layer-2 network connections.

1. Most of the current network topology tools collect and manage networks at the IP layer (layer-3 of the ISO hierarchy) and require network managers to maintain layer-2 connections manually. Furthermore, the Management Information Base (MIB) of layer-2 network elements does not provide information on their immediate neighbors [16]. To overcome this difficulty, the IEEE800.11ab Committee finished a proposal on a new layer-2 discovery protocol called *Link Layer Discovery Protocol* [7]. It allows layer-2 neighbors to notify one another of their presence. However, even if vendors would embrace the protocol, there is a large portion of legacy hardware in networks that still need a general layer-2 network discovery protocol and the proposed protocol may be implemented differently by different vendors.

2. Layer-2 network elements - switches and bridges - are transparent to layer-3 elements such as hosts and routers. Bridges and switches are involved in limited information exchanges. They intensely communicate with their neighbors only during the *spanning tree protocol* operation [15]. The only useable MIB information maintained by switches and bridges is in the *Address Forwarding Table (AFT)* - the set of MAC addresses that are reachable from a port of a given node [8]. If AFTs are $complete$, (that is, they contain all and only nodes that can be reached from a node's port), then the procedure to derive network connections has been described in [5], [6]. However, it is unrealistic to expect that the information in address forwarding tables is complete. First, many bridges and switches are connected within network using "out-of-band" ports. These ports do not participate in the main network and instead these ports are used only for administrative purposes. Secondly, to generate entries in the address forwarding table, the network node must be involved in communication all the time. Thus, some systems [6] generate additional traffic in the network that consumes a bandwidth that otherwise could have been used for genuine information exchange among network nodes. Thirdly, some switches or bridges may not allow to access their database and some other nodes may not be allowed to appear in the address forwarding tables of other nodes for security or any other administrative reasons.

In this paper we prove that finding a layer-2 network topology from a given set of incomplete AFTs is an NP-hard problem. We further prove that deciding whether an incomplete set of AFTs defines a unique network topology is a co-NP-hard problem. We design several heuristic algorithms to find network topology. Our first algorithm discovers the topology if there is node r such that any other node a in the network has a port a_i whose AFT contains r; the other one uses a set of rules to extend AFTs with additional nodes and finds a network topology in polynomial time. While the first algorithm always finds a topology in a polynomial time, the second algorithm may fail to find some network connections. While it is theoretically possible that the algorithm may not find a topology, our tests indicate that it is a very rare event.

port	AFT	port	AFT	port	AFT
a_1	1	b_1	1, 2, a	c_1	3, b
a_2	2	b_2	4	c_2	5, 6, d
a_3	d	b_3	3	c_3	4
d_1	5	d_2	6	d_3	a

TABLE I
INCOMPLETE SET OF AFTS OBTAINED FROM THE NETWORK IN FIG. 1

A. Related Work

Layer-2 network topology problems were addressed in the research community by several researchers [2], [4], [5], [6], [12], [17], [18]. For the set of complete AFTs the algorithms from [3], [5], [6] find the layer-2 topology for multisubnet networks. In [5] the authors observed that for multisubnet networks the network topology may not be unique even for the set of $complete\ AFT$s. In such a case finding an exact topology is not possible and algorithm from [6] generates some network fragments that can be uniquely determined and supplies the network manager with a set of possible topologies. In [10] a criterion was introduced on the set of complete AFTs guaranteeing a unique topology for multisubnet networks. Bejerano [4] proposed a very simple layer-2 topology restoration method for multisubnet networks. While his method restores a layer-2 network topology in a wide variety of cases, it cannot guarantee a topology restoration. His method also requires a completeness of input AFTs.
For networks with a single subnet, the set of incomplete AFTs may also define more than one topology [12]. Lowekamp *et.al.* [12] described a technique for inferring a connection between two nodes based on their AFTs. However, the proposed solution may fail to restore a unique topology even when such topology does exist. Indeed, consider the network N depicted in Figure 1 and whose AFTs are given in Table I. Network N contains non-terminal nodes a, b, c and d each with three ports and terminal nodes 1, 2, 3, 4, 5, and 6.
The algorithm from [12] finds only connections between ports a_3 and d_3 and between ports b_2 and c_1, where the latter is a direct connection. On the other hand, as we show in this paper the network topology depicted in Figure 1 is restorable by algorithms proposed in this paper.

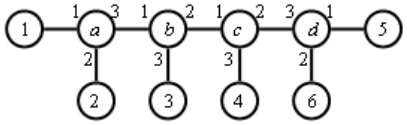

Fig. 1. Example of network that [12] will fail to restore connections

To discover the layer-2 network topology with incomplete AFTs, [17] proposed a two-stage approach. At the first stage they try to complete incomplete AFTs by using AFT's extension rules. If AFTs can be successfully completed, then the topology discovery enters the second stage, where the algorithms from [5], [6] are used to generate the set of network connections. The authors of [17] asserted that their set of rules is complete. That is, an application of their rules completes the set of incomplete AFTs. However, as we prove here the topology restoration problem with incomplete AFTs is NP-hard. Consequently, provided that $P \neq NP$, either the set of rules in [17] requires an exponential time to derive network connections, or the set of these rules is incomplete.
In [18] another method for deriving layer-2 topology was proposed. The method was based on a knowledge of a root of a spanning tree produced by the spanning tree protocol. However, unlike the Bridge information, the information on the spanning tree root is not regularly supplied by the majority of network vendors.
Recently Black *et. al.* [2] listed some problems with finding a layer-2 topology using Bridge MIB data. They proposed a new protocol to find a layer-2 topology without querying network MIB information. However, their approach requires placing custom designed network daemons on each host in the network, which some network managers might find objectionable.

B. Organization of the Paper

The rest of the paper is organized as follows. The next section describes the network model used in this paper. Section III proves the NP-hardness of the topology restoration problem and the co-NP-hardness of deciding whether the set of AFTs guarantees a unique topology. Section IV describes AFT extension rules and proves their correctness. Section V describes a polynomial time topology restoration algorithm for networks where each node sees the root of the network tree. It also describes two heuristic algorithms to find a network topology for an arbitrary set of AFTs. Experimental results are described in Section VI. Finally, Section VII concludes the paper.

II. BACKGROUND AND NETWORK MODEL

We refer to the network domain over which the topology discovery to be performed as a $switching$ domain S. Switching domain is defined as the maximal set of hosts, switches and bridges such that there is a path between every pair of nodes from S involving only nodes from S. In addition, no router is involved in routing packets between any two nodes in S. Switches and bridges in a switching domain employ the *spanning tree protocol* [15] to determine a unique forwarding path between any two nodes in S. Consequently, we model a switching domain as a tree and assume that network N consists of a single switching domain. Thus, we model the network as an undirected tree $N =< V,\ E >$, where V is a set of all network elements and every $e \in E$ represents a direct physical connection between two network elements. The internal (i.e. non-leaf) nodes of the tree represent switches and bridges and the leaf (terminal) nodes represent hosts. Packets in the network are forwarded from node a to node b using the tree path existing between nodes a and b. With each node a of network N we associate a number of ports denoted by $p(a)$ and refer to port i of node a as a_i. If node a is terminal, then it has a single port. We call an edge a *terminal edge* if at least one of its end points is a terminal node.
We say that two nodes a and b are connected by ports a_i and b_j if and only if there is a path in N between nodes a and b that starts at port a_i and ends at port b_j. The $length$ of the path is the number of edges in the path. If the length of the path is one, we say that ports a_i and b_j are $directly$ connected. For example in the network of Figure 1 port a_3 is directly connected to port b_1, whereas port a_3 connected to port d_3.
For each port a_i, the set of nodes that have been learned by the *backward learning algorithm* [19] is called an *Address Forwarding Table (AFT)* and is denoted by $AFT(a_i)$. Intuitively

it means that $AFT(a_i)$ is the set of nodes that are connected to a_i. If $b \in AFT(a_i)$ then we say that port a_i sees node b and that node b is seen by port a_i. If port a_i sees node b, it does not follow that there is a port of node b that sees node a. For example, for the set of AFTs given in Table I port b_1 sees node a ($AFT(b_1)$ contains node a) but no port of a sees node b.

We say $AFT(a_i)$ is *complete* if $AFT(a_i)$ contains *all* nodes to which packets can be sent from port a_i and does not include any node that cannot be reached from a_i. For example, for the set of AFTs given in Table I, $AFT(c_2)$ is complete whereas $AFT(a_3)$ is incomplete, since it does not include for example node b that can be reached from port a_3. If $AFT(a_i)$ is complete for any port of any node in N, then if a_i sees node b, then there is a port b_j that sees node a. Since a terminal node a contains a single port, it can see all nodes of the network except itself. Thus, we assume that the AFT of every terminal node is complete and includes all network nodes, except itself. The set of all nodes that are seen from all ports of node a except port a_i is called *complementary to a_i address forwarding table* and is denoted by $CAFT(a_i)$. Since a network is a tree, no node can see itself on any of its ports. Thus, for any port a_i, $a \in CAFT(a_i)$. To illustrate, for the set of AFTs given in Table I, $CAFT(a_3)$ = {1,2,a}. If node a is a terminal, then $CAFT(a_i)$ contains only a, since it is the only node that a cannot see.

Suppose that there is a path between ports a_i and b_j in N. Then $CAFT(a_i) \bigcap CAFT(b_j)$ is empty, since otherwise we would have at least one node c that can be reached by two different paths: one from $a_k, i \neq k$ and another from $b_l, j \neq l$. Thus, if $CAFT(a_i) \bigcap CAFT(b_j)$ is not empty, then there is no path between ports a_i and b_j.

If ports a_i and b_j are directly connected, then the intersection of $AFT(a_i)$ and $AFT(b_j)$ is empty. In the case of networks with complete AFTs, we proved in [5] the following theorem called *Direct Connection Theorem*.

Theorem II.1: [5] Ports a_i and b_j are directly connected in N if and only if the intersection of $AFT(a_i)$ and $AFT(b_j)$ is empty and union $AFT(a_i)$ and $AFT(b_j)$ is the set of all nodes in N.

The Direct Connection Theorem can be restated as follows.

Theorem II.2: Let $AFT(a_i)$ and $AFT(b_j)$ be complete for every port of node a and node b. There is a direct connection between a_i and b_j if and only if both the intersection of $AFT(a_i)$ and $AFT(b_j)$ and the intersection of $CAFT(a_i)$ and $CAFT(b_j)$ are empty.

Proof Since $AFT(a_i)$ and $AFT(b_j)$ are complete, $CAFT(a_i)$ = $V - AFT(a_i)$ and $CAFT(b_j)$ = $V - AFT(b_j)$. Since the intersection of $CAFT$s is empty, we obtain that V = $AFT(a_i) \bigcup AFT(b_j)$. Using the direct connection theorem from [5], we obtain the theorem assertion. □

In the case of incomplete set of AFTs, the theorem II.2 is only necessary condition for asserting a direct connection between ports a_i and b_j. Thus, to model a direct connection between two ports a_i and b_j for networks with incomplete AFTs, we introduce a notion of a *potential direct connection* as follows. We say that there is a potential direct connection between a_i and b_j if and only if the intersection of $AFT(a_i)$ and $AFT(b_j)$ as well as the intersection of their $CAFT$s are empty. A direct connection between a_i and b_j is also a potential direct connection between them. If, however, there is a potential direct connection between a_i and b_j and the AFTs are incomplete, it does not mean that there is a direct connection between a_i and b_j. For example, it is easy to see that there is a potential direct connection between a_1 and c_2 (see Table I), but there is neither direct or indirect connection between ports a_1 and c_2 in network depicted in Figure 1.

To this end we define a notion of a *Potential Connections Graph (PCG)* for the given set of input AFTs as follows. Each port of N is a node in the PCG. There is an edge between ports a_i and b_j in PCG if and only if there is a potential direct connection between a_i and b_j. Thus, the set of edges in the PCG is a superset of the set of direct connections in N. Figure 2 depicts a potential connection graph for the set of AFTs given in Table I. The thick edges indicate direct connections between these ports and each terminal node is identified with its single port.

Any topology restoration algorithm may start with the set of potential direct connections between any two ports and gradually eliminate those that cannot be direct connections until we, hopefully, find all direct connections. However, as we prove here, this process may require an exponential time unless $P = NP$.

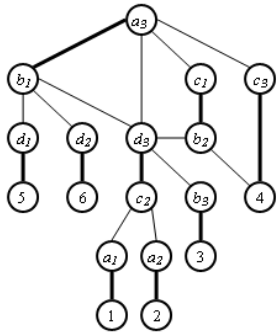

Fig. 2. Potential Connection Graph for the network of Figure 1

III. NP-Hardness Results

In this section we prove that two network topology restoration problems with incomplete set of AFTs are inherently difficult. Let θ be a set of not necessarily complete but non empty AFTs for all network ports in N. Our first problem (termed $Topology\ Restoration\ Problem$) is to restore the network topology for a given set of AFTs in a polynomial (in the number of network nodes) time. Our second problem (termed $Topology\ Uniqueness\ Problem$) is to decide whether a set of given AFTs admits more than one topology. We prove below that the $Topology\ Restoration\ Problem$ is NP-hard and the $Topology\ Uniqueness\ Problem$ is co-NP-Hard in the number of network nodes. The $Topology\ Restoration\ Problem$ remains NP-hard even if the set of input AFTs satisfies the following $separation\ restriction$: *the network contains two distinct nodes a and b such that every node c in N sees either a or b or both but neither the set of nodes in N that sees a nor the set of nodes in N that sees b coincides with the set of all nodes in N*. This is probably the strongest restriction on the set AFTs that makes the problem NP-hard, since as we show in Section V if there is at least one node in N such that every other node sees it, then the problem becomes polynomial.

We prove NP-hardness of $Topology\ Restoration\ Problem$ by reducing to it the betweenness problem (which is known

port	AFT	port	AFT	port	AFT
a_1	x	a_2	y	a_3	t_a
b_1	x, a	b_2	y, β	b_3	t_b
c_1	x	c_2	y	c_3	t_c
α_1	x, t_a	α_2	t_c		
β_1	t_a	β_2	y, t_c		

TABLE II
AFTS FOR THE NETWORKS IN FIG. 3

to be NP-hard [9]). The betweenness problem is defined as follows [9]. *Given a finite set A and a collection C of ordered triples $\{< a,\ b,\ c >\}$ of distinct elements from A, is there an order $<$ on A such that either $a < b < c$ or $c < b < a$?*

Theorem III.1: Let θ be a set of incomplete and non-empty AFTs satisfying the separation restriction. Then the topology restoration problem is NP-hard in the number of network nodes.

Proof Consider an arbitrary instance of betweenness problem $I(A, C)$. For the instance of the betweenness problem we define the set of ports for network N as follows. For every element $a \in A$, we create one non-terminal node a with three ports and one terminal node t_a. For every triple $< a, b, c > \in C$, we create two non-terminal nodes α and β with two ports each. Finally, we introduce two terminal nodes x and y. We define the set of AFTs for each of the ports as shown in Table II. Let N be the set of all nodes in the network. From our construction, it is easy to see that the set of AFTs satisfies the separation restriction for nodes x and y.

We first show that for each triple $< a, b, c >$ there are exactly two topologies T_1 and T_2 shown in Figure 3. Indeed, since α and β are seen on ports b_1 and b_2, respectively, it follows that nodes α and β cannot be directly connected. Thus, t_a is either connected to a_3 or to β_1. In the first case we obtain topology T1 and in the second case we obtain a topology T2.

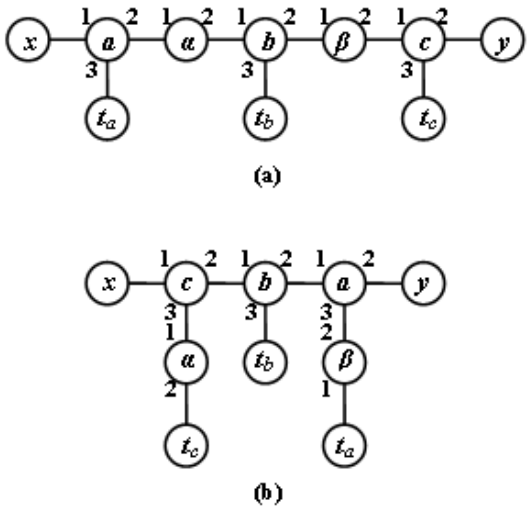

Fig. 3. The two topologies can be constructed for every triple $(a, b, c) \in C$.

We shall show now that a network topology given by θ can be restored in polynomial time if and only if there is a solution to the instance I of the betweenness problem.

Suppose that instance I allows a betweenness ordering on A. Let C be a set of triples $\{a,\ b,\ c\}$ and S be a network segment corresponding to triple $< a,\ b,\ c >$ from C as described above. If triple $< a,\ b,\ c >$) follows the order $a < b < c$ then the topology T_1 depicted in Figure 3 is selected for S, otherwise if triple $< a,\ b,\ c >$ follows the order $c < b < a$ then the topology T_2 depicted in Figure 3 is selected for S. In either case, since there is an ordering on A satisfying the betweenness condition, for each of the topologies, triple elements are located on path between x and y and therefore there is a path between x and y such that all elements of A are on this path. Consequently, merging all network segments we obtain a tree that respects the ordering on A.

Now suppose that for the set of input AFTs we have a topology T. Since T is a tree, there is an order on a set of nodes that correspond to a set of labels A. Let $x,\ a,\ b,\ c,\ y$ be a path in T. Then either there is a triple $< a,\ b,\ c >$ or $< c,\ b,\ a >$ in C. Since there is a network that is compatible with all segments, then the order of the nodes in the network topology provides the required betweenness ordering on A for I. □

Our next question is how difficult it is to decide whether a set of input AFTs defines a unique topology? We prove now that this problem is co-NP-Hard by proving that deciding whether the set of AFTs defines more than one topology is NP-hard. Namely, the following theorem holds.

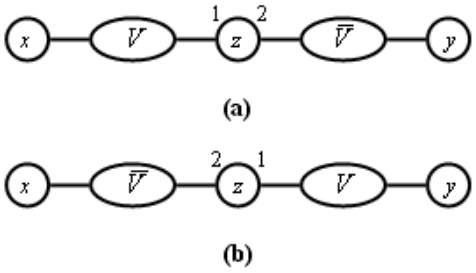

Fig. 4. Two topologies that can be obtained from a given instance of betweenness.

Theorem III.2: Deciding whether the set of AFTs defines more than one topology is NP-hard.

Proof the betweenness problem to our problem. We generate a set of nodes V and AFTs for network N as we defined them in the proof of Theorem III.1. In addition we introduce a set $\bar{V}$ such that if $a \in V$, then $\bar{a} \in \bar{V}$. The set of AFTs for $\bar{V}$ is exactly the same as a set of AFTs for V where each node v in V is replaced with $\bar{v}$ in $\bar{V}$. Finally, we introduce node z that has two ports and $AFT(z_1) = V$ and $AFT(z_2) = \bar{V}$. If a solution to instance I of betweenness is given, we create two topologies T_1 and T_2 as follows. The network generated for the set V is exactly the same as it is in the proof of Theorem III.1 and the same network is created for the set of nodes $\bar{V}$. We then prove that there is more than one topology for the set of constructed AFTs if and only if there is a solution to instance I of the betweenness problem. The first topology then is the network for V placed between x and z_1 followed by the network for $\bar{V}$ placed between nodes z_2 and y. The second topology is the network for $\bar{V}$ placed between nodes x followed by the network for V placed between z_2 and y. The two topologies are depicted in Figure 4(a) and (b), respectively. □

port	AFT	port	AFT
b_1	a	c_2	d
b_2	4	c_3	4
c_1	b		

TABLE III
THE NEW SET OF AFTS OBTAINED FROM TABLE I

IV. HEURISTIC TOPOLOGY ALGORITHM USING AFT EXPANSION RULES

The ultimate goal of the topology discovery process is to find direct connections between network ports. While it is difficult to find network connections for the set of incomplete AFTs, there is at least one case when such connections can be easily determined.

Let t be a terminal node in network N and θ be a set of incomplete AFTs for N. Set θ may have several AFTs each of which contains only node t. If the set of AFTs of node a is complete and $AFT(a_i) = \{t\}$, or there is a single $AFT(a_i)$ that contains only t, then t is directly connected to a_i. Indeed, the AFT of a port that is directly connected to t must contain only t. If the set of input AFTs contains a single $AFT(a_i)$ that contains t, we conclude that a_i is directly connected to t.

If we determined that a_i and t are directly connected, we identify t with a_i and eliminate t from each of AFTs that contains t replacing it with a. We call this the *AFT reduction* process.

In some situation, the reduction process can completely restore the network topology. Indeed, consider the set of AFTs shown in Table I. It is easy to see that terminal nodes 1,2,3,5, and 6 are directly connected to a_1, a_2, b_3, d_2, and d_3, respectively. Using the reduction process we obtain that a and d are two new terminals (since other two ports of a and d are already connected). Thus, the set of new terminals is a, d, and 4. Node b has only two unconnected ports, since b_3 is connected to 3. Consequently, the new set of AFTs is as shown in Table III.

In the new table, terminals a and d are directly connected to b_1 and c_2, respectively. Thus, after the second application of the reduction process the node b is a new terminal and node c has only two ports: one connected to b and another connected to 4. Thus, the network topology is completely restored. The final network is shown in Figure 1.

If there is no terminal node in N such that the set of input AFTs can be reduced, we say that the set of input AFTs is irreducible. In the rest of this section we assume that the set of input AFTs is irreducible.

A. Rules

We introduce now expansion rules to expand AFTs based on the content of other AFTs and based on already established connections between some nodes in network N. Ideally, one would want to generate a set of rules whose systematic application to a set of incomplete AFTs results in a completion of every AFT in a polynomial in the number of network nodes time. However, there is a little hope that such a set of rules would be found due to the $NP - hardness$ of the topology restoration problem. Prior to formulating AFTs expansion rules we first establish a relationship that exists between terminal and non-terminal nodes for the complete $AFT(a_i)$.

Lemma IV.1: Consider port a_i and suppose that $AFT(a_i)$ is complete and contains k non-terminal nodes $a^1,\ a^2,\ \ldots,\ a^k$ where node a^i has n_i ports. Then $AFT(a_i)$ must contain exactly $\sum_{i=1}^{k} n_i$ - $(2k-1)$ terminal nodes.

Proof The lemma assertion is derived by induction on k the number of network nodes in the $AFT(a_i)$. If $k = 0$, then $AFT(a_i)$ does not contain any nonterminal nodes. Since $AFT(a_i)$ cannot be empty, it must contain at least one node and this node is terminal. If $AFT(a_i)$ contains a single terminal node, we obtain the assertion of the lemma for $k = 0$. Suppose that $AFT(a_i)$ contains more than one terminal node. Since $AFT(a_i)$ is complete no other nodes can be added to $AFT(a_i)$. Since there must be a tree T rooted at a_i in N, T must contain all nodes in $AFT(a_i)$. However, since there are no other nonterminal nodes in $AFT(a_i)$ and consequently in T, we will not be able to establish connection between nodes in $AFT(a_i)$. Thus, the lemma for $k = 0$ is proven.

Suppose that the lemma is proven for $k = t - 1$. Let $k = t$. Let T be a tree with root a_i that corresponds to $AFT(a_i)$. Let a be a node in T with n_t ports whose $n_t - 1$ are connected to terminals and a single port p is connected to T. Clearly for any tree T such node a can be found. Let b be a node with n_{t-1} ports in which one of the ports is connected to a. If we remove a from T along with all terminal nodes that are connected to a, we obtain a new tree T' that has $t-1$ nonterminal nodes. Each nonterminal node in T' contains the same number of ports as in T with the exception of node b which contains $n_{t-1} - 1$ ports. Thus, by the induction assumption, T' contains $\sum_{i=1}^{t-1} n_i$ - 1 - $(2(t-1)-1)$ terminal nodes. Thus, T must contain $\sum_{i=1}^{t-1} n_i$ - 1 - $(2(t-1)-1)$ + 1 + n_t +1 which is equal to $\sum_{i=1}^{t} n_i$ - $(2t-1)$. □

We introduce now the following expansion rules for the irreducible set of AFTs.

1. **Basic Expansion Rule:** *If ports a_i and b_j are connected in N, then $AFT(a_i)$ and ($AFT(b_j)$) can be expanded with all nodes from $CAFT(b_j)$ and ($CAFT(a_i)$), respectively.* Indeed, the connection between a_i and b_j implies that every node that is not seen by port b_j (a_i) must be seen by port a_i (b_j) as shown in Figure 5.

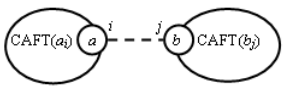

Fig. 5. Basic Expansion Rule

2. **Fork rule:** *If node a sees two nodes c and d on the same port and there is another node b that sees c and d on different ports, then a must also see b.*

For example, consider the network depicted in Figure 6. Suppose that $c \in AFT(b_j)$, $d \in AFT(b_k)$, $j \neq k$ and both nodes c and d are in $AFT(a_i)$, then node b is added to $AFT(a_i)$.

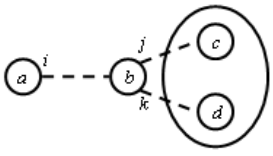

Fig. 6. Fork Rule

Proof of correctness Indeed, if a does not see b on port a_i,

then there is a path between a and b but the path uses a different port than a_i. In this case, regardless the port that b uses to connect to a, we obtain that the network contains a loop since there are at least two paths from a to c or from a to d.

3. **Tree Rule** *If* $a \in AFT(b_j)$ *and* $CAFT(b_j) \cap AFT(a_i) \neq \emptyset$*, then* $CAFT(b_j) \subseteq AFT(a_i)$.

For example, in the network depicted in Figure 7, if $AFT(b_j)$ contains node a and both $AFT(a_i)$ and $AFT(b_k)$ contain node c, then $AFT(a_i)$ contains also node b and all nodes that are in $AFT(b_k)$, $k \neq j$.

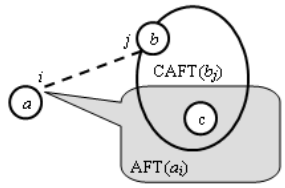

Fig. 7. Tree Rule

Proof of correctness Indeed, suppose that b_j is connected to some port of a. Since on port a_l we see some nodes that are seen by some port b_t, where $t \neq j$, then a_l must see b and b_j and a_l are connected. Indeed, if these ports are not connected, then under the conditions of the tree rule, we obtain that there is a loop in N.

4. **Counting Rule** *If there is a unique combination of terminal and non-terminal nodes that are currently not in* AFT*s of node* a *that together with the* $AFT(a_i)$ *satisfies the conditions of lemma IV.1 and compatible with the given set of* AFT*s, then these nodes should be added to* $AFT(a_i)$. For example, suppose that the current $AFT(a_i)$ contains node b with $k > 1$ ports. Then counting rule requires that the $AFT(a_i)$ must contain at least $k - 1$ terminal nodes. It follows from lemma IV.1 that nodes with exactly two ports do not impact the distribution of nodes that are currently not present in the AFTs. In other words, nodes with two ports may be added arbitrarily to any AFT without any effect on the equation asserted by lemma IV.1.

V. Topology Discovery Algorithms

In this section we describe two topology restoration algorithms. The first algorithm, termed *One-Node*, finds a layer-2 topology in polynomial time for the set of AFTs and node r such that every other node b sees r on one of its ports b_i. This situation can happen in practice, for instance, if each node in the network has a knowledge of the root of the spanning tree. The second algorithm, termed $Connection$, for the set of incomplete and irreducible AFTs, creates a PCG and eliminates from it edges that cannot be direct connections in the network.

A. One-Node Networks

Let r be a node in the network such that every other node b sees r on one of its ports b_j. We start with the description of the *Order Determination Procedure (ODP)* which finds a correct order of nodes b, c, ... on a path between nodes r and a, where $a \neq b$, c, ... After that we describe the $One - Node$ algorithm which uses the ODP to build a path between node r and any terminal node.

A.1 Order Determination Procedure

Consider two distinct nodes a and b that have at least two ports each. Suppose that node a sees nodes c and d on ports a_1 and a_2, respectively. Node b also sees c and d on ports b_1 and b_2, respectively. Suppose that $d \in AFT(a_2)$ and $d \in AFT(b_2)$. We claim then that a and b are seen on the same port of c. Indeed, if a and b are seen on two different ports of c, say c_1 and c_2, then c sees d also on ports c_1 and c_2. But since the network is a tree, each node can be seen only on a single port of any other node. Thus, we have proven the following lemma:

Lemma V.1: If c is in $AFT(a_1) \bigcap AFT(b_1)$ and d belongs to both $AFT(a_2)$ and $AFT(b_2)$, then there is a single port c_i, such that there is a path between c_i and d that includes nodes a and b.

Let a and b be two nodes that satisfy the condition of Lemma V.1. If $CAFT(a_1) \bigcap CAFT(b_2)$ is not empty and $CAFT(a_2) \bigcap CAFT(b_1)$ is empty, then the path between c_i and d includes node a followed by node b. Similarly, if $CAFT(a_2) \bigcap CAFT(b_1)$ is not empty and $CAFT(a_1) \bigcap CAFT(b_2)$ is empty then the path between c_i and d includes node b followed by node a. It is not possible that $CAFT(a_1) \bigcap CAFT(b_2)$ and $CAFT(a_2) \bigcap CAFT(b_1)$ are not empty, since in such a case there is no path between c_i and d that includes nodes a and b which contradicts lemma V.1. However, both $CAFT(a_1) \bigcap CAFT(b_2)$ and $CAFT(a_2) \bigcap CAFT(b_1)$ can be empty. In this case the order of nodes a and b in the path between c_i and d can be arbitrary. Thus, in the conditions of lemma V.1 we consider a procedure of $CAFT$s intersections to determine the order of nodes a and b in the path between c_i and d. We refer to this procedure as *Order Determination Procedure (ODP)*.

A.2 One-Node Algorithm

The input for the *One-Node* is a set of irreducible and incomplete AFTs and node r such that every other node in the network sees r. The output of the algorithm is a network topology consistent with the set of input AFTs. Without loss of generality, we assume that each node a sees r on port 1. Let T be the set of terminals of network N. The algorithm $One - Node$ is described in Figure 8. The algorithm starts with an arbitrary selection of terminal node t from N. It finds all AFTs that contain only t. Let $AFT(a_i)$=... = $AFT(b_j)$ = $\{t\}$. The algorithm decides which port can be directly connected to t by repetitive use of the ODP procedure. Then, applies the reduction process and updates the set of AFTs. This process is repeated until a topology is restored. Observe that since the ODP may make a random choice, the restored topology may not be unique.

To illustrate the algorithm, we consider the set of AFTs obtained from the network depicted in Figure 9(a) and given in Table IV.

Observe that node 1 is seen by every node in the network. The algorithm first builds path from terminal node 2 to node 1. The order of nodes a and b on the path between nodes 1 and 2 is determined by the ODP as follows. Since $CAFT(a_2) \bigcap CAFT(b_1) = \emptyset$ and $CAFT(a_1) \bigcap CAFT(b_2) \neq \emptyset$, there is a path between ports a_1 and b_2. Consequently, nodes b and 2 are merged into new terminal node b as shown in Figure 9(b) and AFTs are updated. Similarly, nodes a and b are merged

Input: A set θ of incomplete AFTs, and node r such that, without loss of generality, $r \in AFT(a_1)$ of every node a in the network
Output: A topology T that is compatible with θ

set $\alpha := \theta$;
set $T := \emptyset$;
set $R :=$ *All terminal nodes in the network*;
set $V :=$ *All non-terminal nodes in the network*;
set $F = \emptyset$; *All AFTs that contain only t*
repeat
1. select terminal node t from R;
2. for every $AFT(a_i) \in \alpha$ do if $AFT(a_i) = \{t\}$, set $F := F \bigcup \{AFT(a_i)\}$;
3. while ($F \neq \emptyset$) do
   (a) Using ODP on F find a port c_k that is directly connected to t;
   (b) Apply reduction procedure to c_k and t;
   (c) eliminate port c_k from node c; remove $AFT(c_k)$ from F;
   (d) if node c has become a terminal node, then
   - remove t from R;
   - set $R := R \bigcup \{c\}$;
   - set $V := V - \{c\}$;
4. if $R = \emptyset$, then return T;

until $R = \{r\}$;

Fig. 8. Formal description of the *One-Node* algorithm.

port	AFT	$CAFT$	port	AFT	$CAFT$
a_1	1	a, 2, 3	b_2	2	a, b, 1
a_2	2	a, 1, 3	c_1	1	c, 3
a_3	3	a, 1, 2	c_2	3	c, 1
b_1	a, 1	b, 2			

TABLE IV
SET OF AFTS FOR THE NETWORK DEPICTED IN FIG. 9

into new node b. Now we build a path between nodes 1 and 3. Since $CAFT(a_3) \bigcap CAFT(c_1) = \emptyset$ and $CAFT(a_1) \bigcap CAFT(c_2) = \emptyset$, the ODP breaks the ties between nodes a and c arbitrarily and chooses port c_2 as direct connection to terminal node 3. Then, a new node c is identified as shown in Figure 9(c). After that, nodes a and c merged into new node a and the topology is restored.

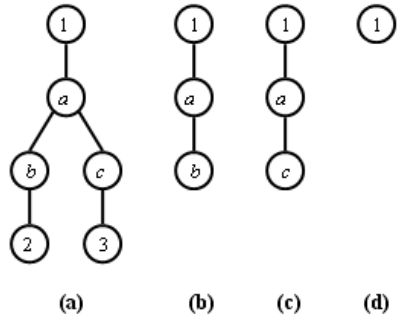

Fig. 9. Example of the *One-Node* algorithm application

B. Rules Procedure

The procedure receives as an input the set of incomplete AFTs. It systematically applies the $basic$, $fork$, $tree$ and $counting$ rules in any order and expands the AFTs. After each rule application the algorithm invokes the reduction procedure that may modify the set of input AFTs. The process continues as long as any rule or a reduction procedure is applicable. The time complexity of this algorithm is also $O(n^2)$.

C. Connection Algorithm

The algorithm receives as an input a set of irreducible and incomplete AFTs. The algorithm first calls the Rules procedure. If Rules restore the topology, the algorithm stops. In the case that the topology is not restored by the Rules procedure, the algorithm builds a PCG and initializes the set of matched port pairs from PCG as empty. We denote by CC the set of connected components of N obtained after the algorithm determined some connections already. Initially, each node of N is a connected component by itself. The algorithm works in stages, where initialization is stage 1. Let CC be the set of connected components after stage j (where $j > 1$). Two nodes a and b belong to the same connected component CC_i if and only if there is a path between some port of a and some port of b that contains only nodes from CC_i. Let $< c_k,\ b_l >$ be a terminal edge in PCG. Let us assume also that c_k belongs to CC_i and b_l belongs to the connected component CC_t, where $i \neq t$. Then, at stage j, the algorithm establishes a connection between c_k and b_l in N and extends M by pair $< c_k,\ b_l >$. The algorithm then replaces component CC_i with a new component CC_i that is obtained as a result of merging CC_i and CC_t. It then removes CC_t from the set of connected components CC. The algorithm then removes from PCG all edges whose end nodes belong to the same connected component. If, however, at stage j the algorithm cannot find any terminal edge $< c_k,\ b_l >$ in PCG, and c_k and b_l belong to different connected components, then the algorithm reports that the topology may not be unique and makes an arbitrary selection of $< c_k,\ b_l >$ for matching, where c_k and b_l belong to different connected components. The algorithm then repeats the process until either a topology is restored or the algorithm cannot make any matchings and there are unmatched ports remaining in PCG. The pseudocode of the $Connection$ algorithm is given in Figure 10.

Input: A set θ of incomplete AFTs
Output: A set $M \subseteq PCG$ of matchings

Invoke Rules Procedure;
If topology is restored, exit;
$M = \emptyset$;
repeat
do while there is a terminal node $a_i \in PCG(\theta)$
  find edge $U(a_i,\ b_j)$ in $PCG(\theta)$;
  1. select $U(a_i,\ b_j)$;
  2. $M = M \cup \{AFT(a_i),\ AFT(b_j)\}$;
  3. remove from $PCG(\theta)$ all edges, whose endpoints are (a_i) and (b_j);
  4. remove from $PCG(\theta)$ all edges that may create loop in the network;
if PCG is not empty
- select an arbitrary $U(a_i,\ b_j)$;
- Goto 2

until (PCG is empty) ;
if PCG is empty and M includes all ports, then print **Topology restored**; return M;
else print **No topology found**;

Fig. 10. A formal description of the Heuristic Topology Restoration Algorithm.

To illustrate the algorithm consider the set of AFTs given in Table V.
Applying the fork rule we determine that the $AFT(c_3)$ becomes $\{3,\ 4,\ d\}$. Then applying the tree rule we determine

port	AFT	port	AFT	port	AFT
a_1	1	c_1	1, a, b	d_2	4
a_2	3	c_2	2	d_3	2
b_1	1	c_3	3, 4	e_1	d
b_2	4	d_1	3	e_2	1, c

TABLE V
SET OF AFTS FOR THE NETWORK RESTORABLE BY CONNECTION ALGORITHM

port	AFT	port	AFT	port	AFT
a_1	1	c_1	1	d_3	5
a_2	2	c_2	2	e_1	5
a_3	4	c_3	3	e_2	3
b_1	3	d_1	4		
b_2	4	d_2	1, 3, c		

TABLE VI
CONNECTION ALGORITHM IS NOT ABLE TO RESTORE TOPOLOGY FOR THIS NETWORK

that the $AFT(a_2)$ becomes {2,3,4,c,d},the $AFT(b_2)$ becomes {2,3,4,c,d} and the $AFT(d_3)$ becomes {1,2,a,b,c}. The other AFTs did not change. Next we use the reduction process to identify c_2 with node 2, d_1 with the node 3 and d_2 with node 4. We observe now that d is a new terminal and nodes 2, 3, 4 are excluded from the terminals. Applying again the tree rule, the $AFT(c_3)$ becomes {d, e} and e_1 is identified with d. Repeating the reduction process we determine that c_3 which is identified with e (that became a new terminal). Node c becomes now new terminal and the set of AFTs is as follows. $AFT(a_1) = \{1\}$, $AFT(a_2) = \{c\}$, $AFT(b_1) = \{1\}$, $AFT(b_2) = \{c\}$. At this point no rules are applicable and the algorithm builds a potential connection graph shown in Figure 11. Arbitrary selection of an edge to break a loop in the graph guarantees a topology that is compatible with the set of given AFTs.

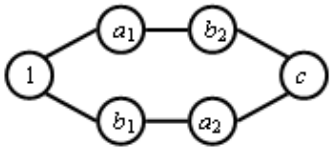

Fig. 11. Potential Connection Graph

Since the problem is $NP-hard$, it is possible that the algorithm may not return any topology. To illustrate, consider a network depicted in Figure 12. The input set of AFTs for this network is given in Table VI.
Observe that the reduction cannot be used and the potential connection graph doesn't contain any terminal edges. If the algorithm selects $(a_1, 1)$ as a connection, then nodes a and c are on the path between terminals 1 and 2, and node b is between nodes a and c. However, since node d sees nodes 1 and 3 on one port and node 5 on another port, node d cannot be placed anywhere to obtain a topology compatible with the set of given AFTs.

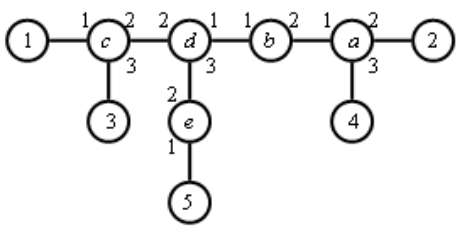

Fig. 12. Network for the AFTs of Table VI

VI. EXPERIMENTAL RESULTS

In this section we report results of our experiments with the $Rules$ and $Connection$ algorithms.

A. The Simulation Settings

We conducted our experiments using the testbed we implemented. Our experiments were run on a Pentium 4 2.4GHZ computer with 512MB of RAM and Windows XP operating system. We used $BRITE$ [14] to generate Power-Law-based random networks with different number of nodes that vary from 100 to 400. 36% of these nodes were switches and the rest were terminals. One of the randomly chosen nodes was used as a root and a shortest-path spanning tree algorithm created a tree rooted at the chosen node. The average degree of a nonterminal node in the tree was 4.3.
Then, a complete AFT was generated for each port of every non-terminal node. Then $s\%$, s=10, 30, 50, 70, 90 , of the randomly selected entries of each AFT were omitted to create an incomplete AFT for a given port. A selection procedure guaranteed that every terminal node appeared at least once in the resulting AFTs. We defined a $loss\ ratio$ as the ratio of the missing entries after we dropped randomly selected entries from the AFTs to the number of entries in the completed AFT. The input value of the loss ratio for the generated tree was defined as an average of loss ratios for every input AFT. We ran two sets of experiments, five times each for the same network configuration and on the same input. The first set of experiments were run for the algorithm $Rules$ and the second set for the algorithm $Connection$.

B. Simulation Results

To characterize the effectiveness of our algorithms we compared the loss ratio of the given input AFTs with the loss ratio of the output AFTs. If the loss ratio of the output AFTs is zero, it means that the topology was uniquely restored. The loss ratio more than 0% indicated that either the topology was successfully restored but it was not unique, or that the algorithm failed to find any valid topology. The results of our simulations are in Tables VII-IX, where Tables VII and VIII show the results of different size network inputs. Table IX shows the usage percentage of each Expansion Rule in restoring AFT entries. The first and fourth columns of Table VII represent the input loss ratio for the given AFTs (I/P %) for networks that contain 100 and 200 nodes, respectively. The second and fifth columns (R %) represent the loss ratio after applying the expansion rules. The third and sixth columns (C %) represent the loss ratio after applying the Connection algorithm. Table Table VIII shows the results for networks that have 300 and 400 nodes.

$I/P\,\%$	$R\,\%$	$C\,\%$	$I/P\,\%$	$R\,\%$	$C\,\%$
10	3.0	0.1	10	0.0	0.0
30	0.0	0.0	30	0.0	0.0
50	8.4	0.0	50	6.2	0.0
70	17.7	0.0	70	18.7	0.0
90	35.1	0.0	90	33.1	0.0

TABLE VII

THE AFT LOSS RATIO BEFORE AND AFTER RUNNING EXTENSION RULES AND CONNECTION HEURISTIC FOR 100- AND 200-NODE NETWORKS.

Our results show that the *Connection* algorithm was very effective in the layer-2 topology restoration. For all networks and for the loss ratio of 30% or less, the $Connection$ algorithm almost always succeeded in restoring a unique layer-2 topology. Furthermore, results also show that applying the expansion rules was sufficient to restore the unique topology when the AFTs have loss ratios of 30% or less in the vast majority of cases. We also observed that the $Rules$ algorithm used most extensively the $fork$ and $tree$ rules. However, the $counting$ rule was also used on the average in 5% of the cases. The results for the networks with high loss ratios indicate that the $Rules$ algorithm was not able to restore network topology in many cases, whereas the $Connection$ algorithm almost uniformly succeeded. The failure rates of the $Connection$ algorithm was very low.

$I/P\,\%$	$R\,\%$	$C\,\%$	$I/P\,\%$	$R\,\%$	$C\,\%$
10	0.0	0.0	10	0.0	0.0
30	0.0	0.0	30	0.0	0.0
50	12.3	0.0	50	8.1	0.0
70	27.2	0.3	70	21.7	0.4
90	35.3	0.0	90	34.3	0.0

TABLE VIII

THE AFT LOSS RATIO BEFORE AND AFTER RUNNING EXTENSION RULES AND CONNECTION HEURISTIC FOR 300- AND 400-NODE NETWORKS.

Network Size:	100	200	300	400
Fork Rule	50.4	59.1	40.6	42.2
Tree Rule	43.3	36.1	47.8	53.9
Counting Rule	6.1	4.6	11.5	3.7

TABLE IX

THE USAGE PERCENTAGE OF EACH EXTENSION RULE FOR DIFFERENT NETWORK SIZES.

VII. CONCLUSION

We addressed the problem of obtaining a layer-2 network topology for large Ethernet networks in the absence of accurate information in the AFTs of layer-2 network elements. We proved in this paper that obtaining a layer-2 topology in the absence of the accurate AFTs information is indeed $NP-hard$. We further proved that for a given set of input AFTs it is not possible to decide whether the given input guarantees a unique topology. We have also designed several heuristic algorithms whose time complexity is $O(n^2)$ and they restore the topology in the most cases. Indeed, even when almost 90% of the AFT information is missing, one of our algorithms restores a network topology with a probability close to one. In summary, we proved that any further attempts to find a polynomial algorithm for restoring layer-2 network connectivity is fruitless. Thus, one may try better and better heuristics.

VIII. ACKNOWLEDGEMENT

Special thanks to Victor Chepoi for suggesting betweenness problem as a candidate for reduction to Network topology problems. The authors also thank Feodor Dragan for many useful discussions.

REFERENCES

[1] A. Bierman and K. Jones, *Physical Topology MIB*. http://info.internet.isi.edu:80/in-notes/rfc/files/rfc2922.txt, September 2000.

[2] R. Black, A. Donnelly, C. Fournet *Ethernet Topology Discovery without Network Assistance*. Proceedings of the 12th IEEEInternational Conference on Network Protocols (ICNP'04), 2004

[3] Y. Bejerano, Y. Breitbart, M. Garofalakis, R. Rastogi, Physical Topology Discovery for Large Multi-Subnet Networks Proceedings of INFOCOM 2003, 2003.

[4] Y. Bejerano A Simple and Efficient Topology Discovery Scheme for Large Multisubnet Networks Proceedings of INFOCOM 2006, 2006.

[5] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz Topology Discovery in Heterogeneous Ip Networks, Proceedings of INFOCOM 2000, 2000.

[6] Y. Breitbart, M.Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz *Topology Discovery in Heterogeneous IP Networks: The Net-Inventory System*, IEEE/ACM Transactions on Networking, v. 12, 3, June 2004

[7] B. Boardman Layer 2 Layout: Layer 2 Discovery Digs Deep Network and System Management Workshop, Nov. 2003

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

[9] M. Garey, D. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completness, W.H. Freeman and Company, New York, 1979.

[10] H. Gobjuka, Y. Breitbart, Characterization of Layer-2 Unique Topologies in Multisubnet Local Networks, Proceedings of 31st IEEE LCN conference, Tampa, Florida 2006

[11] H. Gobjuka, Y. Breitbart, Finding Ethernet-Type Network Topology is Not Easy, Technical report, Department of Computer Science, Kent State University, August 2006.

[12] B. Lowekamp, D. O'Hallaron, and T. Gross, *Topology Discovery For Large Ethernet Networks* Proceedings of ACM SIGCOMM 2001, San Diego, CA, 2001

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

[14] A. Medina, et. al. *http://www.cs.bu.edu/brite*. Boston University, 2002

[15] R. Perlman *Interconnections, Second Edition*, Addison-Wesley, 2000 (Third Edition).

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

[17] Y. Sun, Z. Wu, Z. Shi *The Physical Topology Discovery for Switched Ethernet Base on Connection Reasoning* , Proceedings of ISCIT, 2005pp. 42-45

[18] Y. Sun, Z. Shi, Z. Wu, *A Discovery Algorithm for Physical Topology in Switched Networks*, Proceedings of the IEEE Conference on Local Computer Networks, pp. 311-317, 2005

[19] A. Tanenbaum, *Computer Networks, Fourth Edition*, Prentice Hall Ptr., 2002.