Physical Topology Discovery for Large Multi-Subnet Networks

Yigal Bejerano, Yuri Breitbart*, Minos Garofalakis, Rajeev Rastogi

Bell Labs, Lucent Technologies

600 Mountain Ave., Murray Hill, NJ 07974.

{bej,minos,rastogi}@research.bell-labs.com

Abstract— **Knowledge of the up-to-date physical (i.e., layer-2) topology of an Ethernet network is crucial to a number of critical network management tasks, including reactive and proactive resource management, event correlation, and root-cause analysis. Given the dynamic nature of today's IP networks, keeping track of topology information manually is a daunting (if not impossible) task. Thus, effective algorithms for automatically discovering physical network topology are necessary. In this paper, we propose the *first* complete algorithmic solution for discovering the physical topology of a large, heterogeneous Ethernet network comprising multiple subnets as well as (possibly) dumb or un-cooperative network elements. Our algorithms rely on standard SNMP MIB information that is widely supported in modern IP networks and require no modifications to the operating system software running on elements or hosts. Furthermore, we formally demonstrate that our solution is *complete* for the given MIB data; that is, if the MIB information is sufficient to uniquely identify the network topology then our algorithm is guaranteed to recover it. To the best of our knowledge, ours is the first solution to provide such a strong completeness guarantee.**

Index Terms— **Layer-2 Topology Discovery, Graph Theory, Ethernet LAN, Subnets, SNMP MIB, Switches, Hubs.**

I. INTRODUCTION

Physical network topology refers to the characterization of the physical connectivity relationships that exist among entities in a communication network. Discovering the physical layout and interconnections of network elements is a prerequisite to many critical network management tasks, including reactive and proactive resource management, server siting, event correlation, and root-cause analysis. For example, consider a fault monitoring and analysis application running on a central IP network management platform. Typically, a single fault in the network will cause a flood of alarm signals emanating from different interrelated network elements. Knowledge of element interconnections is essential to filter out secondary alarm signals and correlate primary alarms to pinpoint the original source of failure in the network [1], [2]. Furthermore, a full physical map of the network enables a proactive analysis of the impact of link and device failures.

Despite the critical role of physical topology information in enhancing the manageability of modern IP networks, obtaining such information is a very difficult task. The majority of commercial network-management tools feature an IP mapping functionality for automatically discovering routers and subnets and generating a *network layer* (i.e., ISO *layer-3*) topology showing the router-to-router interconnections and router

interface-to-subnet relationships. Building a layer-3 topology is relatively easy because routers must be explicitly aware of their neighbors in order to perform their basic function. Therefore, standard routing information is adequate to capture and represent layer-3 connectivity. Unfortunately, layer-3 topology covers only a small fraction of the interrelationships in an IP network, since it fails to capture the complex interconnections of *layer-2* network elements (switches, bridges, and hubs) that comprise each Ethernet LAN. Hardware providers, like Cisco and Intel, have designed their own proprietary protocols for discovering physical interconnections but these tools are of no use in a heterogeneous, multi-vendor environment. More recently, the IETF has acknowledged the importance of this problem by designating a "physical topology" *SNMP Management Information Base (MIB)* [3], but the proposal merely reserves a portion of the MIB space without defining any protocol or algorithm for obtaining the topology information. Clearly, as more switches, bridges, and hubs are deployed to provide more bandwidth through subnet microsegmentation, the portions of the network infrastructure that are transparent to current network-management tools will continue to grow. Under such conditions, it is obvious that the network manager's ability to troubleshoot end-to-end connectivity or assess the potential impact of link or device failures in switched networks will be severely impaired.

Developing effective algorithmic solutions for automatically discovering the up-to-date physical topology of a large, heterogeneous Ethernet network poses several difficult challenges. More specifically, there are three fundamental sources of complexity for physical topology discovery.

1) *Inherent Transparency of Layer-2 Hardware.* Layer-2 network elements (switches, bridges, and hubs) are completely transparent to endpoints and layer-3 hardware (routers) in the network. Switches themselves only communicate with their neighbors in the limited exchanges involved in the *spanning tree protocol* [4], and the only state maintained is in their *Address Forwarding Tables (AFTs)*, which are used to direct incoming packets to the appropriate output port. Fortunately, most switches/bridges (see (3) below) make this information available through a standard SNMP MIB [5], [6].

2) *Multi-Subnet Organization.* Modern switched networks usually comprise *multiple subnets* with elements in the same subnet communicating directly (i.e., without involving routers) whereas communication between elements in different subnets must traverse through the

routers for the respective subnets. Furthermore, elements of different subnets are often directly connected to each other. This obviously introduces serious problems for physical topology discovery, since it means that an element can be completely transparent to its direct physical neighbor(s).

3) *Transparency of Dumb or Uncooperative Elements.* Besides SNMP-enabled bridges and switches that are able to provide access to their AFTs, a switched network can also deploy "dumb" elements like *hubs* to interconnect switches with other switches or hosts [1]. Hubs do not participate in switching protocols and, thus, are essentially transparent to switches and bridges in the network. Similarly, the network may contain switches from which no address-forwarding information can be obtained either because they do not speak SNMP or because SNMP access to the switch is disabled. Clearly, inferring the physical interconnections of hubs and "uncooperative" switches based on the limited AFT information obtained from other elements poses a non-trivial algorithmic challenge.

Related Work. SNMP-based algorithms for automatically discovering network layer (i.e., layer-3) topology are featured in many common network management tools, such as HP's OpenView (www.openview.hp.com) and IBM's Tivoli (www.tivoli.com). Recognizing the importance of layer-2 topology, a number of vendors have recently developed proprietary tools and protocols for discovering physical network connectivity. Examples of such systems include Cisco's Discovery Protocol (www.cisco.com) and Bay Networks' Optivity Enterprise (www.baynetworks.com). Such tools, however, are typically based on vendor-specific extensions to SNMP MIBs and are not useful on a heterogeneous network comprising elements from multiple vendors. Peregrine's Infratools software (www.peregrine.com), Riversoft's NMOS product (www.riversoft.com), and Micromuse's Netcool/Precision application (www.micromuse.com) claim to support layer-2 topology discovery, but these tools are based on proprietary technology to which we do not have access.

In our recent work [8], we have proposed an algorithm that relies solely on standard AFT information collected in SNMP MIBs to discover the physical topology of heterogeneous networks comprising switches and bridges organized in multiple subnets. Unfortunately, our algorithm assumes that AFT information is available from *every* node in the underlying network and, thus, cannot cope with hubs or uncooperative switches. In a follow-up paper, Lowekamp et al. [7] suggest techniques for inferring network-element connectivity using incomplete AFT information and also discussed how to handle dumb/uncooperative elements; however, their algorithm is designed to work only in the much simpler case of *a single subnet* and can easily be shown to fail when multiple subnets are present. Thus, there is really no earlier work on physical topology discovery that addresses all three key research challenges outlined earlier in this section.

Our Contributions. In this paper, we propose a novel,

practical algorithmic solution for discovering the physical topology of large, heterogeneous IP networks comprising multiple subnets as well as (possibly) dumb or uncooperative elements; thus, our algorithm is essentially the *first* to address the the physical topology discovery problem in its full generality. Similar to our earlier work [8], the practicality of the solutions proposed in this paper stems from the fact that they rely solely on standard information routinely collected in the SNMP MIBs [5], [6] of elements and they require no modifications to the operating system software running on elements or hosts. Unlike [8], however, our algorithm is designed to infer connectivity information in the presence of hubs and/or switches not speaking SNMP; in fact, it can be shown that the algorithms proposed here completely subsume the solution proposed in our earlier paper.

Abstractly, our topology-discovery algorithm initially employs the AFT information supplied by SNMP-enabled elements to produce a partial, coarse view of the underlying network topology as a collection of *skeleton paths*. Our skeleton-path mechanism is a generalization of traditional paths that basically captures whatever partial knowledge we have accumulated on the actual network topology. Our algorithm then enters an iterative, *skeleton-path refinement* process during which *constraints* inferred from the overall skeleton-path collection are exploited to refine the topology information in individual skeleton paths.[2] Finally, once all skeleton paths have been resolved into complete arrangements of network elements, our algorithm stitches the paths together to infer the underlying network topology including the connections of "invisible" hubs and uncooperative switches.

It is well known that even complete AFT information from all network nodes is often insufficient to uniquely identify the underlying physical network topology; see, e.g., [8] for examples of different network topologies generating identical collections of AFTs. We are able, however, to demonstrate a strong *completeness* property for the solution proposed in this paper. More specifically, we formally prove that if the AFT information is sufficienct to uniquely identify the network topology then our algorithm is *guaranteed* to recover it. To the best of our knowledge, ours is the first SNMP-based topology-discovery algorithm to provide such a strong completeness guarantee. Due to space constraints, some theoretical results in this paper are presented without a complete proof; the details can be found in the full paper [9]. Our algorithm is currently under implementation for Lucent's NetInventory topology-discovery tool.

II. DEFINITIONS AND SYSTEM MODEL

In this section, we present necessary background information and the system model that we adopt for the physical topology discovery problem. We refer to the domain over which topology discovery is to be performed as a *switched domain*, which essentially comprises a maximal set S of switches such that there is a path between every pair of switches involving only switches in S. (Switches are essentially bridges with many ports, so the terms "switch" and "bridge" can be

---

[1] Even though properly-designed networks would not use hubs to interconnect multiple switches, this is a scenario that can easily arise in practice [7].

[2] To the best of our knowledge, existing *constraint-solving tools* cannot handle or solve the type of constraints considered here in order to identify the underlying network topology.

used interchangeably; we will primarily use "switch" in the remainder of this paper.) More specifically, we model the target switched domain as an *undirected tree* $G = (V, E)$, where each node in V represents a network element and each edge in E represents a physical connection between two element ports. The set V comprises both *labeled* and *unlabeled* nodes. Labeled nodes basically represent switches, routers, and hosts that have a unique identifying MAC address and can provide AFT information through SNMP queries to the appropriate parts of their MIB; unlabeled nodes, on the other hand, represent both "dumb" hub devices or switching elements with no SNMP support[3]. To simplify the discussion, we refer to labeled and unlabeled nodes simply as *switches* and *hubs* (respectively) in the remainder of the paper.

Note that the graph G essentially captures the (tree) topology of unique active forwarding paths for elements within a switched domain as determined by the *spanning tree protocol* [4]. Our topology discovery algorithm is based on using the MAC addresses learned through backward learning on ports that are part of the switched-domain spanning tree (and stored at the port AFTs of labeled network nodes). We use the notation (v, k) to identify the kth port of node $v \in V$, and $F_{v,k}$ to denote the set AFT entries at port (v, k) (i.e., the set of MAC addresses that have been seen as source addresses on frames received at (v, k)). (To simplify notation, we will often omit the parentheses and comma from our port-id notation when referring to a specific port of v, e.g., $v1$, $v2$, and so on.) Since G is a tree, we obviously have a unique path in G between every pair of nodes $s, t \in V$, and we use the symbol $P_{s,t}$ to identify the set of port-ids along the *path from s to t* (also referred to as the "$s - t$ path"). We also use the notation $v(u)$ to denote the port of node v that (the address of) node u is found off of (i.e., the port of v leading to u in G). Table I summarizes the key notation used throughout the paper with a brief description of its semantics. Additional notation will be introduced when necessary.

Symbol	Semantics
$G = (V, E)$	Switched-domain network graph (tree)
(v, k)	k^{th} port of node $v \in V$ ($v1$, $v2$, ...)
$F_{v,k}$	AFT entries at (i.e., nodes reachable from) (v, k)
$v(u)$	Port of node v leading to node u in G
$\mathcal{N}_v$	Subnets in G containing v in their spanning subtree
$P_{s,t}$	Set of switch ports along the path from s to t in G
$Q_{s,t}$	Skeleton path from s to t in G
$I_{x,z}^{s,t}$	Set of ports at the intersection of $P_{s,t}$ and $P_{x,z}$
$Q_{x,z}^{s,t}$	Projection of path $Q_{x,z}$ onto path $Q_{s,t}$

TABLE I

NOTATION.

Every labeled node in our switched domain 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 will traverse exactly along the set of

[3]Note that end-hosts and routers in the network are represented as leaf nodes in G, and are practically indistinguishable for the purposes of layer-2 topology discovery.

ports $P_{s,t}$ in G. Typically, every network element within a switched domain is identified with a single IP address and a subnet mask that defines the IP address space corresponding to the element's subnet. For example, IP address 135.104.46.1 along with mask 255.255.255.0 identifies a subnet of network elements with IP addresses of the form 135.104.46.x, where x is any integer between 1 and 254. Let $\mathcal{N}$ be the collection of subnets of the graph G. Every subnet $N \in \mathcal{N}$ defines a *connecting subtree in G*; that is, a tree subgraph of G that is essentially spanned by the nodes in subnet N, and contains all nodes and edges of G that lie on paths between any pair of nodes in N. Let $\mathcal{N}_v \subseteq \mathcal{N}$ denote the collection of subnets containing node $v \in V$ in their connecting subtrees; clearly, the AFTs at the ports of node v contain node-reachability information only for the subnets in $\mathcal{N}_v$. We say that the AFT $F_{v,k}$ of v is *complete* if, for all $N \in \mathcal{N}_v$, $F_{v,k}$ contains the MAC addresses of all nodes in N that are reachable by port (v, k).

Similar to [8], our physical topology discovery algorithms rely on the assumption that the AFT information obtained from labeled nodes in the network is complete. This completeness requirement can be enforced using techniques similar to those in [8] (e.g., using "spoofed" ICMP-echo packets to force switch communication). A second possibility (also proposed in [8]) is to relax this completeness requirement and allow our schemes to make "approximate" decisions while working with only partial AFT information.

III. OVERVIEW OF OUR TOPOLOGY DISCOVERY ALGORITHM

The goal of our proposed algorithm is to discover the physical topology of the underlying multi-subnet network represented by the switched domain graph $G = (V, E)$ as accurately as possible using only the AFT information provided by labeled nodes in G. Thus, our topology-discovery algorithm uses the AFT information provided to (1) discover the direct physical connections between labeled element (i.e., switch) ports, and (2) infer the existence of unlabeled nodes (i.e., hubs) in G as well as the set of switch ports that are connected to each hub. A key concept in our topology discovery algorithm is the concept of *skeleton paths* defined formally below.

Definition 3.1: A *skeleton path* from node s to node t in G is defined as a sequence $Q_{s,t} = < U_1, U_2, \ldots, U_K >$ of non-empty port-id sets $U_1, \ldots, U_K$ forming a partition of $P_{s,t}$ ($U_i \cap U_j = \phi$, $\cup_i U_i = P_{s,t}$) such that: (1) Each U_j contains the port-ids of a *contiguous* segment of the $s - t$ path; and, (2) For each $i < j$, all the port-ids in U_i precede those in U_j on the $s - t$ path. ∎

Intuitively, an $s - t$ skeleton path describes some *partial* knowledge (i.e., port ordering information) that we have about the actual $s - t$ path in the network graph G. This partial knowledge basically describes subsets of ports U_j that we know to be *contiguous* in the path from s to t in G, as well as the ordering of these subsets as we traverse G from s to t. Thus, the "coarsest" $s - t$ skeleton path comprises a single large subset U_i between nodes s and t with essentially no port-ordering information, whereas in the "finest" $s - t$ skeleton path each U_i is a singleton (a single port-id) and the complete ordering of the ports on the $s - t$ path is specified.
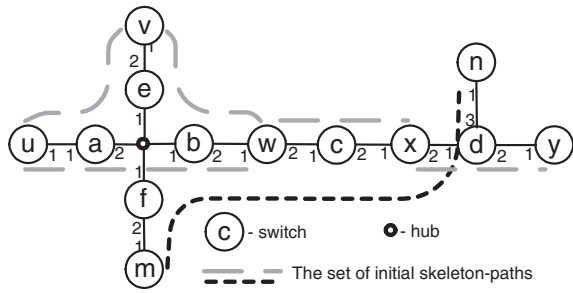
Fig. 1.   Example network graph and its decomposition in skeleton paths.

Note that determining the set of switch port-ids to be included in an $s-t$ skeleton path using AFT information is fairly straightforward when s and t belong to the same subnet. The key observation is that a node v is on the path from s to t in G if and only if there are two distinct ports $v(s)$ and $v(t)$ of v such that v "sees" node s (t) at port $v(s)$ (resp., $v(t)$) (i.e., $s \in F_{v,v(s)}$ and $t \in F_{v,v(t)}$). Also note that, since our skeleton-path definition assumes that the path is *oriented* from s to t, port $v(s)$ will always precede port $v(t)$ on the $s-t$ path; thus, we will always denote $v(s)$ before $v(t)$ in the skeleton path $Q_{s,t}$ (even when these ports are in the same U_i subset). Obviously, this simple port-ordering rule for each node is easily obtained from the AFT information at v.

Example 3.1: Consider the network depicted in Figure 1, where the numbers near the links represent the port-ids. Nodes u, v, w, x, y are in one subnet, nodes m, n in another subnet, and every one of the nodes a, b, c, d, e, f defines a separate subnet (with only one node). One possible skeleton path from node u to node x is: $Q_{u,x} = < \{u1\}, \{a1, a2, b1, b2\}, \{w1\}, \{w2\}, \{c1\}, \{c2\}, \{x1\} >$. Clearly, this skeleton path only provides partial information on the the topology of the true $u - x$ path in G. More specifically, $Q_{u,x}$ specifies that the ports $x1$ and $c2$ are directly connected or they are connected to the same hub. Similarly, $Q_{u,x}$ also indicates that port w_1 is connected (either directly or through a hub) to one of $a2$ or $b2$. (Note that, since port $a1$ ($b1$) precedes $a2$ (resp., $b2$) on the $u - x$ path and $w1$ succeeds nodes a and b in $Q_{u,x}$, $w1$ can only be connected to $a2$ or $b2$.) ∎

At a high level, our proposed topology-discovery algorithm (depicted in Figure 2) represents the underlying network by a collection of skeleton paths, $\mathcal{Q}$, between pairs of nodes belonging to the same subnet, and proceeds by iteratively *refining* $\mathcal{Q}$ to provide more accurate topology information for G. The initial input to our algorithm is the collection of subnets $\mathcal{N}$ in the network G as well as the AFT information from all labeled nodes (switches) in G. As a first step, our algorithm computes an initial collection of skeleton paths $\mathcal{Q}$ that, essentially, captures the given AFT information by identifying the set of port-ids between selected pairs of nodes that "cover" all paths in G (procedure INITSKELETONPATHS). Our algorithm then enters an iterative *skeleton-path refinement* process, that tries to determine a complete port order for each skeleton path in $\mathcal{Q}$. The key idea here is to use the aggregate information in $\mathcal{Q}$ to further divide the internal U_i subsets of each skeleton-path $Q_{s,t} \in \mathcal{Q}$ into smaller subsets, until either a complete order is obtained or no further refinement is possible. This path-refinement task for skeleton path $Q_{s,t}$

is accomplished with the help of two key procedures. First, procedure COMPUTECONSTRAINTS exploits the information in $\mathcal{Q}$ (more specifically, the *intersections* of $Q_{s,t}$ with other skeleton paths in $\mathcal{Q}$) to obtain a collection $\mathcal{S}$ of additional constraints (termed *path constraints*) on the port order in $Q_{s,t}$. Second, procedure REFINEPATH uses the discovered set of path constraints $\mathcal{S}$ to further refine $Q_{s,t}$. When no further skeleton-path refinements are possible, our algorithm invokes a FINDCONNECTIONS procedure that uses the refined paths to output the switch and hub connections discovered in G.

---

procedure TOPOLOGYDISCOVERY($\mathcal{N}$, AFTs)
1. $\mathcal{Q} = $ INITSKELETONPATHS($\mathcal{N}, AFT$)
2.   **do**
3.   *done* = **true**
4.   **for each** $Q_{s,t} \in \mathcal{Q}$ **do**
5. $P_{s,t} = \bigcup_{U_k \in Q_{s,t}} U_k$
6. $\mathcal{S} = $COMPUTECONSTRAINTS($Q_{s,t}$, $\mathcal{Q}$)
7. $Q_{s,t}^{new} = $ REFINEPATH($P_{s,t}$, $\mathcal{S}$)
8.   **if** ($Q_{s,t}^{new} \neq Q_{s,t}$) **then**
9.   replace $Q_{s,t}$ by $Q_{s,t}^{new}$ in $\mathcal{Q}$
10.   *done* = **false**
11.   **endif**
12.   **endfor**
13.   **while** (**not** *done*)
14.   FINDCONNECTIONS($\mathcal{Q}$)

Fig. 2.   Our Topology Discovery Algorithm.

One of the main challenges in our work lies in determining the most complete set of path constraints for each skeleton path $Q_{s,t}$, so that we maximize the amount of port-ordering knowledge incorporated in $Q_{s,t}$ during future iterative-refinement steps. As we show later in this paper, such path constraints can result from rather complicated intersection patterns of several skeleton paths in $\mathcal{Q}$. Thus, it is very hard to directly obtain the "full" set of path constraints that would allow us to refine an initial $Q_{s,t}$ skeleton path into a complete port order in a single step. However, even partial-order information obtained through a subset of the constraints imposed on $Q_{s,t}$ can be used to further refine other skeleton paths in $\mathcal{Q}$ during future iterations. Thus, our topology-discovery algorithm may require several iterative-refinement steps, during which skeleton paths in $\mathcal{Q}$ are further refined from iteration to iteration, until the algorithm eventually converges to the maximal possible port-ordering information for each path in G for the given set of inputs. We discuss the key algorithmic components of our topology-discovery algorithm in detail in the sections that follow.

IV. THE INITIAL SKELETON-PATH COLLECTION

The first task faced by our algorithm is to "translate" the input AFT and subnet information into an initial collection of skeleton paths. The key observation here (already discussed in Section III) is that, for nodes s and t belonging to the *same subnet*, we can easily use the AFT information to determine the set of switch ports $P_{s,t}$ on the $s-t$ path in G: if, for a node $v \neq s, t$, there exist two *distinct* ports $v(s) \neq v(t)$ such that $s \in F_{v,v(s)}$ and $t \in F_{v,v(t)}$ then $v(s), v(t) \in P_{s,t}$; otherwise, v cannot be on the $s-t$ path in G. (Of course, the source and destination ports on nodes s and t can also be simply determined from their AFT information.)

Thus, a simple solution to the initial skeleton-path construction problem is to build a skeleton path $Q_{s,t}$ for each pair s, t of distinct nodes belonging to the same subnet, for each of the underlying subnets. The problem with such a simplistic approach is that it results in very significant overlap between the resulting paths in $\mathcal{Q}$; this, in turn, implies that our algorithm may need to compute the port order for the same path segment several times, resulting in significant computation-time overheads. Instead, our solution relies on constructing a *concise* collection of skeleton paths for each subnet N such that paths between nodes of N in $\mathcal{Q}$: (a) are not contained in other paths between N's nodes, and (b) cannot be broken into smaller paths between N's nodes. Intuitively, the resulting skeleton paths for subnet N "minimally" cover all nodes of N using the smallest possible segments between such nodes. Our INITSKELETONPATHS procedure (depicted in Figure 3) builds this concise collection by simply considering, for each subnet N, all possible $s - t$ paths with $s, t \in N$ and adding an initial $Q_{s,t}$ skeleton path to $\mathcal{Q}$ only if the collection of intermediate nodes on the $s - t$ path (denoted by X in Figure 3) does not contain another N node. As an example, Figure 1 depicts the six initial skeleton paths in $\mathcal{Q}$ for the network in Example 3.1.

```
procedure INITSKELETONPATHS(𝒩, AFTs)
1.  𝒬 = ∅
2.  for each N ∈ 𝒩 do
3.     for each {s, t} ∈ N do
4.        X = {v|v ∈ V − {s, t} ∧ ∃v(s) ≠ v(t),
              such that s ∈ F_{v,v(s)} ∧ t ∈ F_{v,v(t)}}
5.        if (X ∩ N = φ) then
6.           Q_{s,t} =< {s(t)}, {v(s), v(t)|v ∈ X}, {t(s)} >
7.           𝒬 = 𝒬 ∪ {Q_{s,t}}.
8.        endif
9.     endfor
10. endfor
11. return(𝒬)
```

Fig. 3.   The INITSKELETONPATHS Procedure.

V.   COMPUTING SKELETON-PATH CONSTRAINTS

In this section, we address the problem of discovering a collection of constraints that will allow our algorithm to refine the port order for a given skeleton path $Q_{s,t} \in \mathcal{Q}$. Abstractly, these constraints follow (either explicitly or implicitly) from the *intersections* of $Q_{s,t}$ with other skeleton paths in the $\mathcal{Q}$ collection. We begin by presenting some useful definitions and notational conventions.

A. Skeleton-Path Constraints: Definitions and Notation

We say that a skeleton path $Q_{x,z} \in \mathcal{Q}$ *intersects* $Q_{s,t}$ if $P_{s,t} \bigcap P_{x,z} \neq \phi$. Our skeleton-path collection $\mathcal{Q}$ can be partitioned into two subsets $\mathcal{Q} = \mathcal{Q}^I_{s,t} \cup \mathcal{Q}^{NI}_{s,t}$, where $\mathcal{Q}^I_{s,t}$ ($\mathcal{Q}^{NI}_{s,t}$) contains all the paths in $\mathcal{Q}$ that intersect (resp., do not intersect) path $Q_{s,t}$. (Note that, trivially, $Q_{s,t} \in \mathcal{Q}^I_{s,t}$.) For any skeleton path $Q_{x,z} \in \mathcal{Q}^I_{s,t}$, let $I^{s,t}_{x,z}$ denote the collection of port-ids in the intersection of the $s - t$ and $x - z$ skeleton paths, i.e., $I^{s,t}_{x,z} = P_{s,t} \bigcap P_{x,z}$. To simplify the exposition, we assume that all paths $Q_{x,z} \in \mathcal{Q}^I_{s,t}$ have the *same orientation* as $Q_{s,t}$; that is, any port in their intersection $I^{s,t}_{x,z}$ faces either

s and x, or t and z (the starting and ending points of the paths are on the same "side" of the network graph). Of course, either $Q_{x,z}$ or $Q_{z,x}$ must have the same orientation as $Q_{s,t}$, and this can be easily resolved from the AFTs of ports in $I^{s,t}_{x,z}$. Constraints on the port order in $Q_{s,t}$ can result from the *projection* of another path $Q_{x,z} \in \mathcal{Q}^I_{s,t}$ onto $Q_{s,t}$, which is formally defined below.

Definition 5.1: The *projection* of $Q_{x,z} \in \mathcal{Q}^I_{s,t}$ onto $Q_{s,t}$, denoted by $Q^{s,t}_{x,z}$, is the skeleton path that results by taking the intersection of every subset $U_k \in Q_{x,z}$ with the set $P_{s,t}$ and omitting empty sets; that is, $Q^{s,t}_{x,z} =< U_1 \cap P_{s,t}, \ldots, U_K \cap P_{s,t}|U_i \in Q_{x,z}$ and $U_i \cap P_{s,t} \neq \phi >$. ∎

Clearly, any path projection onto $Q_{s,t}$ is essentially a valid skeleton representation for a segment of the true $s - t$ path in G and, as such, can enforce additional constraints on the port order in $Q_{s,t}$. Such constraints can be broadly classified into two types: (1) *Contiguity constraints* forcing a given subset $S \subset P_{s,t}$ of port-ids to define a contiguous segment of the $s - t$ path (e.g., any $S = U_i \cap P_{s,t} \neq \phi$ in Definition 5.1); and, (2) *Order constraints* forcing all port-ids in a subset $S^1 \subset P_{s,t}$ to precede those of another subset $S^2 \subset P_{s,t}$ (e.g., $S^1 = U_i \cap P_{s,t}$ and $S^2 = U_{i+1} \cap P_{s,t}$ in Definition 5.1). We give a generic definition of path constraints that captures both contiguity and order constraints.

Definition 5.2: A *path constraint* $S_i =< S^1_i, S^2_i >$ for skeleton path $Q_{s,t}$ is an ordered pair of two disjoint subsets of port-ids $S^1_i, S^2_i \subseteq P_{s,t}$ such that: (1) S^1_i, S^2_i, and $S^1_i \cup S^2_i$ define contiguous segments of ports on the $s - t$ path, and (2) the ports in S^1_i precede those in S^2_i in the path from s to t in G. ∎

Note that a simple contiguity constraint S can be simply represented as $< S, \phi >$.

B. Computing Skeleton Path Constraints

We now turn to our algorithm for computing a collection of path constraints $\mathcal{S}$ on the skeleton path $Q_{s,t}$ using other paths in $\mathcal{Q}$ (i.e., procedure COMPUTECONSTRAINTS in Figure 2). We first consider the discovery of *explicit* path constraints, i.e., constraints that can be inferred directly from the AFT information and the projections of other skeleton paths in $\mathcal{Q}^I_{s,t}$ onto $Q_{s,t}$. We then discuss the more subtle case of *implicit* path constraints.

Explicit Path Constraints. Consider any switch v on the $Q_{s,t}$ skeleton path. Using the AFT information from v we can readily define the path constraint $< \{v(s)\}, \{v(t)\} >$, basically stating that the two ports of v on the path from s to t must be contiguous and the port facing s must precede that facing t. We add these constraints to $\mathcal{S}$ for all nodes $v \neq s, t$ on the $Q_{s,t}$ path.

Further, consider any (intersecting) skeleton path $Q_{x,z} \in \mathcal{Q}^I_{s,t}$ and its projection $Q^{s,t}_{x,z}$ onto $Q_{s,t}$. As mentioned earlier, such a projection defines a valid skeleton representation for a segment of the true $s - t$ path in G and, thus, defines additional contiguity and order constraints on $Q_{s,t}$. More specifically, for all projections $Q^{s,t}_{x,z} = \{U_1, U_2, \cdots, U_K\}$, we augment $\mathcal{S}$ by adding the path constraints $< U_i, U_{i+1} >$ for all $i = 1, \ldots, K$ (where, we assume $U_{K+1} = \phi$ to cover the case $K = 1$).

Implicit Path Constraints. Abstractly, implicit path constraints on $Q_{s,t}$ are obtained through the intersection of two

or more paths with $Q_{s,t}$ as well as other parts of the network graph G. More specifically, consider the subgraph of G that is obtained by removing all ports in $P_{s,t}$ from our network. Since G is a tree, it is easy to see that this subgraph is essentially a collection of subtrees $\mathcal{T}_{s,t}$ of G such that each $T \in \mathcal{T}_{s,t}$ is attached to a *single connection point* (i.e., switch or hub) on the $Q_{s,t}$ skeleton path. Implicit path constraints result from the intersection of paths in $\mathcal{Q}_{s,t}^I$ with a given subtree $T \in \mathcal{T}_{s,t}$ taking advantage of the above "single-connection-point" observation.

Of course, a problem here is that our algorithm needs to employ some knowledge about the set of port-ids within different subtrees in $\mathcal{T}_{s,t}$ without knowing their exact topology. Our algorithm collects this knowledge using a port-aggregation technique that partitions the ports not included in $P_{s,t}$ into a collection $\mathcal{B}$ of maximal, disjoint *"bins"*, such that the ports in each bin $B \in \mathcal{B}$ are guaranteed to be included in a *single* subtree of $T \in \mathcal{T}_{s,t}$ of G. Note that this is only a sufficient condition, so that port-ids belonging to the same subtree in $\mathcal{T}_{s,t}$ can in fact end up in different bins of $\mathcal{B}$ in our algorithm. Nevertheless, this condition still (conservatively) guarantees that paths in $\mathcal{Q}_{s,t}^I$ intersecting with the same bin $B \in \mathcal{B}$ share a single connection point on $Q_{s,t}$ and, therefore, can enforce implicit path constraints on $Q_{s,t}$. Our technique for aggregating ports into bins relies on the following property, which follows directly from the fact that our network graph G is a tree.

Property 5.1: Any pair of paths $Q_{x,z}$, $Q_{u,v}$ not intersecting $Q_{s,t}$ (i.e., $Q_{x,z}, Q_{u,v} \in \mathcal{Q}_{s,t}^{NI}$) with $P_{x,z} \cap P_{u,v} \neq \phi$ belong to the same subtree $T \in \mathcal{T}_{s,t}$. ∎

Thus, all ports on any two intersecting paths in $\mathcal{Q}_{s,t}^{NI}$ can be safely placed in the same bin in $\mathcal{B}$. Our algorithm works by initially defining: (1) for every node $v \notin P_{s,t}$, a bin B_v containing all of v's ports, i.e., $B_v = \{(v,k)|(v,k) \text{ is a port of } v\}$; and, (2) for every path $Q_{u,v} \in \mathcal{Q}_{s,t}^{NI}$, a bin $B_{u,v} = P_{u,v}$. The algorithm then forms the final collection of bins $\mathcal{B}$ by iteratively merging any two bins whose intersection is non-empty until all bins are disjoint (based on Property 5.1).

Given that the port bins $\mathcal{B}$ computed above are guaranteed to connect to a single point of the $Q_{s,t}$ skeleton path, we can use them in a manner equivalent to subtrees in $\mathcal{T}_{s,t}$ for computing implicit path constraints on $Q_{s,t}$. Consider two (intersecting) paths $Q_{x,z}, Q_{u,v} \in \mathcal{Q}_{s,t}^I$ that also intersect with a single bin $B \in \mathcal{B}$, and let $I_{x,z}^{s,t}$ and $I_{u,v}^{s,t}$ denote their respective intersections with $P_{s,t}$. Since B has a single connection point to $Q_{s,t}$, the segments of $P_{s,t}$ defined by $I_{x,z}^{s,t}$ and $I_{u,v}^{s,t}$ have a common end-point (switch or hub) on the $Q_{s,t}$ path. If $I_{x,z}^{s,t}$ and $I_{u,v}^{s,t}$ are disjoint then they are on opposite sides of the common connection point (Figure 4(a)), so their union $I_{x,z}^{s,t} \cup I_{u,v}^{s,t}$, defines a contiguity constraint on $Q_{s,t}$. If, on the other hand, $I_{x,z}^{s,t}$ and $I_{u,v}^{s,t}$ intersect, then they are on the same side of their common end-point (Figure 4(b)), and one of them contains the other. Suppose that $I_{x,z}^{s,t} \supset I_{u,v}^{s,t}$; then, clearly, $I_{x,z}^{s,t} - I_{u,v}^{s,t}$ also defines a contiguity constraint on $Q_{s,t}$. In general, given $Q_{x,z}, Q_{u,v} \in \mathcal{Q}_{s,t}^I$ intersecting with a single port bin $B \in \mathcal{B}$, all the implicit contiguity constraints added to our path constraint set $\mathcal{S}$ are: $I_{x,z}^{s,t} \cup I_{u,v}^{s,t}$, $I_{x,z}^{s,t} \cap I_{u,v}^{s,t}$, $I_{x,z}^{s,t} - I_{u,v}^{s,t}$, and $I_{u,v}^{s,t} - I_{x,z}^{s,t}$ (where, of course, empty sets are ignored).

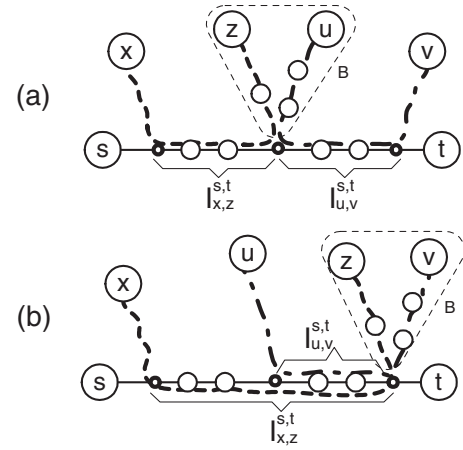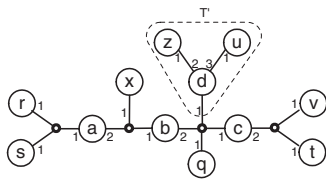The computed port bins and the single connection point

Fig. 4. Computing implicit path constraints using bin B.

property can also be exploited to infer *order constraints* on the $Q_{s,t}$ skeleton path. Consider two paths $Q_{x,z}, Q_{u,v} \in \mathcal{Q}_{s,t}^I$ intersecting with bin $B \in \mathcal{B}$, and assume that $I_{x,z}^{s,t}$ and $I_{u,v}^{s,t}$ are disjoint (Figure 4(a)) (the case of intersecting $I_{x,z}^{s,t}, I_{u,v}^{s,t}$ can be handled similarly). The key to determining the order of $I_{x,z}^{s,t}$ and $I_{u,v}^{s,t}$ on the $s-t$ path lies in discovering if one of the two path segments precedes or succeeds the connection point of the B bin. To describe the two scenarios succinctly, we define the functions FIRST(Q, S) and LAST(Q, S) that receive as input a skeleton path Q and a set of ports S, and return the index j of the first and last (respectively) subset $U_j \in Q$ that intersects S. It is easy to see that if FIRST$(Q_{x,z}, I_{x,z}^{s,t}) <$ LAST$(Q_{x,z}, B)$, then (since the $s-t$ and $x-z$ paths have the same orientation) the segment $I_{x,z}^{s,t}$ precedes the connecting point of bin B and we can conclude the path constraint $< I_{x,z}^{s,t}, I_{u,v}^{s,t} >$. Otherwise, if LAST$(Q_{x,z}, I_{x,z}^{s,t}) >$ FIRST$(Q_{x,z}, B)$, then the segment $I_{x,z}^{s,t}$ succeeds the connecting point of B, giving the path constraint $< I_{u,v}^{s,t}, I_{x,z}^{s,t} >$. (Note that *at most one* of the above conditions can hold since, by definition, $B \cap I_{x,z}^{s,t} = \phi$.) If both conditions are false, then we also check the corresponding FIRST/LAST conditions for $Q_{u,v}$ to see if they can determine an ordering for the two path segments.

Example 5.1: Consider the network depicted in Figure 5(a), where hosts (i.e., leaf nodes) comprise four different subnets, $\{u,v\}, \{s,t\}, \{x,z\}$, and $\{r,q\}$, and each switch (i.e., internal node) comprises a single-element subnet. The complete element AFTs are given in Figure 5(b) and the initial collection of skeleton paths, $\mathcal{Q}$, is shown in Figure 5(c). Consider the path constraints imposed by $\mathcal{Q}$ on the $Q_{u,v}$ path. From the AFT information, we directly conclude the constraints $< \{d_3\}, \{d_1\} >$ and $< \{c_1\}, \{c_2\} >$. Also, $P_{u,v}$ intersects both $P_{s,t}$ and $P_{x,z}$ with $I_{x,z}^{u,v} = \{d_1\}$ and $I_{s,t}^{u,v} = \{c_1, c_2\}$. Further, since both $P_{s,t}$ and $P_{x,z}$ intersect with the bin $B_{r,q} = \{r_1, a_1, a_2, b_1, b_2, q_1\}$ (resulting from $P_{r,q} \in \mathcal{Q}_{u,v}^{NI}$), we have the implicit contiguity constraint $< \{d_3, c_1, c_2\}, \phi >$. It is easy to see that the only $u-v$ path arrangement satisfying the above constraints is $Q_{u,v} = < \{u_1\}, \{d_3\}, \{d_1\}, \{c_1\}, \{c_2\}, \{v_1\} >$.
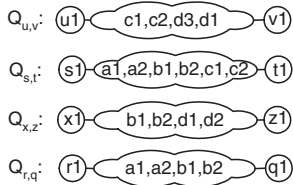
Now, consider path $Q_{s,t}$. Through the intersection of $P_{s,t}$ and $P_{r,q}$, we conclude the (explicit) contiguity constraint $< \{a_1, a_2, b_1, b_2\}, \phi >$. Also, through the intersection of $P_{u,v}$ and $P_{x,z}$ with both $P_{s,t}$ and the bin $B_d = \{d_1, d_2, d_3\}$, we infer the (implicit) contiguity constraint $< \{b_1, b_2, c_1, c_2\}, \phi >$.

$F_{a1}=\{r,s\}$	$F_{a2}=\{q,t\}$
$F_{b1}=\{s,x\}$	$F_{b2}=\{t,z\}$
$F_{c1}=\{s,u\}$	$F_{c2}=\{t,v\}$
$F_{d1}=\{x,v\}$	$F_{d2}=\{z\}$
	$F_{d3}=\{u\}$
$F_{s1}=\{t\}$	$F_{t1}=\{s\}$
$F_{x1}=\{z\}$	$F_{z1}=\{x\}$
$F_{u1}=\{v\}$	$F_{v1}=\{u\}$
$F_{r1}=\{q\}$	$F_{q1}=\{r\}$

(a) The considered network.

(b) The complete AFTs.

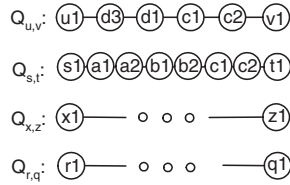(c) The initial skeleton-paths.

(d) The refined skeleton-paths $Q_{u,v}$ and $Q_{s,t}$.

Fig. 5. An example of implicit contiguity and order constraints.

These two constraints are not sufficient to define the port order on $P_{s,t}$ since both $s-a-b-c-t$ and $s-c-b-a-t$ satisfy them. However, with the knowledge of the complete $u-v$ path (above) we can infer an additional implicit order constraint; specifically, since $I_{u,v}^{s,t}$ and $I_{x,z}^{s,t}$ are disjoint and $\text{LAST}(Q_{u,v}, I_{u,v}^{s,t}) = 5 > \text{FIRST}(Q_{u,v}, B_d) = 2$, the connection point of B_d must precede the c node on the $s-t$ path. This implies the constraint $< I_{x,z}^{s,t}, I_{u,v}^{s,t} > = < \{b_1, b_2\}, \{c_1, c_2\} >$ which, in turn, uniquely identifies the underlying $s-t$ path as $s-a-b-c-t$. ∎

The detailed pseudo-code for our COMPUTECONSTRAINTS procedure is depicted in Figure 6. As is also clear from the discussion in Example 5.1, it may be impossible to use our path constraints to infer the complete path topology for a given skeleton path in $\mathcal{Q}$ unless some other path(s) in $\mathcal{Q}$ have been appropriately refined (e.g., consider $Q_{s,t}$ and $Q_{u,v}$ in our example). A key problem here stems from our partial knowledge of the ports that lie in the "single-connection-point" bins used to infer implicit constraints. Thus, our solution (Figure 2) needs to employ *iterative-refinement passes* over all skeleton-paths in $\mathcal{Q}$ until no further path refinements are possible.

VI. THE SKELETON-PATH REFINEMENT ALGORITHM

Once our topology-discovery algorithm has computed the set of path constraints $\mathcal{S}$ imposed on the $Q_{s,t}$ skeleton path, it invokes the REFINEPATH procedure (Step 7 in Figure 2) to "refine" the ordering of the port-ids in the $P_{s,t}$ set using the newly-discovered constraints. Our REFINEPATH algorithm (described in detail in this section) is a recursive procedure that receives as input the collection of port-ids P along the network path being considered, as well as a collection of path constraints $\mathcal{S}$ on the arrangement of those ports. Its output is a skeleton path $Q =< U_1, U_2, \ldots, U_K >$ over the ports in P that satisfies all the constraints in $\mathcal{S}$. Furthermore, as we demonstrate analytically later in this section, if the constraint collection $\mathcal{S}$ uniquely defines the port order for P then every subset U_i in the output path Q comprises a single port in

```
procedure COMPUTECONSTRAINTS(Q_{s,t}, Q)
1.  Q_{s,t}^I = {Q_{x,z}|Q_{x,z} ∈ Q and P_{x,z} ∩ P_{s,t} ≠ φ}
2.  Q_{s,t}^{NI} = Q - Q_{s,t}^I
3.  S = {< {v(s)}, {v(t)} > |∀v ≠ s, t on the s − t path}
4.  for each Q_{x,z} ∈ Q_{s,t}^I do    // Note that Q_{s,t} ∈ Q_{s,t}^I
5.      Compute the projection Q_{x,z}^{s,t} = {U_1, · · · , U_K}
6.      S = S ∪ {< U_1, φ >}
7.      for j = 2 to K do
8.          S = S ∪ {< U_{j−1}, U_j >}
9.      endfor
10. // Compute the port-bin collection B
11. B = {{P_{u,v}|Q_{u,v} ∈ Q_{s,t}^{NI}} ∪
                {{ all ports (v, k)}|v ∈ V − P_{s,t}}
12. while there are B_1, B_2 ∈ B s.t. (B_1 ∩ B_2 ≠ φ) do
13.     B = B − {B_1, B_2} ∪ {{B_1 ∪ B_2}}
14. // Discover implicit path constraints on Q_{s,t}
15. for each Q_{x,z}, Q_{u,v} ∈ Q_{s,t}^I and B ∈ B s.t. [ (I_{x,y}^{s,t} ≠ I_{u,v}^{s,t})
            and (P_{x,y} ∩ B ≠ φ) and (P_{u,v} ∩ B ≠ φ) ] do
16.     S = S ∪ {< I_{x,z}^{s,t} ∪ I_{u,v}^{s,t}, φ >, < I_{x,z}^{s,t} ∩ I_{u,v}^{s,t}, φ >,
                    < I_{x,z}^{s,t} − I_{u,v}^{s,t}, φ >, < I_{u,v}^{s,t} − I_{x,z}^{s,t}, φ >}
17.     if ( I_{x,z}^{s,t} ∩ I_{u,v}^{s,t} = φ ) then
18.         if [ FIRST(Q_{x,z}, I_{x,z}^{s,t}) < LAST(Q_{x,z}, B) or
                LAST(Q_{u,v}, I_{u,v}^{s,t}) > FIRST(Q_{u,v}, B) ] then
19.             S = S ∪ {< I_{x,z}^{s,t}, I_{u,v}^{s,t} >}
20.         else if [ FIRST(Q_{u,v}, I_{u,v}^{s,t}) < LAST(Q_{u,v}, B) or
                LAST(Q_{x,z}, I_{x,z}^{s,t}) > FIRST(Q_{x,z}, B) ] then
21.             S = S ∪ {< I_{u,v}^{s,t}, I_{x,z}^{s,t} >}
22.         endif
23.     else // i.e., I_{x,y}^{s,t} ∩ I_{u,v}^{s,t} ≠ φ, assume I_{x,y}^{s,t} ⊃ I_{u,v}^{s,t}
24.         if [ FIRST(Q_{x,z}, I_{x,z}^{s,t}) < LAST(Q_{x,z}, B) or
                FIRST(Q_{u,v}, I_{u,v}^{s,t}) < LAST(Q_{u,v}, B) ] then
25.             S = S ∪ {< (I_{x,z}^{s,t} − I_{u,v}^{s,t}), I_{u,v}^{s,t} >}
26.         else if [ LAST(Q_{x,z}, I_{x,z}^{s,t}) > FIRST(Q_{x,z}, B) or
                LAST(Q_{u,v}, I_{u,v}^{s,t}) > FIRST(Q_{u,v}, B) ] then
27.             S = S ∪ {< I_{u,v}^{s,t}, (I_{x,z}^{s,t} − I_{u,v}^{s,t}) >}
28.         endif
29.     endif
30. endfor
31. return S
```

Fig. 6. The COMPUTECONSTRAINTS Procedure.

P (i.e., Q defines the *complete* port order for the considered network path).

Abstractly, our REFINEPATH algorithm consists of three key steps: (1) Mapping the path-constraint collection $\mathcal{S}$ to a collection of contiguity constraints $\mathcal{R}$; (2) Using $\mathcal{R}$ and $\mathcal{S}$ to construct an auxiliary skeleton path L; and, (3) Recursing the refinement process on each subset of the auxiliary skeleton path L to obtain the output skeleton path Q. Intuitively, the set of contiguity constraints $\mathcal{R}$ enable us to identify segments of port-ids on the target path that are "connected" through the given set of constraints; these are basically the only (sub)paths for which we stand a chance to recover a complete port order (using the given constraints). The subsets in the auxiliary skeleton path L are then constructed using the derived contiguity constraints $\mathcal{R}$: the goal here is to ensure that our refinement algorithm can safely recurse within each individual subset of L while only considering the constraints "local" to this subset. Further, the path constraints in $\mathcal{S}$ are used to determine the order of subsets in L. Finally, we recurse on each subset of L and concatenate the skeleton (sub)paths returned to obtain the final skeleton path Q.

In the remainder of this section, we first describe the con-

struction of the contiguity constraint set $\mathcal{R}$ and the auxiliary skeleton path L. Then, we discuss our overall REFINEPATH algorithm in detail.

The Contiguity Constraint Set $\mathcal{R}$ and Connected Port Groups. The set of contiguity constraints $\mathcal{R}$ essentially contains all the contiguity constraints that can be directly inferred from the input set of path constraints $\mathcal{S}$. (To simplify the exposition, we will treat $\mathcal{R}$ as a set of port-id sets, i.e., each $R \in \mathcal{R}$ is a set of ports.) To ensure that $\mathcal{R}$ covers all ports in P we add singleton constraints for each port in P; we also exclude from $\mathcal{R}$ the "trivial" contiguity constraints P and ϕ. Thus, we define:

$$\mathcal{R} = \{S_i^1, S_i^2, S_i^1 \cup S_i^2 | \forall < S_i^1, S_i^2 > \in \mathcal{S}\} \cup$$
$$\{\{k\} | \forall k \in P\} - \{P, \phi\}.$$

We say that two sets $R, R' \in \mathcal{R}$ are *connected* in $\mathcal{R}$ if there exists a sequence of sets $R_1 = R, R_2, \cdots, R_k = R'$ in $\mathcal{R}$ such that R_{j-1} intersects R_j for every $j = 2, \ldots, k$. A sub-collection $\mathcal{C} \subseteq \mathcal{R}$ is called a *connected group* in $\mathcal{R}$ if every pair $R, R' \in \mathcal{C}$ is connected in $\mathcal{C}$ and any $R \in \mathcal{C}$ is not connected with any set in $\mathcal{R} - \mathcal{C}$. We also define $C = \bigcup_{R \in \mathcal{C}} R$, i.e., the *union set* of the collection $\mathcal{C}$. It is easy to see that the union sets of all connected groups of $\mathcal{R}$ are disjoint and form a partition of P. The following lemma uses the concept of connected port groups to describe a *necessary condition* for the given set of path constraints to define a unique port order over P.

Lemma 6.1: If the path constraints $\mathcal{S}$ uniquely determine the arrangement of ports in P then the derived contiguity constraints $\mathcal{R}$ satisfy one of the following two conditions: (a) $\mathcal{R}$ comprises a single connected group; or, (b) $\mathcal{R}$ contains two connected groups $\mathcal{C}_1, \mathcal{C}_2 \subset \mathcal{R}$ and $\mathcal{S}$ contains the path constraint $< C_1, C_2 >$ or $< C_2, C_1 >$, where C_i is the union set for group $\mathcal{C}_i$. ∎

Intuitively, the above theorem states that, in order for $\mathcal{S}$ to determine a unique arrangement of P, the contiguity and order constraints in $\mathcal{S}$ should span the entire set of ports in P; otherwise, there would certainly be segments of the path where the port arrangement cannot be determined based on the constraints. Note that Case (b) in Theorem 6.1 could only arise when $C_1 \cup C_2 = P$, since we have excluded the (trivial) contiguity constraint P from $\mathcal{R}$.

The Auxiliary Skeleton Path L. Consider a connected group $\mathcal{C}$ in $\mathcal{R}$, and let $C \subseteq P$ denote its union set. Our goal is to construct a valid port arrangement for the ports in C using the given set of path constraints. Intuitively, our algorithm will accomplish this by building a (coarse) auxiliary skeleton path $L =< U_1, \ldots, U_{|L|} >$ and then recursing on each subset U_i of L, concatenating the results of the recursive calls. However, to be able to recurse independently on each U_i subset using only its "local" set of path constraints, this auxiliary skeleton path L needs to be constructed carefully. Our construction is based on the concept of *Intersecting, Non-Containing (INC)* port sets that we formally define here.

Definition 6.1: Two port sets $R_i, R_j \subset P$ are said to be *Intersecting, Non-Containing (INC)* if and only if they intersect and neither one of them contains the other, i.e., $R_i \cap R_j \neq \phi$, $R_i \not\subseteq R_j$, and $R_j \not\subseteq R_i$. ∎

It is easy to see that having a contiguity constraint R in $\mathcal{C}$ that is INC with one of the subsets U_i in our skeleton path L essentially means that *we cannot independently recurse on that U_i subset*. The problem, of course, is that R would also intersect neighbors of U_i in L and the ports in these sets intersected by R cannot be arranged independently since that would not guarantee that R is satisfied in the final (concatenated) arrangement. On the other hand, recursing on U_i is simple if R is fully contained in or contains U_i: in the former case, R is simply passed as an argument to the recursive call and in the latter R has no effect on the arrangement of U_i since U_i is already required to be contiguous (by the skeleton path definition). Thus, we would like to build an auxiliary path L that is *INC-free for $\mathcal{C}$* as defined below.

Definition 6.2: We say that the skeleton path $L =< U_1, \ldots, U_{|L|} >$ is *INC-free for $\mathcal{C}$* if and only if for every contiguity constraint $R \in \mathcal{C}$ either (a) R is contained in a single $U_j \in L$ (i.e., $R \subseteq U_j$ for some j); or, (b) R is equal to the union of a (sub)sequence of subsets in L (i.e., $R = \bigcup_{j=k_1}^{k_2} U_j$ for some $1 \leq k_1 \leq k_2 \leq |L|$). ∎

The method we use for building a skeleton path L that is INC-free for $\mathcal{C}$ is as follows. Initially, we find the largest port set $R_i \in \mathcal{C}$ and any set $R_j \in R$ that is INC with R_i. (Note that two such sets must exist since $\mathcal{C}$ is a single connected group and the trivial contiguity constraint C is ignored.) From these two sets, we construct an initial skeleton path with three subsets $L =< R_i - R_j, R_i \cap R_j, R_j - R_i >$. (At this point, the orientation of the L path is arbitrary; it is resolved using the given path constraints $\mathcal{S}$ after the whole INC-free path has been built.) Let $L =< U_1, \ldots, U_{|L|} >$ denote the current state of our skeleton path and let P_L be the set of all ports in L. While there exists a set $R \in \mathcal{C}$ that is INC with P_L or one of the subsets $U_j \in L$ (e.g., Figure 7(a)) our algorithm performs the following operations. First, every $U_j \in L$ that is INC with R is replaced by the two subsets $U_j - R$ and $U_j \cap R$. The order of these two subsets in L is determined as follows. If $j < |L|$ and R intersects U_{j+1}, or $j = |L|$ and R does not intersect U_{j-1}, then $U_j - R$ precedes $U_j \cap R$ in L (e.g., the split of U_2 into U_2' and U_3' in Figure 7); otherwise, the two subsets are inserted in the opposite order in L. Second, suppose that R and P_L are INC; this implies that R contains nodes that are not included in the current skeleton path L. After the above splitting of U_j's based on R, it is easy to see that R must completely contain either the first or the last subset of L. If $U_1 \subset R$ then we insert the set $R - P_L$ as the first set of L, i.e., $L =< R - P_L > \circ L$ (where "$\circ$" denotes path concatenation); otherwise, we set $L = L \circ < R - P_L >$ (e.g., attaching U_5' to L in Figure 7). Finally, we update the set $P_L = P_L \cup R$, and return to select a new contiguity constraint R.

Lemma 6.2: Given a single connected group of contiguity constraints $\mathcal{C} \subseteq \mathcal{R}$, the above-described procedure constructs a skeleton path for C that is INC-free for $\mathcal{C}$. ∎

Remember that we built the INC-free path L without paying attention to its orientation. Thus, at this point, either L or REVERSE(L) is the correct skeleton path for C (where the REVERSE function simply reverses the subset order in a given skeleton path). As will become clear in the description of our refinement algorithm, we resolve the orientation for L using
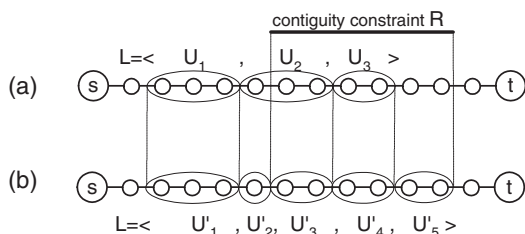
Fig. 7. Building an INC-free auxiliary skeleton path L.

the input set of path constraints $\mathcal{S}$.

The REFINEPATH Algorithm. The detailed pseudo-code for our REFINEPATH procedure is depicted in Figure 8. In its first phase (Steps 2-6), REFINEPATH builds the collection of inferred contiguity constraints $\mathcal{R}$ on P and the resulting connected port groups, and applies Lemma 6.1 to decide if $\mathcal{S}$ can define a unique port arrangement for P. If we discover more than 2 connected groups then, by Theorem 6.1, our algorithm cannot hope to build a skeleton path with ordered subsets of P so it simply returns the trivial skeleton path $L =< P >$. If we find exactly two connected groups (with union sets C_1 and C_2) in $\mathcal{R}$ then procedure ORIENTPATH (described in detail below) is invoked to determine the correct ordering of C_1 and C_2 in the skeleton path using $\mathcal{S}$; then, in Steps 30-37 our algorithm recurses on the two union sets C_1 and C_2 to determine their internal port arrangements and appropriately concatenates the resulting subpaths. Finally, if $\mathcal{R}$ comprises a single connected port group then REFINEPATH builds the auxiliary INC-free skeleton path L as described earlier in this section (Steps 11-26) and uses the ORIENTPATH procedure to determine the correct orientation for L; then, again using Steps 30-37, REFINEPATH recurses on each (non-singleton) subset U_j in the L path using only the constraints local to that subset (i.e., constraints $< S_i^1, S_i^2 >\in \mathcal{S}$ such that $S_i^1 \cup S_i^2 \subseteq U_j$) and concatenates the results of the recursive calls to build the final output path Q.

The ancillary ORIENTPATH procedure (shown in Figure 9) uses the original set of path constraints $\mathcal{S}$ in order to identify the correct direction for an input skeleton path L. ORIENT-PATH relies on the two functions FIRST(L, S) and LAST(L, S) introduced in Section V for identifying the index of the first/last occurrence of an element of S in the L path. More specifically, consider a path constraint $< S_i^1, S_i^2 >\in \mathcal{S}$ such that FIRST$(L, S_i^1) <$ LAST(L, S_i^2); then, clearly, since the constraints in $\mathcal{S}$ characterize a true network path, the ports in S_i^1 should precede those in S_i^2 and, thus, L is the correct skeleton path. Similarly, if there is a path constraint $< S_i^1, S_i^2 >\in \mathcal{S}$ such that FIRST$(L, S_i^2) <$ LAST(L, S_i^1), then the correct path is REVERSE(L). Otherwise, if no constraint in $\mathcal{S}$ can determine the direction of the L path then ORIENTPATH simply returns a trivial single-set skeleton path.

The following theorem establishes the correctness of our algorithm, demonstrating analytically that REFINEPATH always recovers the correct topology for the input ports P as long as it can be identified uniquely from the given set of path constraints $\mathcal{S}$.

Theorem 6.1: The REFINEPATH algorithm returns a feasible skeleton path for the port collection P. Further, if $\mathcal{S}$ uniquely defines the port arrangement in P, then the

---

procedure REFINEPATH$(P, \mathcal{S})$
1. $L =< P >$
2. $\mathcal{R} = \{S_i^1, S_i^2, S_i^1 \cup S_i^2 | \forall < S_i^1, S_i^2 >\in \mathcal{S}\}$
   $\qquad \cup \{\{k\} | \forall k \in P\} - \{P, \phi\}$
3. $\mathcal{C} = \mathcal{R}$
4. **while** there exist $R_i \neq R_j \in \mathcal{C}$ s.t. $R_i \cap R_j \neq \phi$ **do**
5. $\quad \mathcal{C} = \mathcal{C} - \{R_i, R_j\} \cup \{R_i \cup R_j\}$
6. **endwhile**
7. **if** $|\mathcal{C}| > 2$ **then return**(L)
8. **else if** $(\mathcal{C} = \{C_1, C_2\})$ **then**
9. $\quad L =$ ORIENTPATH$(< C_1, C_2 >, \mathcal{S})$
10. **else** // $\mathcal{R}$ comprises a *single* connected group
11. $\quad R_i =$ the largest set in $\mathcal{R}$
12. $\quad R_j =$ set in $\mathcal{R}$ that is INC with R_i.
13. $\quad L = < R_i - R_j, R_i \bigcap R_j, R_j - R_i >; P_L = R_i \bigcup R_j$
14. $\quad$ **while** there exist $R \in \mathcal{R}$ s.t. [(R and P_L are INC) or (R and some $U_j \in L$ are INC)] **do**
15. $\quad\quad$ **for each** $U_j \in L$ that is INC with R **do**
16. $\quad\quad\quad$ **if** [($j < |L|$ and $U_{j+1} \cap R \neq \phi$) or ($j = |L|$ and $U_{j-1} \cap R = \phi$)] **then**
17. $\quad\quad\quad\quad$ replace U_j with $< U_j - R, U_j \cap R >$.
18. $\quad\quad\quad$ **else** replace U_j with $< U_j \cap R, U_j - R >$.
19. $\quad\quad\quad$ **endif**
20. $\quad\quad$ **endfor**
21. $\quad\quad$ **if** R and P_L are INC **then**
22. $\quad\quad\quad$ $P_L = P_L \cup R$
23. $\quad\quad\quad$ **if** $(U_1 \subset R)$ **then** $L = < R - P_L > \circ L$
24. $\quad\quad\quad$ **else** $L = L \circ < R - P_L >$
25. $\quad\quad$ **endif**
26. $\quad$ **endwhile**
27. $\quad L =$ ORIENTPATH$(L, \mathcal{S})$
28. **endif**
29. **if** $|L| = 1$ **then return**$(< P >)$
30. $Q = \epsilon \qquad$ // the empty path
31. **for** $j = 1$ to $|L|$ **do**
32. $\quad$ **if** $(|U_j| = 1)$ **then** $Q = Q \circ < U_j >$
33. $\quad$ **else**
34. $\quad\quad$ $\mathcal{S}_j = \{< S_i^1, S_i^2 >\in \mathcal{S}$ such that $S_i^1 \cup S_i^2 \subseteq U_j\}$
35. $\quad\quad$ $Q = Q \circ$ REFINEPATH$(U_j, \mathcal{S}_j)$
36. $\quad$ **endif**
37. **endfor**
38. **return**(Q)

Fig. 8. The REFINEPATH Procedure.

REFINEPATH algorithm is guaranteed to return the (unique) correct path topology. ∎

VII. INFERRING THE NETWORK TOPOLOGY

The final step of our topology-discovery algorithm is to use the data in the resolved skeleton paths in order to infer the connectivity information for switches and hubs in the underlying network (procedure FINDCONNECTIONS in Figure 2). Given a set of resolved skeleton paths (i.e., path for which a complete port arrangement has been determined), the procedure for inferring element connectivities is fairly straightforward: Ports that are adjacent on some path are directly connected; and, if a port has more than one neighbor in the resolved paths, then a hub is placed to interconnect that port with all its neighboring ports (as well as all other ports connected to ports already on the hub).

The following theorem identifies a strong *completeness* property of our proposed topology-discovery algorithm and is the main theoretical result of this paper. Due to space constraints, a sketch of the proof can be found in the appendix; the complete details can be found in the full version of this

```
procedure ORIENTPATH(L, S)
 1.  for each < S_i^1, S_i^2 >∈ S do
 2.    if ( FIRST(L, S_i^1 ) < LAST(L, S_i^2) ) then return(L)
 3.    else if ( FIRST(L, S_i^2) < LAST(L, S_i^1) ) then
 4.       return(REVERSE(L))
 5.    endif
 6.  endfor
 7.  return(< ⋃_{U_j ∈ L} U_j >)
```

Fig. 9.   The ORIENTPATH Procedure.

(a) The true network topology.

(b) The initial skeleton-paths.

(c) The skeleton-paths after the first iteration.

(d) The skeleton-paths after the second iteration.

(e) The skeleton-paths after the third iteration.

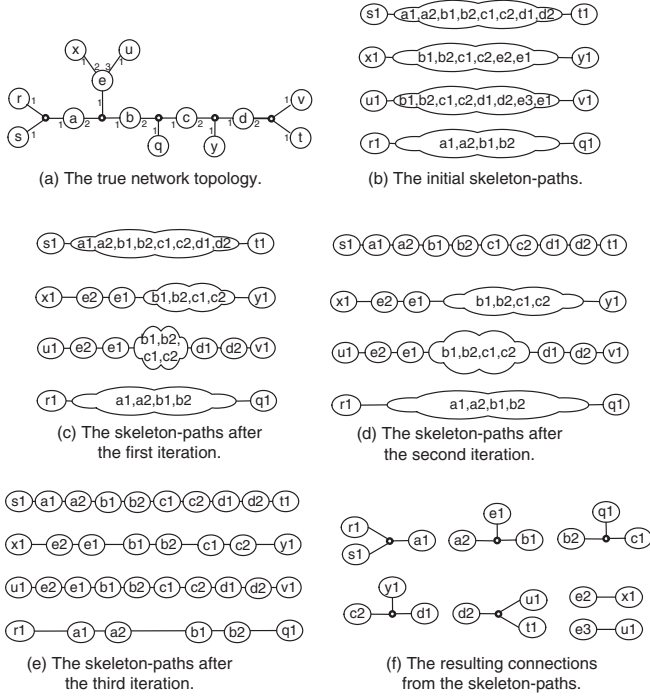(f) The resulting connections from the skeleton-paths.

Fig. 10.   An example execution of our algorithm.

paper [9]. To the best of our knowledge, this is the first result of this type in the area of SNMP-based physical topology discovery.

Theorem 7.1: Our topology-discovery algorithm (depicted in Figure 2) runs in time that is polynomial in the number of network nodes and is *complete* for the given AFT and subnet information. That is, if the input SNMP and subnet data is sufficient to uniquely identify the physical topology of the underlying network, then our algorithm is *guaranteed* to recover that (unique) topology. ∎

VIII. AN EXAMPLE EXECUTION

In this section, we present some key steps of our topology-discovery algorithm in inferring the topology of the example network depicted in Figure 10(a), where we assume that the hosts comprise four different subnets $\{s, t\}$, $\{x, y\}$, $\{u, v\}$ and $\{r, q\}$, and each switch a, b, c, d, e belongs to a different subnet. Our goal is to demonstrate how our algorithm accumulates partial topology information during skeleton-path refinement iterations until the complete network topology is recovered. Let $\mathcal{Q}^i$ denote the skeleton-path collection at the end of the i-th iteration ($\mathcal{Q}^0$ is the initial set). To simplify our discussion, we assume that refinements during the i-th iteration only use skeleton paths in $\mathcal{Q}^{i-1}$.

The initial skeleton path collection $\mathcal{Q}^0 = \{ Q_{s,t}^0, Q_{x,y}^0, Q_{u,v}^0, Q_{r,q}^0 \}$ is shown in Figure 10(b). Suppose that our paths are refined in the order $P_{s,t}$, $P_{x,y}$, $P_{u,v}$, and $P_{r,q}$. For $Q_{s,t}^0$ we compute the collection of path constraints $\mathcal{S}_{s,t}^1$:

$$\mathcal{S}_{s,t}^1 = \left\{ \begin{array}{l} S_1 = <\{s1\}, \{a1, a2, b1, b2, c1, c2, d1, d2\}>, \\ S_2 = <\{a1, a2, b1, b2, c1, c2, d1, d2\}, \{t1\}>, \\ \quad S_3 = <\{b1, b2, c1, c2, d1, d2\}, \phi>, \\ \quad S_4 = <\{b1, b2, c1, c2\}, \phi>, \\ \quad S_5 = <\{a1, a2, b1, b2\}, \phi>, \\ S_6 = <\{a1\}, \{a2\}>, S_7 = <\{b1\}, \{b2\}>, \\ S_8 = <\{c1\}, \{c2\}>, S_9 = <\{d1\}, \{d2\}> \end{array} \right\},$$

where $S_1 - S_5$ follow from the intersections of $Q_{s,t}$ with paths in $\mathcal{Q}^0$ (including $Q_{s,t}$ itself), and $S_6 - S_9$ come from the AFT information at intermediate nodes.

To refine $P_{s,t}$, we use $\mathcal{S}_{s,t}^1$ to compute the INC-free auxiliary path $L_{s,t} = < \{s1\}, \{a1, a2, b1, b2, c1, c2, d1, d2\}, \{t1\} >$ which has the correct orientation (by constraints S_1, S_2). We then recurse on the subset $U_2 = \{a1, a2, b1, b2, c1, c2, d1, d2\} \in L$, and use the constraints "local" to U_2 (i.e., $S_3 - S_5$) to compute the subpath $L' = < \{a1, a2\}, \{b1, b2\}, \{c1, c2\}, \{d1, d2\} >$. Unfortunately, at this point, ORIENTPATH cannot use the input constraints to determine the correct direction for L', so it siply returns the set U_2, which means that the skeleton path returned by REFINEPATH is exactly the same as $Q_{s,t}^0$.

Next, for $Q_{x,y}^0$, we compute the path constraints $\mathcal{S}_{x,y}^1$:

$$\mathcal{S}_{x,y}^1 = \left\{ \begin{array}{l} S_1 = <\{x1\}, \{b1, b2, c1, c2, d1, d2, e2, e1\}>, \\ S_2 = <\{b1, b2, c1, c2, d1, d2, e2, e1\}, \{y1\}>, \\ \quad S_3 = <\{b1, b2, c1, c2, d1, d2\}, \phi>, \\ \quad S_4 = <\{b1, b2, c1, c2, e1\}, \phi>, \\ \quad S_5 = <\{b1, b2\}, \phi>, \\ S_6 = <\{b1\}, \{b2\}>, S_7 = <\{c1\}, \{c2\}>, \\ S_8 = <\{d1\}, \{d2\}>, S_9 = <\{e2\}, \{e1\}> \end{array} \right\}.$$

To refine $P_{x,y}$, REFINEPATH computes the INC-free auxiliary path $L_{x,y} = < \{x1\}, \{b1, b2, c1, c2, d1, d2, e2, e1\}, \{y1\} >$, and recurses to refine its second subset $U_2 = \{b1, b2, c1, c2, d1, d2, e2, e1\} \in L_{x,y}$. Using constraints S_3, S_4, S_5 and S_9, it computes the subpath $L' = < \{d1, d2\}, \{b1, b2, c1, c2\}, \{e1\}, \{e2\} >$. Then, by constraint S_9, ORIENTPATH concludes the reverse direction for L', returning the final subpath $< \{e2\}, \{e1\}, \{b1, b2, c1, c2\}, \{d1, d2\} >$. Additional recursive calls resolve the port order for subset $\{d1, d2\}$ but not for subset $\{b1, b2, c1, c2\}$; thus, the final $x - y$ skeleton path returned is $Q_{x,y}^1 = < \{e2\}, \{e1\}, \{b1, b2, c1, c2\}, \{d1\}, \{d2\} >$. The other two refined skeleton paths $Q_{u,v}^1$ and $Q_{r,q}^1$ are computed similarly, and the path collection $\mathcal{Q}^1$ is shown in Figure 10(c).

Note that, after the first refinement iteration, none of the paths in $\mathcal{Q}^1$ specifies a complete arrangement. However, as we now show, the refined path $Q_{x,y}^1 \in \mathcal{Q}^1$ allows us to refine $Q_{s,t}$ in the second iteration of our algorithm. Consider the set of path constraints $\mathcal{S}_{s,t}^2$ computed for $Q_{s,t}$ during the second iteration. This set is identical to $\mathcal{S}_{s,t}^1$, with the exception of constraint S_3 (resulting from the projection of $Q_{x,y}$ onto $Q_{s,t}$); more specifically, constraint S_3 for this second iteration over $Q_{s,t}$ is $S_3 = < \{\{b1, b2, c1, c2\}, \{d1, d2\} >$. Thus, after REFINEPATH recomputes the subpath $L' = < \{a1, a2\}, \{b1, b2\}, \{c1, c2\}, \{d1, d2\} >$, constraint S_3 can now be used by ORIENTPATH to determine the correct direction for L', and the resulting $s - t$ skele-

ton path returned is $Q_{s,t}^2 = < \{s1\}, \{a1\}, \{a2\}, \{b1\}, \{b2\}, \{c1\}, \{c2\}, \{d1\}, \{d2\}, \{t1\} >$ (Figure 10(d)).

In its third iteration, our topology-discovery algorithm actually recovers the complete port arrangement for all skeleton paths as shown in Figure 10(e). Finally, the FINDCONNECTIONS procedure uses the resolved paths to discover the element connectivities depicted in Figure 10(f). It is easy to see that the connections discovered specify exactly the true network topology shown in Figure 10(a).

IX. CONCLUSIONS

Automatic discovery of physical topology information plays a crucial role in enhancing the manageability of modern IP networks. In this paper, we have proposed the *first* complete algorithmic solution for discovering the physical topology of a large, heterogeneous Ethernet network comprising multiple subnets as well as (possibly) dumb or uncooperative network elements. Our proposed algorithm relies on standard SNMP MIB information that is widely supported in modern IP networks and is the first SNMP-based topology-discovery tool to offer strong completeness guarantees for recovering the true network topology from the given MIB data.

REFERENCES

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

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

[3] A. Bierman and K. Jones, "Physical Topology MIB", Sept. 2000, Internet RFC-2922 (available from `http://www.ietf.org/rfc/`).

[4] S. Keshav, *"An Engineering Approach to Computer Networking"*. Addison-Wesley Professional Computing Series, 1997.

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

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

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

[8] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, S. Seshadri, and A. Silberschatz, "Topology Discovery in Heterogeneous IP Networks", in *Proceedings of IEEE INFOCOM'2000*, Tel Aviv, Israel, Mar. 2000.

[9] Y. Bejerano, Y. Breitbart, M. Garofalakis, and R. Rastogi, "Physical Topology Discovery for Large Multi-Subnet Networks", July 2002, Bell Labs Tech. Memorandum.

APPENDIX

Proof Sketch for Theorem 7.1. Let $\mathcal{Q}^0$ be the initial collection of skeleton-paths calculated by the INITSKELETONPATHS procedure and let $\mathcal{Q}^i$, $i > 0$, be the skeleton-path collection at the end of the i-th iteration. Initially, we prove that the skeleton-path collection $\mathcal{Q}^i$, for every $i \geq 0$, comprises all the AFT information. We consider the connecting tree T^N of every subnet N and we construct an iterative algorithm that computes the complete AFTs for subnet N of every node $v \in T^N$. The algorithm first populates the AFTs of node v with the end-points of the skeleton-paths comprising v. Then, iteratively, it finds which port of v leads to the nodes of the other skeleton-paths in T^N. As a result, any topology that satisfies the skeleton-path constraints also satisfies the network AFT information.

Next, we consider our REFINEPATH procedure and we prove that if the constraint collection $\mathcal{S}$ uniquely defines the port order of the path $\hat{P}$, then this procedure returns the correct and complete path topology. This theorem results from the recursive behavior of the REFINEPATH procedure. The procedure constructs an auxiliary skeleton path $L = < U_1, \ldots, U_{|L|} >$ by using only the contiguity constraints in $\mathcal{S}$, such that every pair of $U_i, U_j \in L$, $i \neq j$ are INC-free. Thus, the internal arrangement of the ports in every set $U_i \in L$ does not affect the port order of any other set $U_j \in L$. This enables the procedure to recursively refine any set $U_i \in L$ until sets with single port-ids are obtained. At this point, either L or REVERSE(L) is the correct skeleton path for $\hat{P}$. By employing the order constraints in $\mathcal{S}$, the ancillary ORIENTPATH procedure identifies the correct direction of L at the end of each invocation of the REFINEPATH procedure.

For completing the proof, we only have to show that our COMPUTECONSTRAINTS procedure computes *all* the contiguity and order constraints that are essential for determining the network topology. We consider first the contiguity constraints. Since such constraints essentially result from path-intersection operations, we show that all the contiguity constraints can be deduced directly from the initial skeleton paths, $\mathcal{Q}^0$.

Identifying all the required order constraints in more challenging task. Recall that the order of a skeleton path L_1 may be enforced by the order of another skeleton path L_2 as a result of an implicit order constraint (even when the two paths do not have any port-id in common). In such cases, we can determine the required order constraints of L_1 only after refining the skeleton-path L_2. We resolve this problem as follows. For simplicity, we assume that all the skeleton-paths are computed in advance without determining their directions and let $\mathcal{L}$ denote the collection of computed skeleton-paths. Then, we divide $\mathcal{L}$ into disjoint sets $\{\mathcal{L}_1, \mathcal{L}_2, \ldots, \mathcal{L}_M\}$ such that: (a) for every pair $L_1, L_2 \in \mathcal{L}_k$, $k \in [1, \ldots, M]$, the orderings of paths L_1 and L_2 are dependent (i.e., ordering one path determines the order of the other); and, (b) for any pair $L_1 \in \mathcal{L}_k$, $L_2 \in \mathcal{L}_{k'}$, $k \neq k'$, paths L_1 and L_2 are order-independent. We prove that each such set $\mathcal{L}_k$ has the following two properties: (a) Every $\mathcal{L}_k$ induces a connected component; in other words, for every $L', L'' \in \mathcal{L}_k$ there is a sequence $\{L_1 = L', L_2, \ldots, L_l = L''\}$ such, that for every $i \in [1, \ldots, (l-1)]$, paths L_i and L_{i+1} intersect; and, (b) If $\mathcal{S}$ defines a unique network topology then every $\mathcal{L}_k$ contains at least one explicit order constraint. By using these two properties, we can show that the **do-while** loop of our TOPOLOGYDISCOVERY algorithm ensures the inference of all explicit order constraints for each skeleton-path. Finally, we tie all our deductions together and conclude that if the AFTs define unique network then our TOPOLOGYDISCOVERY algorithm will identify it.

Our running-time analysis is based on the observation that the number of skeleton-paths and path constraints that our TOPOLOGYDISCOVERY algorithm computes are $O(n^2)$ and $O(n^3)$, respectively. The algorithm contains several finite loops, where set operations are performed on the skeleton-paths or the constraints. Therefore, our algorithm's running time is polynomial and, in fact, it is comparable to that of existing techniques for layer-2 topology discovery [8], [7]. ∎