Discovering Topology of MultiSubnet LAN using MIB

Sapna Angel
Lecture, Department of CSE
HKBK College of Engineering, Bangalore

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.

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. Although network topology information, especially at the LAN level, is important for both the management and use of networks, it is very difficult to obtain such information. The majorities of network-management tools relies only on IP-level discovery of routers and require the network manager to enter level 2 devices such as Ethernet switches manually, without providing facilities for topology discovery. Cisco, Intel, and other hardware providers have designed their own proprietary topology discovery protocols, but these are of little use in a heterogeneous environment.

More recently, the IETF has acknowledged the importance of LAN topology by designating an SNMP MIB to describe topology, but it failed to define a protocol for obtaining that topology

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

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

A. *REMOS*

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

B. *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 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 is oblivious to the hosts in subnet.

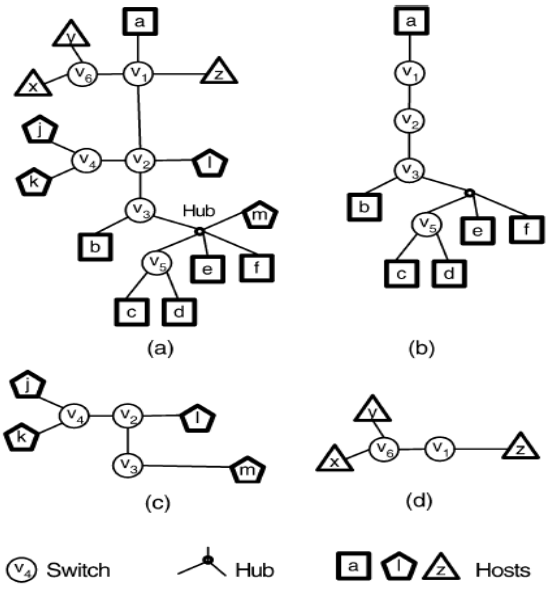

Fig. 1. An example of a network and its subnet connecting trees. (a) The network. (b) The connecting tree of subnet N1. (c) The connecting tree of subnet N2. (d) The connecting tree of subnet N3.

NETWORK MODEL

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

Bridges connect Ethernet segments together in an acyclic network. If cycles are present in the physical connections, the bridges use the Ethernet spanning tree algorithm to select an acyclic subset of those connections. Once the topology has been selected, bridges learn the location of machines on the network by monitoring traffic on each of their ports. When a new node sends a message that appears on that port, the bridge adds it to the list of nodes found off that port. Later, when the bridge receives a message bound for that node, it knows to forward the message to that port. The database that maps addresses to ports is known as the forwarding database (FDB).

For a bridge, *C*, we denote the forwarding set for port *x* as F_C^x .

The first step in discovering a network's topology is determining how a pair of bridges is connected. Two nodes are referred to as *directly connected* if there are no other nodes between them. In particular, if the packets that bridge *A* sends on port *x* are received by port *y* on bridge *B* without going through any other device, bridges *A* and *B* are directly connected via ports *x* and *y*, respectively.

In Figure 1, A and B are directly connected by ports 1 and 4, as are B and D by ports 2 and 2. Furthermore, two bridges are *simply connected* by ports *x* and *y* if they find each other off those ports, but there may be other nodes in between. A and C are simply connected by ports 1 and 1. Directly connected nodes, such as A and B, are also simply connected by ports 1 and 4. At a high level, this algorithm begins by determining all entries in the FDBs, ensuring they are complete. It then selects a single bridge and determines the bridges that are directly connected to each port. The Ethernet topology is then filled in by traversal. At the heart of this algorithm is the direct connection theorem, which is used to determine when two bridges are directly connected.

OVERVIEW OF THE SCHEME

The goal of our *topology discovery* (TD) scheme is discovering the physical topology of the spanning tree(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.

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

THE SKELETON TREE CREATION ALGORITHM

The Skeleton Tree Creation Algorithm (STC) 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 $\upsilon \subseteq V^N$ to H, until all the labeled nodes in V^N are represented in H. that denotes the n_v value of these nodes as defined by Equation 2. In the case that vertex $\mathcal{Y}$ denotes a segment of transit nodes, then all the nodes $\upsilon \in C_{\mathcal{Y}}$ 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 $\upsilon \in C_{\mathcal{Y}}$ During the initialization stage, the algorithm finds the root-port, $\upsilon(r)$,for each node $\upsilon \in V^N$ 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 $\mathcal{Y}$ that represents the root r, *i.e.*, $C_{\mathcal{Y}} = \{r\}$ and $n_{\mathcal{Y}} = |N| + 1/2$. 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 υ^{1}, 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 υ^{1}. Since a is the only frontier arc that represents a link along the path $P_{r,\upsilon}$ from the root r to node υ^{1} in T^{N}, $B_a \supseteq B_{\upsilon^{1}}$. 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 $n_y = n_{\upsilon^{1}}$ (and they are integer numbers) then node υ^{1} is a transit node included in the segment represented by vertex y. In such case, node υ^{1} is added to C_y. When $n_y > n_{\upsilon^{1}}$ then node υ^{1} is represented by a new vertex y^{1} in H. The vertex y^{1} is associated with set $C_{y^{1}} = \{\upsilon^{1}\}$ and a value $n_{y^{1}} = n_{\upsilon^{1}}$. For every leaf-port $k \in D^{N}_{\upsilon^{1}}\text{-}\{\upsilon(r)\}$ of υ^{1}, the algorithm creates a new frontier arc a^{1} to Z incident from vertex y^{1}, it associates the new arc a^{1} with the set $B_{a^{1}} = F^{N}_{\upsilon^{1},k}$ and ads a^{1} to Z. Now, if $B_a = B_{\upsilon^{1}}$, there are no intermediate nodes between the node represented by vertex y and node υ^{1}. Thus, the algorithm connects arc a to the vertex y^{1} and removes a from the frontier arc set Z,

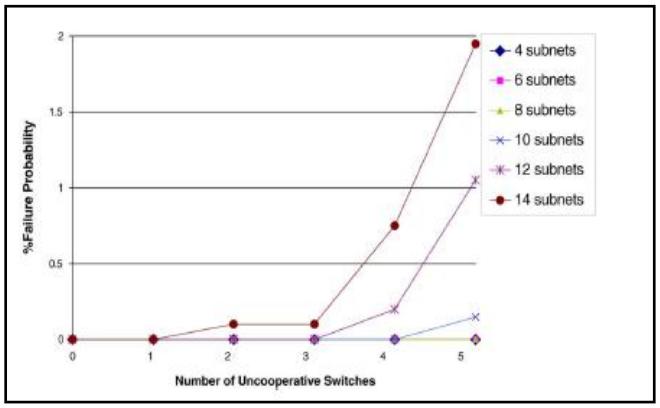

Fig 2 Simulation results of networks with 10 switches (each one with 24 ports), 10 dumb-hubs (each one with eight ports), and 200 hosts.

ACKNOWLEDGMENT

The author sincerely thanks Mr. Brietbart for setting a path into topology discovery schemes for layer 2, and also I thank the esteemed organization HKBKCE, and all my collegeuaes for helping me to carry out the research.

REFERENCES

[1] Bierman and K. Jones, "Physical topology MIB," Internet RFC-2922, Sep. 2000 [Online]. Available: www.ietf.org/rfc/

[2] B. Donnet and T. Friedman, "Internet topology discovery: A survey," *IEEE Commun. Surveys & Tutorials*, vol. 9, no. 4, pp. 56–69, 2007.

[3] Y. Breitbart, M. Garofalakis, B. Jai, C. Martin, R. Rastogi, and A. Silberschatz, "Topology Discovery in Heterogeneous IP Networks: The NetInventory System", *IEEE/ACM Transactions on Networking*, vol. 12, no. 3, June 2004, pp. 401-414.

[4] Y. Breitbart and H. Gobjuka, "Characterization of layer-2 unique topologies", *Information Processing Letters*, vol. 105, no. 2, Jan. 2008, pp. 52-57.

[5] J. Liang, R. Kumar, and K. W. Ross, "The Kazaa Overlay: A Measurement Study," *Computer Networks*, vol. 49, no. 6, Oct.2005.

[6] Z. Wen, S. Triukose, and M. Rabinovich, "Facilitating Focused Internet Measurements," *Proc. ACM SIGMETRICS*, June 2007.

[7] Layer-2 Path Discovery Using Spanning Tree MIBs David T. Stott Avaya Labs Research, Avaya Inc.2002

[8] Muhammad Azizur Rahman, Algirdas Pakštas, Frank Zhigang Wang, "NeDaSE: A Bridge Between Network Topology Generator, Network Design and Simulation Tools", *Proc. of the EUROCON 2005 - The International IEEE Conference on Computer as a tool*, November 21-24, 2005,Sava Center, Belgrade, Serbia & Montenegro.

[9] R. Black , A. Donnelly, C. Fournet, Ethernet Topology Discovery without Network Assistance, In Proceedings of International Conference on Network Protocols (ICNP), 2004, pp. 328-339.