Taking The Skeletons Out Of The Closets: A Simple And Efficient Topology Discovery Scheme For Large Ethernet LANs

Yigal Bejerano

Bell Laboratories, Lucent Technologies

Abstract: Today, *Ethernet* is the dominant *local area network* (LAN) technology. These networks, typically, comprise large number of elements from different vendors. This raises considerable difficulties in performing network management tasks, such as resource management and root cause analysis, which are practically impossible without an up-to-date knowledge of the physical network topology. Given the dynamic nature of today's LANs, keeping track of topology information manually is a daunting (if not impossible) task. Therefore, it is essential to develop practical schemes for automatic inference of the physical topology of Ethernet networks. In this paper, we propose a simple and efficient algorithmic solution for discovering the physical topology of large, heterogeneous Ethernet LANs that may include multiple subnets as well as uncooperative network elements, like hubs. Our scheme utilizes only generic MIB information and *does not* required any hardware or software modification of the underlying network elements. By rigorous analysis, we prove that our method correctly infers the network topology and has low communication and computational overheads. Our simulation results show that the scheme successfully infers the complete topology in the vast majority of the cases, including many instances in which other methods fail. Finally, we implemented the proposed scheme to verify its inference capabilities. These properties confirm the practicality of our scheme for network management.

Keywords: Layer-2 Topology Discovery, Ethernet LANs, Subnets, SNMP MIB, Switches, Hubs, Graph Theory.

I. Introduction

Modern *Ethernet Local Area Networks* (LANs) are typically large networks that comprise hundreds or thousands of network elements from different vendors. Their complexity and heterogeneity raise arduous management tasks to network administrators. To this end, maintaining an accurate and complete knowledge of the *physical network topology* is a prerequisite to many critical network management tasks, including network diagnostic, resource management, event correlation, root cause analysis and server placement. This knowledge refers to the actual physical connections between the existing network elements. Due to the frequent changes of the element connectivity, accurate topology information, cannot be practically maintained without the aid of automatic topology discovery tools. In spite of its critical role for network management, it is very difficult to obtain this topology information. Consequently, the majority of commercial network-management tools, such as HP's OpenView (www.openview.hp.com) and IBM's Tivoli (www.tivoli.com),

are only capable of providing *network layer* connectivity (i.e., ISO *layer-3* also called *IP layer*). They present only router-to-router interconnections and router interface-to-subnet relationships. Inferring the network layer topology is relatively easy, since routers are aware of their immediate layer-3 neighbors as well as attached subnets and they publish this information through their *SNMP Management Information Base (MIB)* [1]. This information is sufficient to determine layer-3 topology, but it fails to capture the complex interconnections of the Ethernet LANs, that underlie the logical links of layer-3. Unfortunately, layer-2 elements, *e.g.*, "bridges" and "switches", do not provide similar information of their immediate layer-2 neighbors, which complicate the discovery of the physical network topology.

A. The Challenges

In this study we address the difficult task of inferring the physical network topology of Ethernet LANs, by using only generic layer-2 MIB information. Every algorithmic solution that resolves this challenge is required to address three fundamental sources of complexity.

(i) Inherent Transparency of Layer-2 Hardware. Layer-2 network elements, *i.e.,* switches, bridges, and hubs, are completely transparent to end-point hosts and layer-3 routers. Furthermore, they do not keep records of their layer-2 neighbors. The only state that layer-2 switches maintain is their *Address Forwarding Tables (AFTs)* that are used to forward incoming packets to the appropriate output port. Fortunately, most layer-2 elements (see (ii) below) make this information available through a standard SNMP MIB [1], [2].

(ii) Transparency of Dumb or Uncooperative Elements. Ethernet LANs deploy heterogeneous layer-2 switching elements with different capabilities. Some elements may not provide access to their AFT information, while other may be "dumb" elements, like *hubs*, even without MAC addresses. Since hubs do not communicate with other elements, they are, essentially, invisible to the other network elements and cannot be detected directly. Clearly, inferring the physical connections of hubs and "uncooperative" switches based on the limited AFT information obtained from other elements poses a non-trivial algorithmic challenge. An example of a LAN that contains a "dumb-hub" is illustrated in Figure 1-(a)

(iii) Multi-Subnet Organization. Large Ethernet LANs typically support *multiple subnets* that divide the network elements into groups. Elements in the same subnet can communicate directly with each other without involving a router, while communication between elements in different subnets must traverse through

(a) The network.

(b) The connecting tree of subnet N1.

(c) The connecting tree of subnet N2.

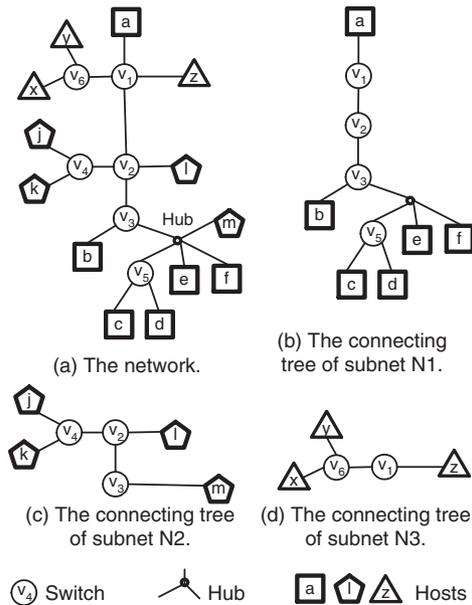(d) The connecting tree of subnet N3.

Fig. 1. An example of a network and its subnet connecting trees.

routers even if they are directly connected to each other. Consequently, packets generated by elements of a given subnet traverse mainly along paths of the subnet *connecting tree*, which is spanned by the nodes that belong to the considered subnet. As a result of subnets, an element can be completely invisible to its direct physical neighbors. For instance, consider the LAN pictured in Figure 1-(a). Its hosts are divided into three subnets, $N_1 = \{a, b, c, d, e, f\}$, $N_2 = \{j, k, l, m\}$ and $N_3 = \{x, y, z\}$, and their corresponding connecting trees are presented in Figures 1 (b)-(d), respectively. In this example, node v_1 is oblivious to the hosts in subnet N_2, while node v_2 is unfamiliar with the elements in subnet N_1, although these two nodes are adjacent.

B. Related Work

The need for an up-to-date layer-2 topology information has been recognized by both the telecommunication industry and the research community. A few infrastructure providers have already deployed their own proprietary protocols for discovering physical interconnections. Examples of such mechanisms include Cisco Discovery Protocol (www.cisco.com) and Extreme Discovery Protocol (www.extremenetworks.com). These tools utilize proprietary neighbor detection mechanisms which are not applicable in a heterogeneous, multi-vendor environment. Other topology discovery tools, such as Peregrine's Infratools software (www.peregrine.com) and Micromuse's Netcool/Precision application (www.micromuse.com), claim to provide layer-2 maps, but these tools are based on proprietary technologies that their details have not been published. Acknowledging the importance of maintaining accurate topology information, the IETF rectified in 2000 a new "physical topology" MIB [2] that records link layer interconnections. However, the proposal does not specify any protocol for populating these MIB entries. Recently, the *IEEE 802.1ab* committee finished its proposal for a new layer-2 discovery protocol, called *link layer discovery pro-*

tocol (LLDP) [3]. LLDP enables neighboring elements to notify each other of their presence and, consequently, to populate their "physical topology" MIBs [2]. This solution provides adequate information to infer the physical topology, however, LLDP cannot be easily deployed on *all* the legacy equipment. Consequently, it cannot easily be used at the huge installed base of heterogeneous Ethernet LANs of today. These industrial efforts, however, endorse the need for practical topology discovery tools for heterogeneous Ethernet LANs.

The challenge of physical topology discovery has also been addressed by the research community. Several studies [4], [5], [6], [7] introduced *network tomography* methods for inferring the network topology. These techniques use solely end-to-end measurements of packet drop rates or delay variance to infer the logical structure of an investigated network without collecting MIB information. These studies claim to provide relatively accurate logical graphs of the explored LANs. However, they require the deployment of monitoring software at the end-users and generating a large number of probe messages. Moreover, they cannot provide the actual mapping of network elements to logical nodes, which makes them less desirable for practical network management.

The most relevant results to our work are the three prominent studies in [8], [9] and [10] that rely solely on AFT information to infer the topology of Ethernet LANs. The considered topology discovery problem was first formulated by Breitbart *et al.* in [8]. They showed that the network topology may not be uniquely defined even if the switches have *complete* AFT information, *i.e.,* each switch has an AFT entry of the appropriate output port of every possible packet that it may receive. In such cases, of course, discovering the network topology is impossible. Then, they proposed an algorithmic solution for network settings that contain only cooperative switches with complete AFT information. For such cases, the study first presents necessary conditions that must be satisfied by any two directly connected elements (without any intermediate nodes), termed *valid union*. Then, it utilizes a matching algorithm to infer the network topology. Unfortunately, this approach suffers from two main drawbacks. Complete AFT information requires packet exchanges between every pair of elements in the same subnet, which in practice is very hard to obtain, if not impossible. Moreover, Ethernet LANs, typically, contain dumb-hubs and uncooperative switches that does not provide AFT information. Thus, for instance, the valid union approach cannot infer the topology of the LAN presented in Figure 1-(a).

In [9], Lowekamp *et al.* propose techniques for inferring node connectivity giving incomplete AFT information and they also address the presence of hubs. For their needs, they introduce the concept of simple connection, where a *simple connection* between any two network elements u, v is defined if it is known which port of u leads to element v and, vice versa, which port of v points to node u. Then they presented several rules to identify such simple connections and used them to construct a "divide and conquer" method that discovers the network topology. This method finds first a node that has simple connections to all the other network elements and uses it as a root node. The other elements are divided into subsets according to the ports of the root-node to which they have simple connections. The algorithm

repeats this process recursively until it infers the network topology. By implementation, the authors showed that the proposed solution, usually, works well for instances with a single subnet. Nevertheless, since this approach requires at least one node to have a simple connections with all the other network elements, it cannot easily be extended to the case of multiple subnets. As an example, the scheme fails to discover the topology of the network presented in Figure 1-(a) that contains only three subnets, *but without any element that is included in the connecting trees of all three subnets*. Moreover, the scheme does not perform any preliminary AFT populating process. Thus, its available AFT information depends on the user traffic and the AFT aging process[1], which impairs the scheme success probability.

Addressing the drawbacks of the previous methods, Bejerano *et al.* , presented in [10] a novel algorithm that considers the full generality of the problem, *i.e.,* it handles multiple subnets as well as dumb or uncooperative elements. The proposed method is based on an iterative *path refinement* process that gradually infers the element order along the path between every pair of elements. The study proved that this method has a strong completeness property that ensures the discovery of the physical topology, if it is uniquely defined by the AFTs. Although this study presents an interesting scheme, its contribution is mainly theoretical, because it is hard to implementation and it has high computational complexity. In addition, this algorithm also assumes complete AFT information for all the cooperative switches, which is hard to achieve. These, limitations of the above three methods raise the need for simple and practical new topology discovery scheme that considers the entire complexity of heterogeneous Ethernet LANs.

C. Our Contributions

We propose a novel *topology discovery* (TD) scheme for large heterogeneous Ethernet LANs with multiple subnets and dumb/uncooperative elements that uses only generic MIB information. Unlike the solution in [10], that guarantees completeness in the case of complete AFT information with the price of high computational complexity, we introduce a simple, efficient and practical method that determines the physical network topology with a very high probability, while using only limited AFT information. Our scheme consists of two stages. In the first stage, the scheme performs an AFT populating process, in which a single station sends `ping` messages to all the network elements. Then the scheme collects the AFT information by using SNMP queries. This simple process is sufficient to populate the switch AFTs with enough MAC addresses, which enables us to calculate the complete AFT information. This is a significant advantage over the methods described in [8] and [10] that require hard-to-obtain complete AFT information or the Lowekamp *et al.* method, [9], that relies on sporadic AFT information. In the second stage, the scheme invokes a *topology discovery* (TD) algorithm. The latter utilizes an auxiliary data structure, termed a *skeleton tree*, to represent the topology information detected so far. Initially, the TD algorithm uses skeleton trees to characterize the connecting tree of each subnet. Then, by performing a sequence of *merge operations*, it integrates pairs

of skeleton trees until it obtains a single tree that represents the network topology.

We evaluate the scheme characteristics from four different aspects: *correctness, complexity, completeness* and *practicality*. We formally prove that whenever the topology discovery algorithm ends with a single skeleton tree then this tree represents the accurate physical topology of the considered network. Moreover, in case of a single subnet LAN, the scheme accurately infers the network topology. Through rigorous complexity analysis, we prove that the running time of the topology discovery algorithm is $O(n^3)$, which is significantly lower than the other solutions. Then, by extensive simulations we show that the proposed scheme successfully infers the complete network topology in the vast majority of the cases and for practical network settings its success probability is tangent to 100%. This includes many instances where the methods presented in [8] and [9] have failed. We demonstrate the practicality of our solution by implementing the scheme and inferring the topology of various Ethernet LANs. Finally, we show that the proposed scheme can be used to infer the topology of virtual LANs (VLAN), [14], where each VLAN spans its own independent connecting tree.

II. THE NETWORK MODEL

In this study we consider a connected Ethernet LAN that is comprised of a large number of network elements, such as switches (also known as bridges), hosts and routers. Since the considered LAN is connected, there is a path between every pair of network elements that involves only layer-2 elements, *i.e.,* switches and hubs, that are including in the considered LAN. The LAN may have an arbitrary topology, however, we assume that the switches execute the *spanning tree protocol* [12] to determine their active ports. As a result, the active forwarding paths between the elements yield a tree topology within the explored LAN. We model this spanning tree as an *undirected tree*, $G(V, E)$, where node in V represents a network elements and each edge in E represents a physical connection between two active element ports. The spanning tree $G(V, E)$ is the target of our topology discovery algorithm and it is termed the *explored network* or simply the *network*. In this tree, switches are internal nodes while hosts and routers are represented as leaf nodes. Since, the latter are practically indistinguishable for the purposes of layer-2 topology discovery, we refer in the following to routers as hosts[2].

Packets are forwarded only along the links of the spanning tree and their routes are determined based on the *address forwarding table* (AFT) information obtained through backward learning. In other words, the AFT information of a given active port is comprised of the MAC addresses that have been seen as source addresses on packets received by this port. Since G is a tree, there is a single path in G between every pair of nodes $s, t \in V$, denoted by $P_{s,t}$. For every node $v \in V$, we denote by D_v its set of active ports. We use the notation (v, k) to identify the kth port of node $v \in V$, and $F_{v,k}$ to denote the set of AFT entries at port (v, k). We also denote by $v(u)$ the port of node v that leads to node u in G, *i.e.,* node $u \in F_{v,v(u)}$.

The set V is comprised of both *labeled* and *unlabeled* nodes. Labeled nodes, basically, represent elements that have a unique identifying MAC address and can provide AFT information through SNMP queries. Unlabeled nodes, on the other hand, represent both "dumb" hub devices or switching elements with no SNMP support. Thus, AFT information can be obtained only from labeled nodes. To simplify the discussion, we refer to labeled and unlabeled switching elements simply as *switches* and *hubs*, respectively and we consider all the hosts as labeled nodes.

Every labeled node in G is associated with one or more *subnets*. A subnet is a maximal set of network elements $N \subseteq V$ such that any two elements in N can communicate directly with each other without involving a router, while communication across different subnets must go through a router. Thus, a packet from node s to node t in the same subnet N traverses along the path $P_{s,t}$ in G. Let $\mathcal{N}$ be the collection of subnets of the graph G, where each subnet $N \in \mathcal{N}$ is defined by the set of its members. Typically, the subnet association of a node is determined by its IP address and a corresponding network mask. For example, IP address 145.112.47.10 along with mask 255.255.255.0 identifies a subnet of network elements with IP addresses of the form 145.112.47.x, where x is any integer between 1 and 254. Thus, a packet sent between the hosts with IP addresses 145.112.47.**10** and 145.112.47.**200** traverses along the directed path between these two hosts in the LAN. Though, a packet sent between the hosts 145.112.**47**.10 and 145.112.**40**.10 must traverse through a router.

We denote by N the set of labeled elements in a given subnet *or* the set of labeled elements of several subnets in $\mathcal{N}$. Every set N (with either members of a single subnet or elements in several subnets) defines a *connecting tree* in G, denoted by $T^N(V^N, E^N)$, which is a sub-tree of G spanned by the nodes in N. Essentially, $T^N(V^N, E^N)$ is actually the tree obtained by taking the union of all paths $\{P_{s,t}\}$ between every pair of nodes $s, t \in N$, where, V^N and E^N are the sets of nodes and edges in any such $P_{s,t}$ path, respectively. Recall that V^N contains the set N itself, *i.e.,* $N \subseteq V^N$. We assume that V^N does not represent hubs (unlabeled nodes).

Example 1: Figure 1-(a) depicts an example of network with 6 switches, one hub and 13 hosts. The hosts are divided into three disjoint subnets; $N_1 = \{a, b, c, d, e, f\}$, $N_2 = \{j, k, l, m\}$ and $N_3 = \{x, y, z\}$, where each switch is solely included in a separate subnet. The connecting trees of N_1, N_2 and N_3 are presented in Figures 1 (b)-(d), respectively. □

Since a switch may serve hosts in different subnets, for every switch $v \in V$ we denote by $\mathcal{N}_v \subseteq \mathcal{N}$ the collection of subnets that v is included in their connecting trees. Essentially, the AFTs of a given node v contain mainly reachability information for the nodes in the subnets in $\mathcal{N}_v$. We use the notation $F_{v,k}^N$ to denote all the AFT entries for nodes in $N \in \mathcal{N}_v$ that are included in the AFT $F_{v,k}$ of v. We also denote by D_v^N its set of active ports in T^N, *i.e.,* $k \in D_v^N$ if $F_{v,k}^N \neq \emptyset$. We say that the AFT $F_{v,k}^N$ of v is *complete for subnet N* if $F_{v,k}$ contains the MAC addresses of all nodes in N that are reachable by port (v, k). Similarly, the AFT $F_{v,k}$ of v is termed *complete* if it is completed for all the subnets $N \in \mathcal{N}_v$. In other words, the AFT $F_{v,k}$ contains all the MAC addresses of every node $v \in N$ accessible by port (v, k) for every subnet $N \in \mathcal{N}_v$ (v is included in its connecting tree).

Symbol	Semantics
$G(V, E)$	The explored network (spanning tree).
(v, k)	k^{th} port of node $v \in V$ ($v1, v2, ...$)
$F_{v,k}$	AFT entries at (v, k) (i.e., nodes reachable by (v, k))
D_v	The set of active ports of node v.
$v(u)$	The port of node v that leads to node u in G
$\mathcal{N}$	The set of subnets included in the network.
$\mathcal{N}_v$	The subnets that contain v in their connecting trees
$P_{s,t}$	The set of nodes along the path from s to t in G.
$T^N(V^N, E^N)$	The connecting tree of the set (subnet) N.
N	The set of nodes (subnets) that induce $T^N(V^N, E^N)$.
V^N	All the network elements in T^N (the set N and the other switching elements in T^N).
E^N	All the links included in T^N.
$F_{v,k}^N$	The AFT entries at (v, k) of the nodes in N.
D_v^N	The set of active ports of node v in the tree T^N.
r	The root of a connecting tree T^N.
B_v	The set of nodes in N that are included in the subtree of T^N rooted by v.
n_v	The number assigned to node $v \in V^N$.
X	The anchor nodes of the tree T^N.
$H(Y, A)$	The skeleton-tree of the connecting tree T^N.
C_y	The nodes in V^N represented by the vertex $y \in Y$.

TABLE I

NOTATION.

A summary of the paper main notation is given in Table I.

In this study, we assume, first, that all the AFTs are complete and we relax this requirement in Section IV-C. Although, our model *does not* explicitly consider virtual LANs (VLANs), as defined by the IEEE 802.1-Q standard [14], we extend our solution to support VLANs in Section VIII.

III. OVERVIEW OF OUR SCHEME

The goal of our *topology discovery* (TD) scheme is discovering the physical topology of the spanning tree, $G(V, E)$, embedded in the explored Ethernet LAN. The scheme relies only on AFT information provided by labeled nodes in G for the following two purposes:
(1) Detecting the direct physical connections between active ports of labeled elements.
(2) Identifying the existence of unlabeled nodes, *i.e.,* hubs, in G and their adjacent elements.
The TD algorithm consists of two phases. First, it determines the topology of the connecting-tree T^N of every subnet $N \in \mathcal{N}$ by using only AFT information of the nodes in T^N. Since, this information may be insufficient for determining the complete topology of T^N, it deploys an auxiliary data structure, termed a *skeleton-tree*. The later represents the connecting-tree topology within a certain degree of accuracy. After calculating all the subnet skeleton-trees, the algorithm iteratively merges these trees together. The *merge operation* enhances the topology knowledge until a single tree, that represents the complete network topology, is obtained.

A. The Skeleton-Tree Data Structure

We now define the *skeleton-tree* data structure. Consider a connecting-tree $T^N(V^N, E^N)$ of subnet $N \in \mathcal{N}$ and let us assume complete AFT information. We distinguish between two
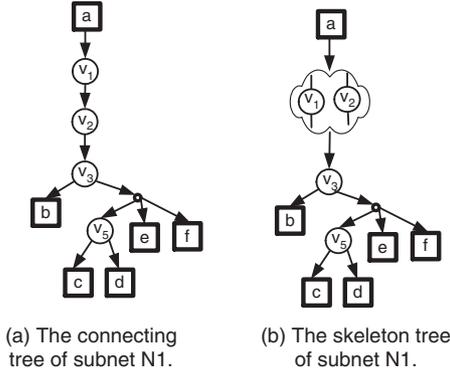
(a) The connecting tree of subnet N1.

(b) The skeleton tree of subnet N1.

Fig. 2. The connecting trees of subnet N_1 and its corresponding skeleton tree.

types of switches in T^N; The first are *junction* nodes with degree 3 or more, while the second are *transit* nodes with degree 2 in T^N. The latter can be divided into *segments* such that each one identifies a successive path between two junction nodes or a junction and a leaf node. For instance, consider the connecting tree of subnet N_1, depicted in Figure 2-(a). Here, v_1 and v_2 are transit nodes, while nodes v_3 and v_5 are junctions.

Definition 1: The *skeleton-tree* of the connecting-tree T^N, is a graph $H(Y, A)$ with a tree topology such that each *vertex* $y \in Y$ represents a set of nodes $C_y \subseteq V^N$ (C_y may be empty) and the set A of *arcs* represents the tree links. Each node $v \in V^N$ is included in exactly one set C_y, $y \in Y$. Each leaf or junction node in T^N is exclusively represented by a single vertex $y \in Y$ and a segment of transit nodes is identified by one or more vertices in Y. Moreover, the topology of T^N is obtained by replacing each vertex $y \in Y$ by the corresponding node or segment of nodes represented by the set C_y.

For the sake of clarity, we use different terminology to distinguish between the elements of a connecting tree T^N and its corresponding skeleton-tree $H(Y, A)$. Given that a *connecting tree* T^N is part of the explored network, we use the terms *nodes* and *links* to denote its network elements and their interconnections, respectively. We use the terms of *vertices* and *arcs* to denote the elements of the *skeleton tree* and their interconnections, accordingly.

From a graph theoretic perspective, the graphs T^N and $H(Y, A)$ are *homeomorphic*, [11]. In other words, the topology of T^N can be obtained from the graph H by subdividing arcs until every transit node is solely represented by a vertex. Similarly, the graph H can be obtained from T^N by smoothing away the corresponding transit nodes. Consequently, it is easy to see that there is a unique mapping from each node $v \in V^N$ to a single vertex $y \in Y$. However, a reverse mapping may not be uniquely defined. If there is a unique mapping from a vertex $y \in Y$ to a single node (network element) in T^N, then they both are called *anchors*. Thus, the network topology is uniquely defined when all the nodes are anchors.

Example 2: Consider the network depicted in Figure 1-(a). The skeleton trees of the subnets N_2 and N_3 are given in Figures 1-(c) and 1-(d), respectively. These two skeleton-trees provide the complete topology of the trees T^{N_2} and T^{N_3}. Figures 2-(a) and 2-(b) present the connecting tree of subnet N_1

and its corresponding skeleton-tree, respectively. In this example, the segment $\{v_1, v_2\}$ of transit nodes is represented by single vertex at the skeleton tree, while each one of the junction nodes v_3 and v_5 is represented by a unique vertex. Thus, v_3 and v_5 are anchor nodes. In the following, we use this tree to illustrate an execution of the skeleton-tree creation algorithm. Since our solution assume a tree root, Figure 2 presents directed trees rooted at node a. □

B. Overview of the STC Algorithm

We now provide a brief description of the *skeleton-tree creation* (STC) algorithm, which is an essential building block of our solution. *Initially, the STC algorithm is used to calculate the individual skeleton-tree for every subnet in $\mathcal{N}$. Then, it is used again by the merge operation to consolidate the individual skeleton-trees of the subnets into a single tree that represents the network topology.* We provide a detailed description of the STC algorithm in Section IV and we elaborate on the merge operation is Section V-B.

Our algorithm is based on a very simple property of the connecting tree $T^N(V^N, E^N)$. Let us first select an arbitrary root-node $r \in N$. Thus the connecting tree T^N becomes a directed tree rooted by node r. For each node $v \in V^N$, let $B_v \subseteq N$ denote the set of nodes in N that are also included in the *subtree* of T^N rooted by v and let $|B_v|$ denote its size. For instance, Figure 2-(a) presents a directed connecting tree rooted at node a. Since $N_1 = \{a, b, c, d, e, f\}$, in this example $B_{v_1} = B_{v_2} = B_{v_3} = \{b, c, d, e, f\}$ while $B_{v_2} = \{c, d\}$. As we describe below, the sets B_v, $v \in V^N$, can be easily calculated from the AFT information and they have the following property.

For every pair of nodes $v, u \in V^N$ such that u is a descendent of v, it follows that B_v subsumes the set B_u, i.e., $B_v \supseteq B_u$. Moreover, if v is a junction node or $v \in N$, then the set B_u is an explicit subset of B_v, i.e., $B_v \supset B_u$.

We use this observation to compile a list L of the nodes in V^N sorted in non-increasing order according to their $|B_v|$ values. In the case of a tie, transit nodes appear before the junction nodes that have the same $|B_v|$ value. This list defines an order relationship between the anchor nodes of the tree such that *each junction node and every node in N appears in L before each one of its descendents nodes and after its ancestor anchor nodes.* Thus, at any valid permutation of L, the root r is always the first and the leaves are at the end. This compelling property enables us to conduct a top-down discovery of the corresponding skeleton tree topology as we demonstrate in Example 3

Example 3: Consider the connecting tree T^{N_1} rooted at node a, as depicted in Figure 2-(a). At any valid permutation of L, the root a is the first node and it is followed by nodes v_1 and v_2. Since v_3 is junction node, it appears after the two transit nodes v_1 and v_2 and it is followed by node v_5. Finally, the leaf-nodes are at the end of the list. Thus, one feasible permutation of L is $L = \{a, v_2, v_1, v_3, v_5, e, b, d, c, f\}$. The STC algorithm extracts the nodes from L according to their order. First it removes the root a and creates a skeleton-tree with a single vertex that represents a, as shown in Figure 3-(a). Then, it extracts node v_2 and creates a new vertex for it, as shown in Figure 3-(b). In the following, the algorithm removes v_1 from L. Since v_1 and v_2 are transit nodes, the AFT information does not specify their pre-
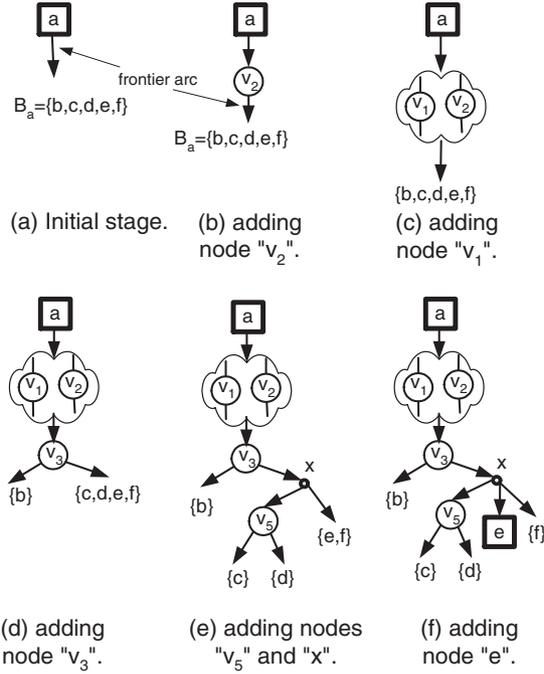
Fig. 3. An example of an execution of the skeleton-tree creation algorithm.

cise order inside the tree T^{N_1}. Therefore, v_1 is assigned to the same vertex of v_2, as depicted in Figure 3-(c). In Figure 3-(d), we show the skeleton-tree after adding node v_3. In the sequel, the algorithm extracts node v_5. Then, by observing discrepancies between the AFTs of node v_3 and the set B_{v_5}, it concludes the presence of an unlabeled node between v_3 and v_5 and adds a hub between these two nodes, as shown in Figure 3-(e). Finally, the algorithm adds the leaf-nodes as illustrated in Figure 3-(f). The final skeleton-tree is presented in Figure 2-(b). □

IV. THE SKELETON TREE CREATION ALGORITHM

This section provides a detailed description of our skeleton-tree creation (STC) algorithm. First, we present several basic properties that cast the theoretical foundations of our algorithm. Then we utilize them in our algorithmic description.

A. Basic Properties

Consider a set of nodes N (in one or more subnets) and let $T^N(V^N, E^N)$ be the directed connecting tree spanned by the set N with an arbitrary root node $r \in N$. Initially, we assume that the AFTs of all the nodes $v \in V^N$ are complete for the set N. We relax this requirement in Sub-section IV-C.

For each node $v \in V^N$, we term its port $v(r)$ that leads to the root as *root-port* and we refer to all its other active ports, $D_v^N - v(r)$, as *leaf-ports*. For every node $v \in V^N$, we define a set $B_v \subseteq N$ that contains all the nodes of N included in the subtree of T^N rooted by node v. Let us denote the size of this set by $|B_v|$. The set B_v can be easily calculated by taking all the leaf-port AFT entries of nodes in N. Then, adding node v to B_v if $v \in N$. Formally,

$$B_v = \bigcup_{k \in D_v - \{v(r)\}} F_{v,k}^N \cup (\{v\} \cap N) \qquad (1)$$

The sets $\{B_v\}$ have the following properties.

Property 1: For every node $v \in V^N$, $B_v \neq \emptyset$.
Proof: The tree T^N is the connecting tree of the set N. Thus, every leaf node of T^N is included in N. Since every sub-tree contains at least one leaf, every set B_v, $v \in V^N$, contains at least one node. □

Property 2: For every node v and any descendant u of node v, $B_v \supseteq B_u$ and $|B_v| \geq |B_u|$.
Proof: The sub-tree rooted at node u is included in the sub-tree rooted at node v. Thus, $B_v \supseteq B_u$. □

Property 3: For every node $v \in N$ and any one of its descendant $u \in V^N$, $B_v \supset B_u$ and $|B_v| > |B_u|$.
Proof: According to Property 2, $B_v \supseteq B_u$. Since, node $v \in N$ it is included in B_v but not in the sets B_u. □

Property 4: For every junction node $v \in V^N$ and any of its descendant $u \in V^N$, $B_v \supset B_u$ and $|B_v| > |B_u|$.
Proof: Since v is a junction, it has at least $k \geq 2$ children. From Property 1 each sub-tree rooted by one of node v's children contains at least one node in N. Thus for every descendant u of node v follows that $B_v \supset B_u$ and $|B_v| > |B_u|$. □

These properties enable us to establish order relationships between the nodes. We assign a value n_v to each node $v \in V^N$ as defined by Equation 2.

$$n_v = \begin{cases} |N| + 1/2 & : \quad \text{If } v = r \\ |B_v| - 1/2 & : \quad \text{Else if } v \in N \text{ or a junction} \\ |B_v| & : \quad \text{Otherwise.} \end{cases} \qquad (2)$$

Let L be a list of all nodes sorted in non-increasing order according to their n_v values. We say that *node v has* complete order relationship *if in any feasible permutation of L it appears after all its ancestors and before all its descendants nodes in the directed tree T^N rooted by r.*

Lemma 1: Every node in N and every junction node has complete order relationship.
Proof: From Equation 2, the root r is the first node in the list L, since its value $n_r = |N| + 1/2$. Now, let v be a node in N or a junction node. From Properties 3 and 4, it follows that $|B_v| \geq |B_u| + 1$ for every descendant u of node v. Since $n_v = |B_v| - 1/2$ and $n_u \leq |B_u|$, then $n_v > n_u$. By using similar arguments, the Lemma is also satisfied for every ancestor of node v that is included in N or it is a junction node. We only have to show that the Lemma is valid for any transit ancestor u of node v. In this case, $n_u = |B_u|$. Thus, from Property 3, $|B_u| \geq |B_v|$. Since, $n_v = |B_v| - 1/2$, the value of $n_u > n_v$. This completes the proof. □

In the cases that all the nodes have complete order relationship, the network topology is uniquely defined and can be calculated by the STC algorithm. However, complete order relationship is not guaranteed to transit nodes that are not included in N. Thus, a segment of transit nodes (not in N) is represented by single skeleton-tree vertex. This raises the need to divide transit nodes into successive segments. For this need, we use the following property.

Lemma 2: Consider two transit nodes $u, v \in V^N - N$ such that $B_u = B_v$, then every node x in the path between node u and v is also a transit node with $B_x = B_v$.
Proof: Let us assume for the sake of contradiction that this Lemma is not satisfied. Thus, there is a pair of transit nodes

u, v with $B_v = B_u$ such that the path between them contains a node x, which is either in N or a junction node. Let us assume w.l.o.g. that node u is a descendant of node v. Thus, x is a descendant of node v and an ancestor of node u. According to Properties 2, $B_v \subseteq B_x$ and $B_x \subseteq B_u$. Thus, $B_x = B_v$. Moreover, from Properties 3 and 4, node x cannot be neither in N or a junction node. So it must be a transit node. □

Corollary 1: Consider a transit node $v \in V^N$ and let $C \subset V^N$ be the set of *all* transit nodes such that for every $u \in C$, $B_u = B_v$. Then, all the nodes of C are included in a successive segment of transit nodes in the tree T^N.

Example 4: Consider the connecting tree T^{N_1} of the subnet N_1 with root node a, (as shown in Figure 2-(a)). In this case, the sets $B_{v_1} = B_{v_2} = B_{v_3} = \{b, c, d, e, f\}$. However, these nodes have different n_v values, *i.e.,* $n_{v_1} = n_{v_2} = 5$ while $n_{v_3} = 4.5$. Consequently, the nodes v_1 and v_2 always appear before node v_3 at any feasible permutation of L, as argued in Example 3 □

B. The Algorithm

The *skeleton-tree creation* (STC) algorithm computes a skeleton-tree $H(Y, A)$ that retains the topology knowledge of a given connecting tree $T^N(V^N, E^N)$. It is an iterative algorithm that starts with an empty skeleton-tree H and adds at each iteration a new labeled node $v \in V^N$ to H, until all the labeled nodes in V^N are represented in H. The algorithm receives as input the sets N and V^N, the root node $r \in N$, and AFT information, *i.e.,* $F_{v,k}^N$ for each $v \in V^N$. For its needs, it maintains a directed graph $H(Y, A)$ that denotes the skeleton tree calculated so far. Every vertex $y \in Y$ maintains a set $C_y \subset V^N$ of the nodes that it represents and a parameter n_y that denotes the n_v value of these nodes as defined by Equation 2. Recall that in the case that vertex y denotes a segment of transit nodes, then all the nodes $v \in C_y$ have the same n_v value. The arcs represent the known links in the considered tree. During the calculation, an arc may have only one known end-point, while its other end-point node has not been discovered yet. Such arcs are called *frontier arcs* and they are stored in a set, denoted by Z, until both their end-points are detected. For every arc $a \in A$ the algorithm keeps a set B_a with all the nodes in N reachable through this arc. The set B_a is actually the AFT, $V_{v,k}^N$, of the corresponding port k of a node $v \in C_y$. To clarify the details of algorithm, we demonstrate again an execution of the STC algorithm for calculating the skeleton tree given in Figure 2-(b).

During the initialization stage, the algorithm finds the root-port, $v(r)$, for each node $v \in V^N$ and calculates its set B_v and its value n_v, as defined by Equations 1 and 2. Then, it compiles a list L of the nodes $V^N - \{r\}$ sorted in non-increasing order according to their n_v values. In addition, it initializes a skeleton tree $H(Y, A)$ with a single vertex y that represents the root r, *i.e.,* $C_y = \{r\}$ and $n_y = |N| + 1/2$. The algorithm also originates a set Z with $|D_r^N|$ frontier arcs that denotes the incident links of the root-node r. Each arc $a \in Z$ is associated with a set $B_a = F_{r,k}^N$ for each one of the root's active ports in T^N, *i.e.,* $k \in D_r^N$. An example of the initialization stage is presented in Figure 3-(a) and in Figure 3-(b) node v_2 is added to H.

After the initialization stage, the algorithm iteratively extracts the first node from L, denoted by v', and modifies the skeleton-tree $H(Y, A)$, accordingly. First, it identifies the frontier arc

$a \in Z$ that will be connected to the newly-discovered node v'. Since a is the only frontier arc that represents a link along the path $P_{r,v}$ from the root r to node v' in T^N, $B_a \supseteq B_{v'}$. We use it to identify the corresponding arc a. In the following, let us denote with $y \in Y$ the known end-point (vertex) of a in H.

The algorithm distinguishes between two basic cases. If $n_y = n_{v'}$ (and they are integer numbers) then from Lemmas 1 and 2 follows that node v' is a transit node included in the segment represented by vertex y. In such case, node v' is added to C_y, as described in Example 5.

Example 5: In our example, nodes v_1 and v_2 are transit nodes with $B_{v_1} = B_{v_2} = \{b, c, d, e, f\}$ and $n_{v_1} = n_{v_2} = 5$. Consequently, they are represented by the same vertex y, as illustrated in Figure 3-(c). □

When $n_y > n_{v'}$ then node v' is represented by a new vertex y' in H. The vertex y' is associated with a set $C_{y'} = \{v'\}$ and a value $n_{y'} = n_{v'}$. For every leaf-port $k \in D_{v'}^N - \{v(r)\}$ of v', the algorithm creates a new frontier arc a' incident from vertex y', it associates the new arc a' with the set $B_{a'} = F_{v',k}^N$ and adds a' to Z. Now, if $B_a = B_{v'}$, there are no intermediate nodes between the node represented by vertex y and node v'. Thus, the algorithm connects arc a to the vertex y' and removes a from the frontier arc set Z, as described in Example 6

Example 6: Consider the addition of node v_3 to the skelton-tree, as shown in Figure 3-(d). Since, $n_{v_1} = n_{v_2} = 5$ while $n_{v_3} = 4.5$ (see Example 4), node v_3 is represented by a separate vertex y. However, $B_{v_1} = B_{v_2} = B_{v_3} = \{b, c, d, e, f\}$. Thus, we conclude that v_3 is directly connected to either node v_1 or node v_2. □

Now, suppose that $B_a \supset B_{v'}$. Consequently, we conclude that the parent of v' in T^N is an unlabeled node and we consider two situations. If, vertex y represents a labeled node, *i.e.,* $C_y \neq \emptyset$, then a new vertex x is created and inserted between the vertices y and y'. Since, x represents unlabeled node it is associated with a set $C_x = \emptyset$ and a value $n_x = |B_a| - 1/2$ (according to Equation 2). Vertex x is attached to two new arcs a_1, a_2 with $B_{a_1} = B_v$ and $B_{a_2} = B_a - B_v$. The algorithm connects vertex x to the arc a that incidents vertex y and attaches vertex y' to the arc a_1 of x. The arc a is removed from Z and the arc a_2 is inserted. An illustration of such situation is given in Example 7.

Example 7: Consider the skeleton tree presented in Figure 3-(d) and let v_5 be the newly discovered node represented by vertex y'. Note that vertex y represents node v_3 and its is associated with two frontier arcs. Since $B_{v_5} = \{c, d\}$, node v_5 should be connected to the frontier arc a with $B_a = \{c, d, e, f\}$. Since, $B_a = \{c, d, e, f\} \supset \{c, d\} = B_{v_5}$, The algorithm detected the hub between nodes v_3 and v_5. It adds a new vertex x that denotes the hub and connects x to the vertices y (represents v_3) and y' (denotes v_5). Thus, vertex x has one frontier arc a_2 with $B_{a_2} = \{e, f\}$. The computed tree is shown in Figure 3-(e). □

The second situation occurs when $B_a \supset B_v$ and vertex y represents hub (unlabeled node), *i.e.,* $C_y = \emptyset$. In this case, the hub is the direct parent of node v as well as some other nodes in T^N that have not been discovered yet. Consequently, the algorithm creates a new frontier arc $\hat{a}$ for y, which denotes the hub ports that lead to its undetected children. Thus, $B_{\hat{a}} = B_a - B_v$ and $\hat{a}$ is added to Z. In addition, it connects vertex y' to the

arc a (with $B_a = B_v$) and removes a from Z. Such a case is presented in Example 8.

Example 8: Consider the skeleton tree presented in Figure 3-(e) and let us assume that the algorithm has just extracted node e from the list L. Since vertex x has a frontier arc a with $B_a = \{e, f\}$, it follows that node e should be reachable through this arc. However, in this case B_a contains also node f. Thus, there is either another hub between node e and the hub represented by x, or x has another port that leads to f. In practice, we cannot distinguish between these two scenarios and we assume the presence of a single hub. Thus, the algorithm connects node e directly to vertex x and adds a new frontier arc $\hat{a}$ with $B_{\hat{a}} = \{f\}$, as depicted in Figure 3-(f). □

From Example 8 follows that two or more adjacent hubs in the actual network are represented by a single hub in our inferred solution.

At the end of the iteration, the algorithm extracts the next node v' from L, until L is empty. In our description we omit port-id information but this can easily incorporate to our algorithm. A formal description of the algorithm is given in Figure 4.

Theorem 1: Consider a set $N \subseteq V$, its corresponding connecting-tree $T^N(V^N, E^N)$ and any given root node $r \in N$. Then, the STC algorithm computes a skeleton-tree $H(Y, A)$ of T^N, where every node $v \in N$ and every junction node in T^N is represented by anchor vertices in Y.

Proof: The nodes in L are added to the skeleton-tree $H(Y, A)$ according to their order in L. From Lemma 1, every node $v \in N$ or a junction node in T^N is inserted to H after its ancestors and before its descendants in T^N. Node v is attached to the arc $a \in Z$ such that $B_a \supseteq B_v$ therefore its is placed in its correct position in the tree. Note that the algorithm inserts unlabeled nodes when $B_a \neq B_v$. By using Lemma 2, the algorithm groups together transit nodes with the same B_v sets and represents them with a single vertex in Y. Thus, the graph $H(Y, A)$ is a valid skeleton tree of T^N. □

C. Relaxing The Complete AFTs Requirement

We now replace the hard-to-obtain complete AFT requirement with much simpler requirement that can be easily obtained. As defined in Section III, the set B_v of every node $v \in V^N$ contains only AFT entries of its leaf-ports. Thus, only the leaf-ports must be completed for the set N, while the root-port AFTs need to include only the address of the root-node. The later is used just for identifying the root-ports of the nodes. This requirement is formulated as follows,

Requirement 1: Consider the connecting tree $T^N(V^N, E^N)$ of a given set N with a root node r. For every node $v \in V^N$, the AFT of the root-port, $v(r)$ must include the address of the root-node r and the AFTs of all the leaf-ports, $D_v^N - \{v(r)\}$, must be complete for the set N.

Requirement 1 can be easily achieved by sending probe messages from a single point, as we show in Section V-A.

D. Calculating Extended AFTs

We argued in Section IV-C that a skeleton tree can be calculated without complete AFTs. More specific, for a given root node, it is sufficient if every root-port AFT contains the root-node address and only leaf-port AFTs are complete. Such AFT

```
procedure SKELETONTREE(N, V^N, r, AFTs)
Initialization:
1.  for every node v ∈ V^N − {r} do
2.      B_v = ⋃_{k∈D_v−{v(r)}} F^N_{v,k} ⋃ ({v} ⋂ N)
3.      if v ∈ N or |D_v^N| ≠ 2 then n_v = |B_v| − 1/2
4.      else n_v = |B_v|
5.  end for
6.  L = Sorted list of V^N − {r} in non-increasing
        order according to their n_v values.
7.  Create a new vertex y, with C_y = {r}
8.  n_y = |N| + 1/2.
9.  for every port k ∈ D_r^N do
10.     Create an outgoing arc a for y, B_a = F^N_{r,k}.
11.     Z = Z ⋃ {a}
12. end for

Main Loop:
13. while L ≠ ∅ do
14.     v' = get and remove the first node in L.
15.     Find arc a ∈ Z s.t. B_a ⊇ B_{v'}.
16.     y = the starting-point of a in Y.
17.     if n_y = n_{v'} then
18.         C_y = C_y ⋃ {v'}
19.     else
20.         Z = Z − {a}
21.         Create a new node y' with C_{y'} = {v'}.
22.         n_{y'} = n_{v'}
23.         for every port k ∈ D_v^N − {v(r)} do
24.             Create an outgoing arc a' for y'.
25.             B_{a'} = B_{v',k}
26.             Z = Z ⋃ {a'}
27.         end for
28.         if B_a = B_{v'} then
29.             Connect node y' to arc a.
30.         else if C_y = ∅ then
31.             Create an outgoing arc â for y.
32.             B_â = B_a − B_{v'}.
33.             Z = Z ⋃ {â}
34.             Connect node y' to arc a.
35.             B_a = B_{v'}
36.         else
37.             Create node x with C_x = ∅.
38.             n_x = |B_a| − 1/2
39.             Create two outgoing arcs a_1, a_2 from x,
                    with B_{a_1} = B_{v'} and B_{a_2} = B_a − B_{v'}
40.             Z = Z ⋃ {a_2}
41.             Connect node x to arc a.
42.             Connect node y' to arc a_1.
43.         end if
44.     end if
45. end while
46. return H(Y, A)
```

Fig. 4. A formal description of the STC Algorithm.

---

information that satisfies Requirement 1 can be easily obtained for the case of a single subnet, as described in Section V. However, the required AFT information may not be available for using the STC algorithm to merge two skeleton-trees. To overcome this deficit, we present an auxiliary algorithm, refer to as the EXTENDEDAFTS routine, that augment the AFT information collected from the network elements with additional entries that are required for merge operations. We refer to this additional information as *extended AFTs*.

Consider a skeleton-tree $T^N(V^N, E^N)$ and let $X \subseteq V^N$ be the set of anchor nodes in the tree T^N. The AFT augmentation ensures that the AFTs of the nodes in T^N are complete for the set X. Our algorithm is based on the observation that T^N

```
procedure EXTENDEDAFTs(y, X, H(Y, A))
1.  for every child y_j, j ∈ [1..J] of y do
2.      X_{y_j} = EXTENDEDAFTs(y_j, X, H(Y, A))
3.  end for
4.  X_y = (⋃_{j=1}^{J} X_{y_j}) ⋃ (X ⋂ C_y)
5.  for every node v ∈ C_y do
6.      F_{v,v(r)} = F_{v,v(r)} ⋃ (X − X_y)
7.      for every port v, k_j, j ∈ [1..J] do
8.          F_{v,k_j} = F_{v,k_j} ⋃ X_{y_j}
9.      end for
10. end for
```

Fig. 5. A formal description of the EXTENDEDAFTs routine.

uniquely specifies the location of every anchor node $x \in X$. In other words, for every anchor node $x \in X$ and every node $v \in V^N$, the skeleton-tree T^N uniquely define the port $v(x)$ of v that leads to node x. The extended AFTs are calculated in a recursive manner by performing a post-order tour [11] on the skeleton-tree, starting with the root r. In this tour, it calculates the set of anchor nodes X_y of every subtree of H rooted by a vertex $y \in Y$. Then, it uses these sets to augment the nodes' AFTs.

We now elaborate on the EXTENDEDAFTs routine. Consider a vertex $y \in Y$ and let us denote its children in H by $y_1, y_2, \cdots, y_J \in Y$. During the post order tour, each child y_j returns to its parent y the set of anchor nodes X_{y_j} that are included in its subtree. Clearly, if y is a leaf vertex in $H(Y, A)$ then the algorithm does not perform recursive calls. After obtaining all the sets $\{X_{y_j}\}$ from each child y_j of y, the routine calculates the set X_y of anchor nodes in the subtree of vertex y. To this end, it takes the union of all the sets X_{y_j} and if y is an anchor vertex, then it also adds the node represented by y (the set C_y), *i.e.*, $X_y = (⋃_{j=1}^{J} X_{y_j}) ⋃ (X ⋂ C_y)$. Now, the routine extends the AFTs of the nodes represented by vertex y, *i.e.*, the nodes in C_y. For every leaf-port k of every node $v \in C_y$, it matches the corresponding child y_j and adding the set X_{y_j} to its AFT. Then, the root port $v(r)$ of every node $v \in C_y$ is augmented with the set $X − X_y$. A formal description of the routine is given in Figure 5.

Theorem 2: Consider a connecting tree $T^N(V^N, E^N)$ and its corresponding skeleton tree $H(Y, A)$. Let X be the set of anchors in both trees. Then the EXTENDEDAFTs routine calculates the complete AFTs for the set X for every node $v \in V^N$.

<u>Proof:</u> This theorem results directly from the post-order tour that the algorithm performs over the skeleton-tree. □

V. THE TOPOLOGY DISCOVERY SCHEME

We now elaborate on our *topology discovery* (TD) scheme. The scheme is typically deployed on a *network management station* (NMS) for discovering the topology of given LANs. Let us denote by $\mathcal{N}$ the set of subnets that comprise the explored LAN and let $T^{N_i}(V^{N_i}, E^{N_i})$ be the connecting tree of every subnet $N_i \in \mathcal{N}$. The TD scheme, initially, collects the required AFT information and then it executes a *topology discovery* (TD) algorithm. Below, we elaborate on these two phases and a formal description of the TD scheme is given in Figure 6.

```
procedure TOPLOGYDISCOVERY(V, 𝒩)

Obtaining the required information:
1.  Send a ping message to each node v ∈ V.
2.  Collect the AFT information from each node v ∈ V.
3.  for every subnet N_i ∈ 𝒩 do
4.      Identify its root node r_i ∈ N_i.
5.      Identify the set of nodes V^{N_i} of its connecting tree.
6.  end for

Inferring the network topology:
7.  for every subnet N_i ∈ 𝒩 do
8.      H_i(Y_i, A_i) = SKELETONTREE(N_i, V^{N_i}, r_i, AFTs)
9.      X_i = The set of anchor of H_i
10.     Let y_i be the vertex of H_i that represents r_i.
11.     EXTENDEDAFTs(y_i, X_i, H_i(Y_i, A_i))
12. end for

13. while (there are two skeleton trees
        H_i and H_j with X_i ⋂ X_j ≠ ∅) do
14.     N_k = N_i ⋃ N_j
15.     r_k = Any node in X_i ⋂ X_j.
16.     V^{N_k} = V^{N_i} ⋃ V^{N_j}
17.     H_k(Y_k, A_k) = SKELETONTREE(N_k, V^{N_k}, r_k, AFTs)
18.     Let y_k be the vertex of H_k that represents r_k.
19.     EXTENDEDAFTs(y_k, X_k, H_k(Y_k, A_k))
20.     Remove H_i and H_j
21. end while
22. return H_k(Y_k, A_k) (or all calculated skeleton trees)
```

Fig. 6. A formal description of the TD scheme.

A. Collecting The Required information

For calculating the LAN topology our scheme is required to gather the following information for every subnet $N_i \in \mathcal{N}$:
(1) The set of nodes N_i included in the considered subnet.
(2) The set V_i^N of nodes included in the corresponding connecting-tree T^{N_i}.
(3) A root node r_i.
(4) AFTs that satisfy Requirement 1.

Input - The scheme receives as input the *network-ids* and the *network-masks* of all the subnets in $\mathcal{N}$. Then, by using appropriate SNMP quires, it obtains the required AFT information.

Collecting the nodes' MAC and IP addresses - The NMS sends `ping` messages to all the possible IP addresses defined by network-id domains. Each `ping-reply` message identify a network element and its IP and MAC addresses. By sending appropriate SNMP queries to each detected element, the NMS infers the element type and if it has additional addresses (e.g. switches or routers). This enables us to map MAC addresses and IP addresses to a specific network element (regardless the number of ports and addresses that it has). Thus, in the following we assume that each node has a single identification number that can be obtained from its IP or MAC addresses.

Selecting root-nodes - For every subnet N_i, the scheme selects a root node r_i. We considered two cases. If the NMS is included in the subnet N_i, then the NMS itself serves as the root node r_i. Otherwise, NMS uses the *designated router* of subnet N_i as the root node. The later is used by the nodes in N_i for their inter-subnet communication. Thus, every message, sent by the NMS to any node $v \in N_i$, traverses through the designated router and a reply is forwarded on the reverse path. The designated router can be easily found, *e.g.*, by utilizing `traceroute`, [13].

Populating the AFTs - As mentioned in Section II, the AFT entries of the switches are seeded through backward learning on active ports. In other words, the AFT information of a given port is comprised of the MAC addresses that have been seen as source addresses on packets received by this port. We exploit the backward learning property in our *AFT populating process*. Lets assume that the NMS sends a `ping` message to a node $v \in N_i$. This message passes through the root node r_i of subnet N_i and traverses along the single path $P_{r_i, v}$, between r_i and v. Then, a reply is forwarded along the reverse path. These two messages populates the AFTs along this path with the MAC addresses of r_i and v, accordingly. Consequently, by sending `ping` messages to every node $v \in N_i$ from the NMS, the scheme populates the leaf-ports of all the nodes in T^{N_i} with complete AFT information for the set N_i. Moreover, each root-port is seeded with the address of the root node r_i. This ensures that the AFTs of the connecting tree T^{N_i} satisfy Requirement 1 for subnet N_i.

Collecting the AFT information - The scheme collects the AFTs by using standard SNMP queries and infers the set V^{N_i} of the nodes included in T^{N_i}. This set contains all the nodes in N_i and all the switching elements in the connecting tree T^{N_i}. Recall that every such switch has at least two active ports in T^{N_i}. Thus, it can be identified by discovering two of its ports that have AFT entries for nodes in N_i.

Theorem 3: Consider a subnet N_i, its connecting tree $T^{N_i}(V^{N_i}, E^{N_i})$ and its root node $r_i \in N_i$ (as determined above). Then, the AFT populating process ensures that the AFTs of every node $v \in V^{N_i}$ satisfy Requirement 1.

B. The Topology Discovery Algorithm

The *topology discovery* (TD) algorithm comprises of two stages. In the first stage, it calculates a skeleton tree for each subnet $N_i \in \mathcal{N}$ by invoking the STC algorithm (presented in Section IV-B) with the following parameters:
(1) The nodes of the considered subnet, N_i.
(2) The root, r_i, of the connecting tree as determined in Section V-A.
(3) The set V^{N_i} of the connecting tree nodes.
(4) The collected AFT information.
After calculating the skeleton tree, the algorithm uses the EX-TENDEDAFTs routine to augment the AFTs with the anchor set X_i of every connecting tree T^{N_i}.

In the second stage, the TD scheme uses the STC algorithm again to merge pairs of skeleton-trees, until a single skeleton-tree is obtained or the skeleton-trees cannot be merged any more. Let $H_i(Y_i, A_i)$ and $H_j(Y_j, A_j)$ denote the skeleton-trees of two different node sets N_i and N_j, where each not set, N_i or N_j, may contain members of a single subnet or several subnets. Let X_i and X_j be their anchor sets, respectively. The TD algorithm can merge these skeleton-trees if they have a *common anchor node*, which is a node $r \in X_i \bigcap X_j$. After detecting two skeleton trees with a common anchor, the TD algorithm merges them by calling to the STC algorithm with the appropriate parameters:
(1) The node set $N = N_i \bigcup N_j$.
(2) The common anchor node $r \in X_i \bigcap X_j$ as a root node.
(3) The set of nodes $V^N = V^{N_1} \bigcup V^{N_2}$ that comprises all the node of the connecting tree T^N.

(4) The augmented AFT information.

As we prove below, the provided AFTs satisfy Requirement 1 for the set N and the root r. The resulting skeleton tree represents the connecting tree spanned by the set $N = N_i \bigcup N_j$. After the merge operation, the TD algorithm augments the AFTs with the anchor set of the new tree and seeks for another pair of skeleton trees with common anchors.

The TD algorithm fails only when it cannot find any more pairs of skeleton trees with a common anchor node. As shown in Section VII, the probability of such event is very low.

Example 9: Consider the three skeleton trees spanned by the subnets N_1, N_2 and N_3 (see Figures 2-(b), 1-(c) and 1-(d) above). The skeleton trees of subnets N_1 and N_2 can be merged by using node v_3 as a common anchor. In the resulting skeleton-tree, nodes v_1 and v_2 become both anchor nodes. Then, the new skeleton tree $T^{N_1 \bigcup N_2}$ is merged with the skeleton tree of subnet N_3 where node v_1 is used as a common anchor. This merge yields the complete network topology. □

VI. THE ALGORITHM ANALYSIS

We now prove the *correctness* of the TD algorithm and to calculate its *complexity*. Due to space limitation, we provide only the main properties and the omitted proofs can be found in [15]. Our correctness analysis relies on the correctness of the STC algorithm and the AFT populating process proven in Theorems 1 and 3. First, we consider the skeleton-trees of the subnets.

Theorem 4: The TD algorithm calculates a feasible skeleton tree for every subnet $N_i \in \mathcal{N}$.

We now show the correctness of the merge operation. Consider two skeleton trees $H_i(Y_i, A_i)$ and $H_j(Y_j, A_j)$ that represent the connecting trees of the sets N_i and N_j. Lets assume that their anchor sets X_i and X_j contain a common anchor, denoted by r. Moreover, let $N = N_i \bigcup N_j$ and let $V^N = V^{N_i} \bigcup V^{N_j}$, as calculated by the algorithm. We now prove that the AFTs of the nodes V^N satisfy Requirements 1 for the set $N \cup \{r\}$.

Lemma 3: Consider a node $v \in V^{N_i} - V^{N_j}$. Then port $v(r)$ leads to all the nodes of V^{N_j}.

Lemma 4: The AFTs of the nodes V^N satisfy Requirement 1. From Theorem 1 and Lemmas 3 and 4 follows,

Theorem 5: Consider two skeleton-trees with a common anchor node that represent the connecting-trees T^{N_i} and T^{N_j}. Then, the outcome of their merge operation is a valid skeleton-tree that represents the connecting tree spanned by the set $N = N_i \cup N_j$.
From Theorems 4 and 5 we conclude Theorem 6

Theorem 6: If the topology discovery (TD) scheme ends with a single skeleton tree, then its result is a valid skeleton tree of the explored network.
From Theorem 6 follows,

Corollary 2: If the TD algorithm ends with a single skeleton-tree and the degree of each switch is at least 3, then this skeleton tree contains only anchor nodes and it represents the actual network topology.

Corollary 3: If the explored network comprises a single subnet and the degree of each switch is at least 3, then the TD scheme can infer the network topology.

We now present the algorithm complexity. We compute first the running time of the STC algorithm and EXTENDEDAFTS routine.

Lemma 5: the running time of the STC algorithm is $O(|V|^2)$.

Lemma 6: The running time of the EXTENDEDAFTS routine is $O(|V|^2)$.

From Lemmas 5 and 6 follws,

Theorem 7: The running time of the topology discovery algorithm is $O(|V|^3)$.

VII. SIMULATION RESULTS

We now address the *completeness* aspect of the TD scheme. As shown in [8], the AFT information does not always define a unique network topology, even if the AFTs are complete. Consequently, our scheme provides a complete solution only if (i) the AFTs define a unique network topology and (ii) the TD algorithm can merge all the subnet skeleton trees into a single one. However, by extensive simulations, we validate that our scheme infers the complete network topology in the vast majority of the cases included many cases that other schemes fail.

A. The Simulation Settings

We generated a large number of random networks with different numbers of switches, hosts, hubs and subnets. In order to imitate practical network topologies, each simulated instance has a hierarchical tree structure and it was constructed as follows; Initially, we generated a random tree of switches that defines the switches' connectivity. Then, we randomly connect the required number of hubs to free switch ports and we uniformly attached hosts to the switches and the hubs of the constructed tree. In the following, we arbitrarily associated each host with a single subnet-id and we connected the constructed tree to a router with a dedicated port for each subnet.

After determining the instance topology, we populated the AFTs by emulating transmissions of `ping` messages from the router to each one of the hosts. We assumed that each message traversed through the unique path between the corresponding router-port and the selected host and the reply message was forwarded along the opposite direction. Then, we collected the AFT information and used our scheme to infer the network topology. Finally, we checked if the result is a single skeleton-tree that represents the complete network topology.

We considered two types of switches, one with 8 ports and the second with 24 ports. In each instance, we either used 8-port switches or 24-ports switches but not both. We also assumed that the dumb-hubs contain only 8 ports. In addition, we distinguished between *cooperative* and *uncooperative* switches. While, the first provide their AFT information to the TD scheme, the later do not response to SNMP quires. Consequently, one of the main goals of our simulations is to evaluate the ability of the TD scheme to infer the network topology, with different numbers of uncooperative switches and subnets in the network.

B. Simulation Results of Our Scheme

Our simulations evaluate the *failure probability* (in percentages) of the proposed TD scheme to infer the complete topology of the simulated networks. We considered as failure all the
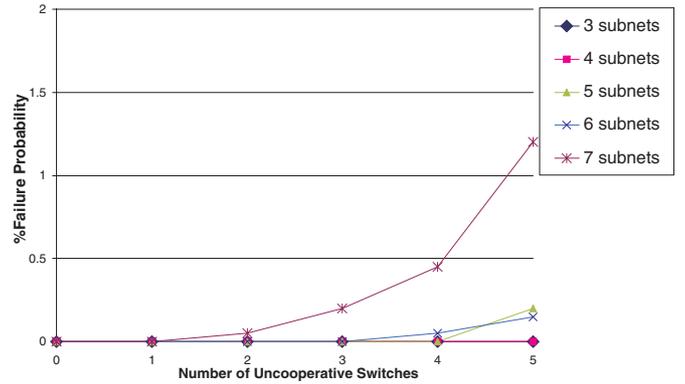

Fig. 7. Simulation results of networks with 10 switches (with 8 ports), 10 dumb-hubs (with 8 ports) and 100 hosts.

instances where the TD scheme could not infer a single skeleton tree of the simulated network, either where the AFTs don't define unique topology or when the TD scheme could not find a common anchor node. We provide typical results of our simulations in Figures 7-10, where each chart considers network instances with given numbers of switches, hubs and hosts. Each chart represents the scheme *failure probability* as the number of subnets and uncooperative switches increase. Each point results from topology discovery attempts of 2000 random instances with the same characteristics (number of switches, hubs, hosts, subnets and uncooperative switches). For the sake of clarity, we add lines between the points to show the trends that result from changing the fraction of uncooperative switches in the simulated networks, when the number of subnets is fixed.

Figures 7-8 consider networks that are comprised only of 8-port switches and 8-port hubs. Each instance contains the same number of switches and hubs, but the number of host is 10 times the number of switches. Thus, on average 5 hosts are associated with each switch and hub. Similarly, Figures 9-10 consider networks that are comprised only of 24-port switches and 8-port hubs. Also here, each instance contains the same number of switches and hubs, however, the number of hosts is 20 times more than the number of switches. In these instances the average number of hosts that are attached to a switch is 15, while only 5 hosts are attached on average to a hub. In our simulations, we considered the affect of different subnet sizes (starting with few large subnets and ending with an average number of 15 hosts per subnet) and different number of uncooperative switches.

Our simulations indicate that the TD scheme always found the topology when the explored networks had small number of uncooperative switches, *i.e.,* tangent to 100% success probability. This is itself an impressive result, since in all of the evaluated instances half of the switching elements were dumb-hubs that do not provide AFT information. Moreover, the simulations also reveal that the success probability was always above 99.75% when the average number of hosts in a subnet was 20 or more. This high success probability remained also when the number of uncooperative switches was 50%. Recall that in these cases, 75% of the network switching elements are uncooperative and only 25% of the switching elements provide AFT information.
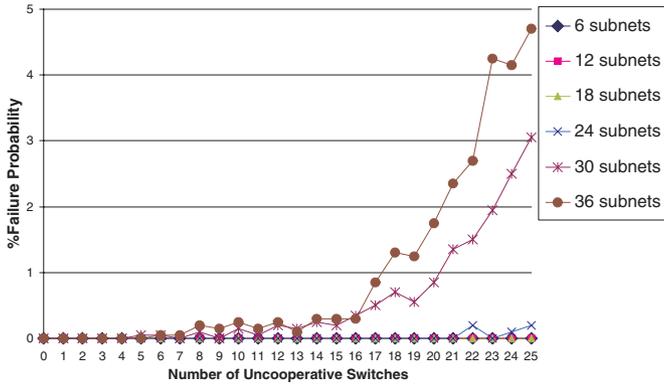
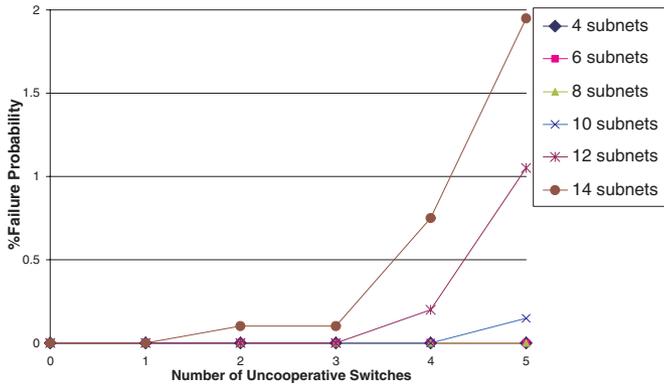Fig. 8. Simulation results of networks with 50 switches (with 8 ports), 50 dumb-hubs (with 8 ports) and 500 hosts.

Fig. 10. Simulation results of networks with 50 switches (each one with 24 ports), 50 dumb-hubs (each one with 8 ports) and 1000 hosts.

Fig. 9. Simulation results of networks with 10 switches (each one with 24 ports), 10 dumb-hubs (each one with 8 ports) and 200 hosts.

Fig. 11. Simulation results of the LHG shceme for networks with 50 switches (each one with 8 ports), 50 dumb-hubs and 500 hosts.

The simulations show that also in extreme cases, where the average subnet size is 15 hosts or less and only 25% of the switching elements (switches and dumb-hubs) are cooperative, the success probability of the TD scheme is above 95%. These remarkable results indicate that even with very limited AFT information the success probability of the proposed TD scheme is still very high.

C. Comparison with other Schemes

We also compared our scheme with other AFT-based TD schemes. In this comparison, we considered only the methods in [8], [9] that were implemented and are known as "practical". We first considered the Breitbart *et al.* scheme presented in [8]. This scheme can infer the network topology only when it has the complete AFT information from all the switching elements, *i.e.,* no hubs in the network. Thus, it cannot infer the topology of the simulated networks where half of the switching elements are hubs. In the journal version, the authors extended their scheme to detect hubs in the case of a single subnet, if they are located between a single switch and several hosts. This limited solution cannot address uncooperative switches. So, it cannot find the topology of the vast majority of the simulated instances.

We also simulated the scheme of Lowekamp *et al.* , [9], denoted by LHG. This scheme initially identifies simple connections between the network elements. Here, a simple connection
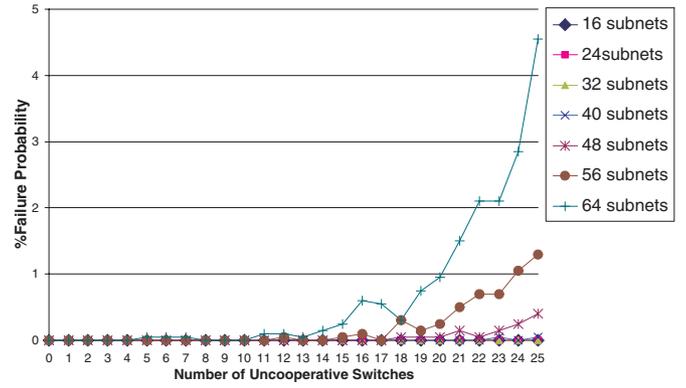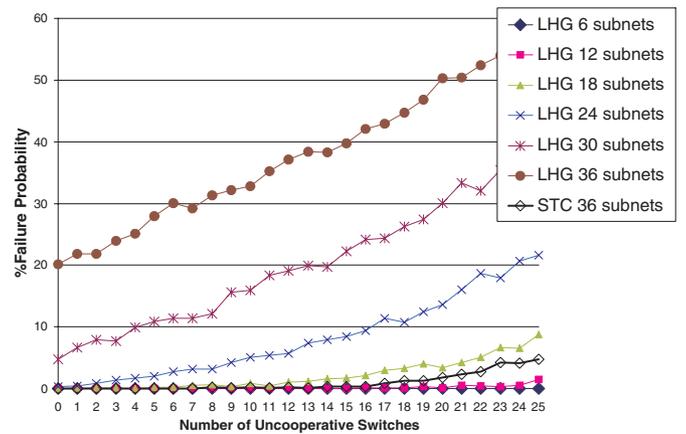
between a pair of elements u, v is defined if the port of u that leads to element v and the port of v that points to node u are known. Then, the scheme checks if there is any "root-node" that has simple connections with all the other elements. Finally, The scheme utilizes a "divide and conquer" algorithm. The later uses the simple connections of the root-node to divide the network elements into subsets and recursively infers the network topology. Consequently, the LHG scheme succeeds in discovering the network topology only when such a root-node is detected.

For a fair comparison, we evaluated the success probability of the LHG scheme for the same simulation instances that we used to evaluate our own scheme (see Section VII-A above). The LHG scheme does not perform any preliminary AFT populating process and it relies on sporadic information when reading the AFTs. However, in our simulations we seeded the AFTs with complete information by emulating transmissions of `ping` messages between every pair of nodes in the same subnet. Clearly, using complete AFTs increases the success probability of the LHG scheme. Thus, *our simulation results can be viewed as upper bounds of the LHG scheme success probability.*

Typical simulation results of the LHG scheme are presented in Figures 11 and 12, for networks with 50 switches and 50 hubs.
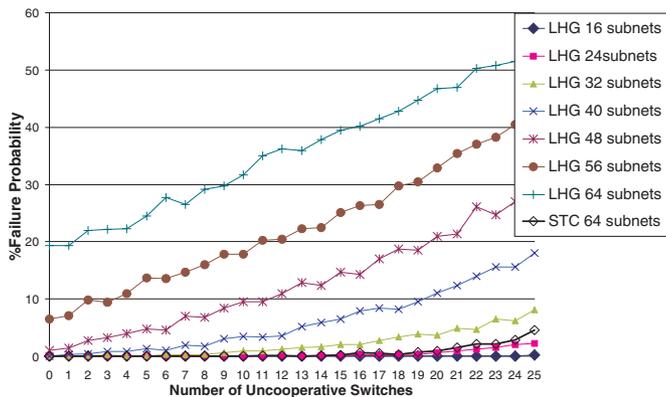
Fig. 12. Simulation results of the LHG scheme for networks with 50 switches (each one with 24 ports), 50 dumb-hubs and 1000 hosts.

In Figure 11 the simulated instances are comprised of 8-ports switches and 500 hosts, while in Figure 12 the networks contain 24-ports switches and 1000 hosts. The two charts show that, in general, the LHG scheme has high success probability in the cases of few subnets with large number of hosts. However, the success probability rapidly drops when the number of subnets increases and the average subnet size decreases. For instance, the scheme has only success probability of 80% for instances with average subnet size of 15 hosts when only cooperative switches are used (when simulating both 8-port and 24-port switches). The success probability significantly declines when the number of uncooperative switches increases. For comparison, we add to each one of Figures 11 and 12, a graph that indicates the success probability of our scheme, denoted with "STC", with average subnet size of 15 hosts. Thus, the two figures clearly demonstrate the superiority of our TD scheme over the LHG method.

VIII. IMPLEMENTATION AND VLAN TOPOLOGY DISCOVERY

Finally, we implemented the TD scheme to demonstrate its practicality and we were able to discover the network topology of the Computer Science Dept. of Bell-Labs in Murray Hill. Due to space limitation, we omit the implementation details and they can be found in [15]. We now present few useful observations for topology inference of Ethernet networks with virtual LANs (VLANs), [14], that we have learnt from our experiments with the TD scheme.

A typical skeleton-tree of a subnet - We investigated the structure of the skeleton-tree of each individual subnet in our LAN. We found that, in general, such skeleton-tree spans only a branch of the LAN spanning tree with only few switches. But, due to the switches high degree, they usually serve as anchor nodes of the corresponding skeleton tree. In other words, the skeleton trees typically provide the complete topology of the subnet connecting trees. Accordingly, in many cases the network topology can be obtained by taking the union of all the detected links.

Virtual Local Area Networks (VLANs) - Conceptually, VLANs provide similar functionality as subnets. They partition the hosts into separate groups, such that a message sent by a host in one VLAN to a host in another VLAN must traverse through a router. However, unlike subnets, that share the same spanning tree, each VLAN may have its own independent spanning tree [14] and a union of these trees does not necessarily yield a tree structure. Consequently, our scheme can calculate the skeleton-tree of each VLAN, without the ability to merge them. However, as mentioned above, the calculated skeleton-trees are frequently comprised only of anchor nodes. Thus, the network topology can be obtained from the union of the calculated skeleton-trees.

IX. CONCLUSION

This study addresses the challenge of inferring the topology of large multi-subnet Ethernet LANs. Since, the network topology cannot easily be obtained from the MIB information of the elements, it can be considered as the network secret. To this end, we presented a simple and efficient topology discovery scheme that utilizes a novel data structure, termed a skeleton-tree. The scheme infers the topology of each individual subnet with a certain degree of accuracy and performs a sequence of merge operations to discover the complete network topology. In other words, the scheme uses skeleton trees to reveal the network secret hidden in the switch closets. Figuratively, it can be viewed as *Taking the skeletons out of the closets*.

X. ACKNOWLEDGEMENT

REFERENCES

[1] W. Stallings, *"SNMP, SNMPv2, SNMPv3, and RMON 1 and 2"*, Addison-Wesley Longman, Inc., 1999, (3rd Edition).

[2] A. Bierman and K. Jones, "Physical Topology MIB," Internet RFC-2922 (www.ietf.org/rfc/), Sept. 2000.

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

[4] R. Caceres, N.G. Duffield, J. Horowitz, D. Towsley and T. Bu, "Multicast-Based Inference of Network-Internal Characteristics: Accuracy of Packet Loss Estimation", in *Proc. of IEEE INFOCOM'99*, Mar. 1999.

[5] T. Bu, N. Duffield, F. Lo Presti and D. Towsley, "Network Tomography on General Topology", in *Proc. of ACM SIGMETRICS'2002*, June 2002.

[6] M. Rabbat, R. Nowak and M. Coates, "Multiple Source, Multiple Destination Network Tomography", in *Proc. of IEEE INFOCOM'2004*, Mar., 2004.

[7] R. Black, A. Donnelly and C. Fournet, "Ethernet Topology Discovery without Network Assistance", in *Proc. of IEEE ICNP'2004*, Oct., 2004.

[8] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology Discovery in Heterogeneous IP Networks," in *Proc. of IEEE INFOCOM'2000*, Mar. 2000. Also appear in IEEE/ACM Transactions on Networking, Vol. 12, No 3, June 2004, pp. 401-414.

[9] B. Lowekamp, D.R. O'Hallaron, and T.R. Gross, "Topology Discovery for Large Ethernet Networks," in *Proc. of ACM SIGCOMM*, Aug. 2001.

[10] Y. Bejerano, Y. Breitbart, M. Garofalakis, and R. Rastogi, "Physical Topology Discovery for Large Multi-Subnet Networks," in *Proc. of IEEE INFOCOM'2000*, Mar. 2003.

[11] J. Gross and J. Yellon, "Graph Theory and Its Applications," *CRC Press*, 1999.

[12] R. Perlman "Interconnections: Bridges, Routers, Switches, and Internetworking Protocols," *Addison-Wesley*, 1999.

[13] W. R. Stevens, "TCP/IP Illustrated, Volume 1, The Protocols," *Addison-Wesley*, 1994.

[14] "802.1Q, IEEE Standards for Local and metropolitan area networks Virtual Bridged Local Area Networks," *IEEE*, 2003 Edition.

[15] Y. Bejerano, "Taking The Skeletons Out Of the Closets: A Simple And Efficient Topology Discovery Scheme for Large Multi-subnet LANs", *ITD-05-46100J*, Lucent Technologies, March 2005.