PAPER *Special Section on Internet Technology and its Architecture for Ambient Information Systems*

Ethernet Topology Detection from a Single Host without Assistance of Network Nodes or Other Hosts

Yohei HASEGAWA[†a)] ***and*** **Masahiro JIBIKI**[†], ***Members***

SUMMARY Topology information has become more important for management of LANs due to the increasing number of hosts attached to a LAN. We describe three Ethernet topology discovery techniques that can be used even in LANs with Ethernet switches that have no management functionality. Our "Shared Switch Detection (SSD)" technique detects the Ethernet tree topology by testing whether two paths in the network share a switch. SSD uses only general MAC address learning. By borrowing MAC addresses from hosts, SSD can be run from a single host. The second technique determines whether two paths between two pairs of hosts contain a switch. The third reduces the number of shared switch detections. Simulation showed that these techniques can be used to detect the Ethernet topology with a reasonable search cost. Examination on a real-world testbed showed that they could detect an Ethernet topology consisting of six hosts and two switches within one second.

key words: *Ethernet, topology detection, network probing*

1. Introduction

The demands for communication quality and connectivity within LANs have recently become more severe because the number of hosts in a LAN has been increasing and quality sensitive applications such as audio and video streaming have become more popular. For example, in an enterprise LAN, such as a data center, hundreds of hosts are usually configured for large-scale service. Even in a home LAN, various kinds of home electronic devices such as televisions, video-game machines, mobile phones, and cameras are attached. Therefore, information about the topology of LANs is becoming more important.

Ordinary topology detection techniques often require the network routers and end-hosts to have specific functionalities. For example, some topology detection techniques need to gather forwarding table information from a network router and/or switches [5], [6]. Other techniques require a router to respond [1], [3], [4]. Even an end-host-based topology detection technique needs other hosts to respond to packets only for topology detection. However, Ethernet switches in LANs usually do not support network management functionalities for topology detection. And hosts in LAN, such as network printers, have no special functionality for topology detection.

We thus need topology detection techniques that can be run using only the generally available functionalities of hosts and switches. We have developed three techniques for Ethernet topology detection.

- A technique for topology detection, based on shared switch detection (SSD), that tests whether two paths share a switch.
- A technique for performing SSD from a single host in a switched Ethernet network.
- A technique to reduce the number of SSDs needed for topology detection.

These techniques enable a single host to detect the topology of an Ethernet network consisting of switches with no management functionality. They use only basic Ethernet forwarding functionality and general request-replies (ICMP ECHO REPLY, TCP SYN-ACK, etc.) from hosts in the network.

The rest of this paper is organized as follows. We briefly review related work in Sect. 2. We describe our techniques for Ethernet topology detection in Sect. 3. In Sect. 4, we describe the evaluation of these techniques in a simulation environment and on a test bed system in a real-world environment.

2. Related Work

We review related work on topology detection by describing the three main approaches.

In the first approach, the IP forwarding table MIB [13] (management information base), which contains information gathered from routers and/or switches via SNMP [12], is used. This approach can be adapted to various kinds of networks including Ethernet ones. Inference techniques [5], [6] have been proposed for use when a complete forwarding table with information from all switches and routers is unavailable. However, this approach still needs to access MIB information for most of the routers and switches in the network through the their management functionalities. This approach is thus not suitable when that access privilege is not granted to the one wanting to obtain topology information. Moreover, this approach cannot be used to detect the topology of networks consisting of switches that have no (SNMP) management functionality.

In the second approach [1], [3], [4], the network topology is determined by grouping sets of addresses obtained using a probing tool such as traceroute on ICMP or Ethernet OAM [9], [11]. Rocketfuel [3] is a set of techniques that implement this approach/router-level mapping tool where one of the underlying ideas is to focus on one specific ISP net-

Manuscript received July 28, 2008.
Manuscript revised November 17, 2008.

[†]The authors are with System Platforms Research Laboratory, NEC Corporation Inc., Kawasaki-shi, 211-8666 Japan.
a) E-mail: y-hasegawa@bk.jp.nec.com

DOI: 10.1587/transcom.E92.B.1128

work at the time and to map it as completely as possible. It uses techniques that reduce the number of probe packets needed to infer the network topology.

This approach can be used in several situations because it can be used to detect the network topology without having network administrator privileges. The recently standardized Ethernet OAM (IEEE802.1ag [10]) can be used for Ethernet level probing using, for example, the ping and traceroute tools. However, Ethernet OAM cannot detect the topology of a network containing switches without Ethernet OAM functionality.

In the third approach, the network topology is inferred by monitoring packet arrival patterns at receiving hosts [7], [8]. The multicast-based topology detection technique [7] refers to the packet loss pattern. If multiple hosts did not receive a particular packet, they are assumed to be in the same part of the multicast tree and to share the link in which the packet was dropped. However, this technique still requires that all the hosts have special functionality for topology detection.

In short, previously reported topology detection techniques are often unable to detect the network topology because they require that the routers, switches, and/or hosts support various functionalities. This is especially true for Ethernet networks because the LAN switches rarely have network management functionality. Network administrators need a topology detection technique that does not depend on the Ethernet switches and end hosts having management functionality.

3. Proposed Techniques

Our techniques for Ethernet topology detection overcome the problems described above. The first detects the tree network topology by determining whether two paths in the network share a switch. The second determines whether two paths between two pairs of hosts contain a switch. The third reduces the number of shared switch detections.

3.1 Topology Detection Based on SSD

Our topology detection technique is designed to detect a logical Ethernet tree topology constructed using a spanning tree algorithm. The redundant parts of the network that have no path branches are omitted. For example, in the three-host topology shown in Fig. 1(a), all switches except s_1 are omitted because they have only two links and no path branch.

We use the following definitions to describe our topology detection technique.

Definition 1:
In a network with a known topology, x denotes the number of hosts, and y denotes the number of switches. Let $n_i(i = 1, 2, \ldots x)$ denote hosts that have already been detected, and let $s_j(j = 1, 2, \ldots y)$ denote switches that have already been detected. l_A^B is a bi-directional link connecting nodes A and B, which are a switch or a host. Note that l_A^B is equal to l_B^A. Let $P_{A\leftrightarrow B}$ denote the set of switches and links located on the path between hosts A and B. For example, the tree network shown in Fig. 1(a) contains $P_{n_1\leftrightarrow n_2}$ such that $P_{n_1\leftrightarrow n_2} = \{l_{s_1}^{n_1}, s_1, l_{n_2}^{s_1}\}$.

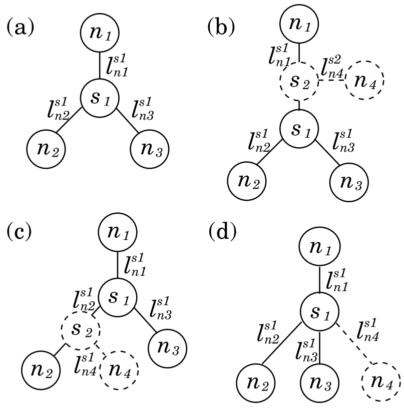


Fig. 1 Example of determining connecting point of n_{x+1}.

Definition 2:
Let P_u $(u = 1, 2, \ldots {}_xC_2)$ denote the set of switches and links on a path between known hosts. $\mathbb{P}$ is the set of P_us:

$$\mathbb{P} = \sum_{u=1}^{{}_xC_2} P_u = \{P_{n_a\leftrightarrow n_b}\} \quad (a, b) = \left(\begin{array}{c} x \\ 2. \end{array} \right)^{\dagger}.$$

$\hat{P}_v(v = 1, 2, \cdots x)$ is the set of switches and links on the path between a known and unknown host, n_{x+1}. $\hat{\mathbb{P}}$ is the set of possible $\hat{P}_v$s:

$$\hat{\mathbb{P}} = \sum_{v=1}^{x} \hat{P}_v = \{P_{n_{x+1}\leftrightarrow n_i}\} \quad (i = 1, 2, \cdots x).$$

Definition 3:
The function $\boldsymbol{f}(P_u, \hat{P}_v)$ tests whether any switches or links are in both P_u and $\hat{P}_v$. Although elements in $\hat{P}_v$ are actually unknown, the function $\boldsymbol{f}$ can detect that P_u and $\hat{P}_v$ have the same element. We describe the procedure for function $\boldsymbol{f}$ in Definition 4. It returns a "1" when two paths share a switch and "0" otherwise.

$$\boldsymbol{f}(P_u, \hat{P}_v) = \begin{cases} 1 & \exists s_k \in P_u : s_k \in \hat{P}_v \parallel \exists l_h \in P_u : l_h \in \hat{P}_v \\ 0 & \forall s_k \in P_u : s_k \notin \hat{P}_v \parallel \forall l_h \in P_u : l_h \notin \hat{P}_v \end{cases}$$

Definition 4:
Host n_{x+1} is an unknown host that has not been detected. U is the set of candidate points (switches and links) to which n_{x+1} might be connected. U contains all detected switches and links in the network. U_0 is initial value of U. U_0 is equal to $\mathbb{P}$.

We define procedure $\boldsymbol{F}$ and use it to calculate $\boldsymbol{f}$ for all possible combinations of P_u and $\hat{P}_v$. For $\boldsymbol{f}(P_u, \hat{P}_v) = 0$, P_u and $\hat{P}_v$ do not have the same elements, so the switches and links in P_u cannot be connected to n_{x+1}. Therefore,

[†] $\binom{x}{2}$ means ${}_xC_2$ pairs chosen; e.g., $\binom{3}{2}$ is the set of (1,2), (1,3), and (2,3).

the switches and links in P_u are excluded from U when $\boldsymbol{f}(P_u, \hat{P}_v) = 0$. The procedure is as follows.

```
$\boldsymbol{F}(U)$ {
    $for\ ({}^{\forall u}P_u, {}^{\forall v}\hat{P}_v)$ {
        $if\ (\boldsymbol{f}(P_u, \hat{P}_v) = 0)\ then\ U = U \wedge \overline{P_u}^{\dagger}$
    } $returnU$
}
```

If, after $\boldsymbol{F}$ is applied, candidate set U has only one link, that link is connecting point t for the unknown host, n_{x+1}. If it has only one switch and multiple links connected to the switch, the switch is the connecting point. After the connecting point has been determined, the unknown host is added to the network.

1. If the connecting point is a link, the link is divided by adding new switch s_{y+1}. Host n_{x+1} is connected to this switch via new link $l^{n_{x+1}}_{s_{y+1}}$.
2. If the connecting point is a switch, e.g., s_l, the unknown host is connected to the switch via new link $l^{n_{x+1}}_{s_l}$

Once the connecting point for the unknown host has been determined, the host is connected to the connecting point. If there are multiple unknown hosts, this procedure is repeated for each, one by one.

Let's consider an example of adding an unknown host using our topology detection technique. In Fig. 1(a), there are three known hosts, n_1, n_2, and n_3, connected to switch s_1. Unknown host n_4 needs to be added to the network.

We assume that the connecting point for n_4 is link $l^{s1}_{n_1}$, as shown in Fig. 1(b). Procedure $\boldsymbol{F}$ is used to examine function $\boldsymbol{f}$ with some combinations of P_u and $\hat{P}_v$. If P_u and $\hat{P}_v$ are such that $P_u = P_{n_2 \leftrightarrow n_3}$ and $\hat{P}_v = P_{n_1 \leftrightarrow n_4}$, $\boldsymbol{f}$ is equal to zero: $\boldsymbol{f}(P_{n_2 \leftrightarrow n_3}, P_{n_1 \leftrightarrow n_4}) = 0$. Note that P_u contains two links and one switch: $P_u = P_{n_2 \leftrightarrow n_3} = \{l^{s_1}_{n_2}, s_1, l^{s_1}_{n_3}\}$. Since set U_0 contains the candidate connecting points (switches and links), $U_0 = \{s_1, l^{s_1}_{n_1}, l^{s_1}_{n_2}, l^{s_1}_{n_3}\}$. One link is left in U:

$$U = U_0 \wedge \overline{P}_{n_2 \leftrightarrow n_3} = l^{s1}_{n_1}.$$

Therefore, the connecting point for n_4 is determined to be link $l^{s1}_{n_1}$. New switch s_2 is added to that link, and host n_4 is connected via new link $l^{n_4}_{s_2}$. If the connecting point were another link, as illustrated in Fig. 1(c), the connecting point would be similarly determined.

If n_4 was connected to switch s_1, as illustrated in Fig. 1(d), every path would contain s_1. Therefore, no element would be excluded from U. However, U has only one switch after procedure $\boldsymbol{F}$, so our detection technique judges that switch s_1 is the connecting point of n_4. Therefore, n_4 is attached to s_1 via new link $l^{n_4}_{s_1}$.

3.2 Proof for Completeness of Topology Detection

Here we prove that our technique can determine the correct connecting point for unknown hosts. First, we describe a case in which there is only one switch in the network. Then, we extend the proof to the case in which there are $y (y \geq 2)$ switches in the network.

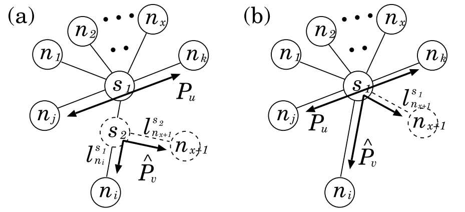


Fig. 2 Paths between hosts in network with one switch: (a) connecting point t is a link; (b) t is a switch.

Case A: One-switch topology

We describe two cases for the one-switch network: the connecting point of the unknown host is a link; it is a switch.

A-1: Link connecting point

We assume that unknown host n_{x+1} is connected to link $l^{n_i}_{s_1}$ as illustrated in Fig. 2(a). The initial candidate for connecting point U_0 is $U_0 = \{s_1, l^{n_k}_{s_1}\}(k = 1, 2, \cdots x)$. The n_j and n_k represent hosts except n_i $(j \neq k,\ j \neq i,\ k \neq i)$. A path from n_j to n_k never contains $l^{n_i}_{s_1}$.

Thus, the following P_us and $\hat{P}_v$s do not contain the same elements, as shown by equation (1).

$$\begin{aligned} P_u &= \{l^{s_1}_{n_j}, s_1, l^{s_1}_{n_k}\} \\ \hat{P}_v &= \{l^{s_1}_{n_i}\} \\ (j, k) &= \begin{pmatrix} x \\ 2 \end{pmatrix}, j \neq i, k \neq i \end{aligned} \tag{1}$$

Therefore, with this P_u and $\hat{P}_v$, the function $\boldsymbol{f}$ is always zero: $\boldsymbol{f}(P_u, \hat{P}_v) = 0$. This means that U can be calculated using procedure $\boldsymbol{F}$:

$$U = U_0 \wedge \overline{\{l^{s1}_{n_j}, s_1, l^{s_1}_{n_k}\}}.$$

With the precondition of equation (1), as

$$\overline{\{l^{s1}_{n_j}, s_1, l^{s_1}_{n_k}\}} = l^{s_1}_{n_i}.$$

Finally, U contains only one link:

$$U = U_0 \wedge l^{s_1}_{n_i} = l^{s_1}_{n_i}.$$

A-2: Switch connecting point

Next, we assume that the connecting point is switch s_1, as illustrated in Fig. 2(b). In this case, the paths between the unknown host and the known hosts always pass through s_1 because all known hosts are connected to switch s_1. Moreover, the paths between the known hosts also pass through s_1 because there is only one switch connecting them. Thus, ${}^{\forall u}P_u \ni s_1$ and ${}^{\forall v}\hat{P}_v \ni s_1$. Therefore, the result of $\boldsymbol{f}$ is always one.

$$\boldsymbol{f} = ({}^{\forall u}P_u, {}^{\forall v}\hat{P}_v) = 1$$

This means that candidates will never be excluded from U. However, suppose that a connecting point could be any link.

[†] $\overline{P_u}$ is a complement set of P_u.

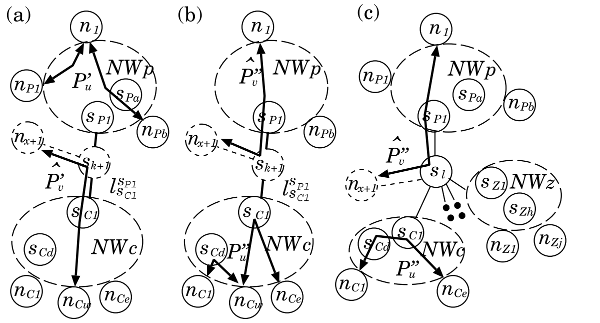


Fig. 3 Networks with switches: (a) connecting point t is a link; (b) t is a switch. (c) network with unknown host n_{x+1} connected to switch s_l.

Table 1 List of elements in network with unknown host n_{x+1} connected to link $l^{s_{c1}}_{s_{p1}}$.

$l^{s_{c1}}_{s_{p1}}$	link that is connecting point t.
s_{p1}	switch connected to root side of $l^{s_{c1}}_{s_{p1}}$
s_{c1}	switch connected to leaf side of $l^{s_{c1}}_{s_{p1}}$
NW_p	set of links and switches of root side of $l^{s_{c1}}_{s_{p1}}$
$s_{p1}, \ldots s_{pa}$	switches in NW_p
$n_{p1}, \ldots n_{pb}$	hosts connected to links in NW_p
NW_c	set of links and switches of leaf side of $l^{s_{c1}}_{s_{p1}}$
$s_{c1}, \ldots s_{cd}$	switches in NW_c
$n_{c1}, \ldots n_{c_e}$	hosts in NW_c

There must be case in which $\boldsymbol{f} = 0$, as illustrated in Fig. 2(a). Therefore, a connecting point being a link is incompatible with $\boldsymbol{f} = 1$ for any P_u and $\hat{P}_v$. This means that the connecting point must be a switch if $\boldsymbol{f} = 1$ for any P_u and $\hat{P}_v$.

Case B: Multiple-switch topology

Next, we describe the case in which the network has y switches. Again, we describe two cases.

B-1: Link connecting point

We assume that unknown host n_{x+1} is connected to link $l^{s_{c1}}_{s_{p1}}$. The network topology is shown in Fig. 3(a). The elements in the network are listed in Table 1.

The initial value of candidate set U containing all links and switches in the network is

$$U_0 = \{l^{s_{C1}}_{s_{P1}}, NW_p, NW_c\}. \tag{2}$$

We use the following theorem, which is true because the assumed tree network has no loop and all the leaves of the network are hosts (a detailed proof is given in Appendix).

Theorem 1: paths from any host in a network to the other hosts can pass through any switch or link in the network.

$$^{\forall z}P_{n_1 \leftrightarrow n_z} = \mathbb{P}$$

As illustrated in Fig. 3(a), NW_p is part of the network. Let $\mathbb{P}'$ denote the sum of the links and switches on paths P'_u, which are the paths from root host n_{P1} to the hosts connected to NW_p. Given theorem 1, we can describe $\mathbb{P}'$ as

$$\mathbb{P}' = \sum_{u=1}^{h-1} P'_u = \{P_{n_{P1} \leftrightarrow n_{Ph}}\} = NW_p.$$

$$(h = 2, 3, \ldots b) \tag{3}$$

We choose $\hat{P}'_v$ such that

$$\hat{P}'_v = P_{n_{x+1} \leftrightarrow n_{Cw}},$$

where host n_{Cw} is connected to NW_c.

As evident from Fig. 3(a), $\hat{P}'_v$ is obviously always exclusive to NW_p. Therefore,

$$\boldsymbol{f}(^{\forall u}P'_u, \hat{P}'_v) = 0.$$

Then, with equations 2 and 3, procedure $\boldsymbol{F}$ can subtract all links and switches from U_0:

$$U_1 = U_0 \wedge \{^{\forall u}\overline{P'_u}\} = U_0 \wedge \overline{NW_p} = \{l^{s_{C1}}_{s_{P1}}, NW_c\}.$$

Similarly, in accordance with theorem 1, the set of links and switches on the paths between n_{C1} and $n_{Ch}(h = 2, \ldots e)$, as illustrated in Fig. 3(b), includes all links and switches in NW_c.

Let $\mathbb{P}''$ denote the sum of P''_u, the set of links and switches on the path between hosts n_{C1} and $n_{Ch}(h = 2, 3, \ldots e)$. Therefore,

$$\hat{\mathbb{P}}'' = \sum_{u=1}^{e-1} P''_u = \{^{\forall h}P_{n_{C1} \leftrightarrow n_{Ch}}\} = NW_c.$$

$$(h = 2, 3, \ldots e) \tag{4}$$

If we set $\hat{P}''_v$ as follows, P''_u and $\hat{P}''_v$ do not share any switch.

$$\hat{P}_v{}'' = P_{n_1 \leftrightarrow n_{x+1}}$$

Therefore,

$$\boldsymbol{f}(^{\forall u}P''_u, \hat{P}''_v) = 0,$$

and procedure $\boldsymbol{F}$ excludes all links and switches in NW_c from U_0.

$$U2 = U_1 \wedge \{^{\forall u}\overline{P''_u}\} = U_1 \wedge \overline{NW_c} = \{l^{s_{C1}}_{s_{P1}}\}$$

Thus, all links and switches in both NW_P and NW_C are excluded from U. Only link $l^{s_{C1}}_{s_{P1}}$ is left in U.

Since there is only one link left in U, connecting point t for the unknown host must be that link.

B-2: Switch connecting point Next, we assume that the connecting point of unknown host n_{x+1} is switch s_l. The tree network can be drawn as shown in Fig. 3(c). NW_p, $NW_c, \ldots NW_z$ denote parts of the network. NW_p is located on the root side of the network from s_l, and $NW_c, \ldots NW_z$ is located on the leaf side. First, we consider NW_c as a leaf-side element. The other leaf-side elements can be considered to be the same as NW_c. The elements in this network (Fig. 3(c)) are listed in Table 2.

U_0, the initial value of candidate U is as follows.

$$U_0 = \{s_l, l^{s_{P1}}_{s_l}, l^{s_{C1}}_{s_l}, NW_p, NW_c\}$$

Similar to the previous case, the links and switches in NW_p and NW_c are excluded from U. Given theorem 1, the

Table 2 List of elements in network with unknown host n_{x+1} connected to switch s_l.

s_l	switch that is connecting point t
s_{P1}	switch connected to root side of s_l
s_{C1}	switch connected to leaf side of s_l
NW_c	set of links and switches on leaf side of s_l via $l_{s_l}^{s_{c1}}$

paths from n_1 to every host in NW_p pass through any switch or link in NW_p. On the other hand, the path from the unknown host to any host in NW_c includes no links or switches in NW_p. Therefore, the links and switches in NW_p are excluded from U_0.

$$U_1 = U_0 \wedge \overline{NW_p}$$

Next, we consider the paths from host n_{C1} to every host $n_{Ch}(h = 2, 3, \ldots e)$, shown as P''_u in Fig. 3(c). These paths cover all links and switches in NW_c and never include s_l. If we take another path from n_1 to n_{x+1} as $\hat{P}'_v$, this path does not include NW_c. Therefore, NW_c is also excluded from U.

$$U_2 = U_1 \wedge \overline{NW_c}$$

Similarly, the other leaf side of the network is excluded. For example, NW_d and NW_e are excluded.

$$U_3 = U_2 \wedge \overline{NW_d}, \quad U_4 = U_3 \wedge \overline{NW_e}$$

Finally, we obtain

$$\begin{aligned} U &= (((U_0 \wedge \overline{NW_p}) \wedge \overline{NW_c}) \wedge \overline{NW_d}) \ldots \wedge \overline{NW_z} \\ &= \{s_l, l_{s_l}^{s_{P1}}, l_{s_l}^{s_{C1}}\}. \end{aligned}$$

Thus, only one switch, s_l, is left in U, so connecting point t is uniquely determined.

This discussion proves that our topology detection technique, procedure $\boldsymbol{F}$, can determine the connecting point of an unknown host in a tree network topology.

3.3 Switch Sharing Detection Technique

Next, we explain the details of $\boldsymbol{f}$, our technique for detecting whether two paths share a switch. This technique can test any two paths for which the four edge hosts include a tester host and three other hosts.

First, the MAC address is learned for each switch on one of the two paths, called "path A." Then, an Ethernet frame with the same MAC address as the destination is sent along the other path, "B." If these two paths share a switch, the Ethernet frame will be forwarded along path A. Then, which host received the frame is identified.

In our description here, we consider a network with four hosts: tester host A and hosts B, C, and D. Two paths, A to C and B to D, are tested. We assume that all switches have learned all MAC addresses in the network.

Step 1. Host A sends a unicast address resolution protocol (ARP) request to host C with source address B, as shown by arrow 1 in Fig. 4(a). The address field in the packet headers is [IP dst: C src: B, MAC dst: C, src: B]. The request uses MAC source address B so that switches on this path learn that packets with the MAC address of host B should be forwarded to host A.

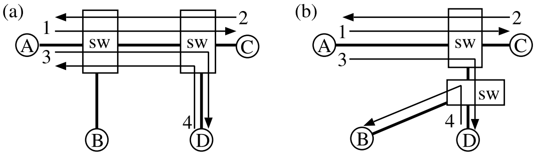


Fig. 4 Example testing configurations: (a) two paths share a switch; (b) two paths do not share a switch.

Step 2. Host C sends an ARP reply to host A, as shown by arrow 2 in Fig. 4(a). The address field in the packet headers is [IP dst:B src:C, MAC dst:B src:C]. Host A needs to be modified in order to receive the ARP reply packets from the other hosts. When host A receives a reply from C, it knows that each switch has learned the MAC address of host B, so packets for host B are forwarded to host A.

Step 3. Host A sends an ICMP ECHO request with source IP address B to host D, as shown by arrow 3 in Fig. 4(a). The address field in the packet headers is [IP dst: D src: B, MAC dst: D src: A].

Step 4. Host D replies to the ICMP ECHO request. The reply packets have an IP and MAC destination of host B. If the path from D to B and that from A to C share a switch, the packets are forwarded to host A, as shown by arrow 4 in Fig. 4(a). Thus, the two paths are determined to share a switch if host A receives the ICMP ECHO reply packets from host D. If the packets are not delivered within a specific timeout period, the two paths are determined not to share a switch, as shown by arrow 4 in Fig. 4(b).

During this procedure, none of the hosts can reach host B because all packets sent to host B are automatically forwarded to host A. To the sending host, it appears that the packets were lost somewhere along the path to B. Therefore, once the testing has been completed, the initial MAC address for host B must be relearned. This can be done, for example, by having tester host A send an ICMP ECHO request packet to host B with an IP source address unused in the network. Host B will then broadcast an ARP request for the unused address because it does not know the MAC address to which it corresponds. This will cause all switches in the network to re-learn the correct MAC address for host B. The SSD procedure can fail if host B sends a packet during the testing because that would initiate MAC address learning. In Sect. 5, we will discuss how background traffic affects the accuracy of our technique and how our technique affects background traffic.

3.4 Reducing Number SSDs

The number of SSDs required by procedure $\boldsymbol{F}$ is an important concern because several tens of hosts are usually connected to a LAN. We thus developed a technique for reducing the number of SSDs required. First, we explain how many SSDs are needed for basic topology detection, which we described above as procedure $\boldsymbol{F}$. Then, we will describe our reduction technique and estimate the number of SSDs required when it is used.

Procedure $\boldsymbol{F}$ repeats SSD for all combinations of two paths for which the edges are a tester host and three other hosts. The number of SSDs is proportional to x^3, where x is the number of hosts.

$$ {}_{x}\mathrm{C}_3 = \frac{x(x-1)(x-2)}{6} = O(x^3) $$

Our reduction technique focuses on one switch at a time and, using SSD, determines toward which link the unknown host is connected. It starts by focusing on the switch nearest the tester host and then recursively determining toward which link the unknown host is connected.

Eventually, these searches are done only for switches on the path from the tester host to the unknown host. The location of the unknown host is determined with fewer SSDs because switches not on the path to the unknown host are not searched.

To test if a link could be in the direction towards the unknown host, the SSD is done using two paths, such as a path passing through the focused switch and a path from the unknown host to a child host connected through the target link. As shown in Fig. 2(a), two paths do not share a switch only when the unknown host is connected through the target link. Thus, if $\boldsymbol{f} = 0$, the link to which the unknown host is connected is known. If $\boldsymbol{f} = 1$, the unknown host is connected directly to the target switch.

Let's look at an example of detecting the location of an unknown host in a network with four known hosts connected to the same switch (Fig. 5(a)). Tester host n_1 attempts to detect the location of unknown host n_5.

Step 1. The switch nearest n_1 is s_1, so it is focused on. The hosts connected to the focused switch via every link are identified. In this example, all hosts are identified because each link from s_1 is connected to only one host.

Step 2. Next, the link through which the unknown host is connected is identified. When testing link $l_{s_1}^{n_1}$, for example, two paths are selected for SSD: 1) from n_2 to n_3, where $P_{n_2 \leftrightarrow n_3} \ni s_1$ and $P_{n_2 \leftrightarrow n_3} \not\ni l_{s_1}^{n_1}$ and 2) from n_1 to n_5, where n_1 is connected through $l_{s_1}^{n_1}$ from s_1.

If these two paths do not share a switch, i.e., $\boldsymbol{f}(P_{n_1 \leftrightarrow n_5}, P_{n_2 \leftrightarrow n_3}) = 0$, $P_{n_1 \leftrightarrow n_5}$ does not contain s_1. Therefore, unknown host n_5 is located towards link $l_{s_1}^{n_1}$, as shown in Fig. 5(b). The case where $\boldsymbol{f} = 0$, i.e., the unknown host is on link $l_{s_1}^{n_2}$, is shown in Fig. 5(c).

This testing continues with other path combinations until the combination of paths giving $\boldsymbol{f} = 0$ is found or all links have been tested with a result of $\boldsymbol{f} = 1$.

Since $\boldsymbol{f} = 1$ for all combinations of paths in this example, unknown host n_5 is not connected through any of the tested links from s_1. Therefore, it must be connected to s_1 via a different link, as shown in Fig. 5(d).

Thus, our SSD technique can detect the link through which an unknown host is connected through the focused switch.

If the next hop node from the selected link in step 2 were a switch, it would be set as the new focused switch and step 2 would be done again. If the next hop node were a host, the link would be considered a connecting point.

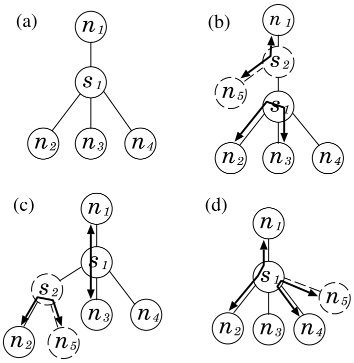


Fig. 5 Example of topology detection: (a) original network, (b) unknown host n_5 connected through $l_{s_1}^{n_1}$, (c) n_5 connected towards $l_{s_1}^{n_2}$, and (d) n_5 connected towards $l_{s_1}^{n_4}$.

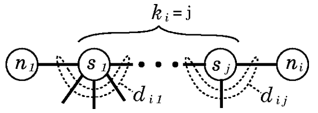


Fig. 6 Hop and degree of switches.

The number of SSDs for detecting the location of an unknown host basically depends on the number of branches on the path from the tester host to the unknown host because each link of each switch on the path from the tester to the unknown host may be tested. Let's consider this number in more detail.

Assume that tester host n_1 is attempting to detect the location of host n_i. Let k_i denote the number of switches on the path from n_1 to n_i, and let d_{ij} denote the degree (number of links) of each switch on the path. The case in which there are j switches on the path from n_1 to $n_i (k_i = j)$ is illustrated in Fig. 6. In the worst case, every link of every switch will be tested in an effort to detect the link towards the unknown host. Therefore, the number of SSDs for adding one host, n_i, is given by

$$ w_i = \sum_{j=1}^{k_i} d_{ij} = \bar{d}_i k_i. $$

Note that $\bar{d}_i$ is the average of d_{i1} to d_{ij}. The total number of SSDs needed for detecting x hosts is given by

$$ W = \sum_{i=1}^{x} w_i = \sum_{i=1}^{x} \bar{d}_i k_i. $$

Here, we introduce $\bar{k}$ as the average hop to all hosts to be detected and $\bar{d}$ as the average degree of switches. That is,

$$ W = x\bar{d}\bar{k}. $$

Note that $\bar{d}$ and $\bar{k}$ are averages for the detected parts of the network only, not the average for the whole network.

The average degree for the whole network is given by $\bar{d}' = z/y = (x + y - 1)/y$, where x is the number of hosts, y is the number of switches, and z is the number of links.

The average hop, $\bar{k}$, is less than the number of switches, $\bar{y}'$, so $\bar{d} \le \bar{d}'$ and $\bar{k} \le \bar{y}' \le y$. This means that the number of SSDs can be expressed as

$$W = x\bar{d}\bar{k} \le x\bar{d}'\bar{y}' = x(x + y - 1)/2.$$

Generally, the number of switches is less than the number of hosts. Here, however, we assume that the number of switches is the same as the number of hosts and that the average degree is half that of the whole network. In this case, the number of SSDs is proportional to the square of the number of hosts.

$$W = \frac{x\bar{d}'\bar{y}'}{2} \le \frac{x(x + x - 1)}{4} = \frac{x(2x - 1)}{4} = O(x^2)$$

4. Simulation Tests

We investigated the number of SSDs required by simulation. We varied the number of hosts and switches and several network settings (tree height, host distribution, etc.).

1. Number of hosts and switches: We first simulated a network in which each switch has the same number of connected hosts. We used a tree topology with a height of four or eight hop switches from the root.

The results for when the height was set to eight are illustrated in Fig. 7(a). The number of SSDs is shown on the z-axis. Those for a height of four are shown in Fig. 7(b). The number of SSDs increased in proportion to the number of hosts. It was higher when there were only a few switches in the network because the degree of switches was huge. For both heights, the number of SSDs was around 1000 to 1500 when there were 100 hosts.

2. Degree of switches and hops from tester: In the next simulation, we varied the average degree of switches and the average number of hops from the tester to hosts in the network. The results for when the degree was constant throughout the network are shown in Fig. 8(a). Those for when the degree was exponentially distributed are shown in Fig. 8(b). The fewer SSDs with the exponential distribution indicate that our technique is efficient for actual networks because the degree of switches is unequal in most networks.

These simulation results demonstrate that our technique can detect the network topology using fewer SSDs than the square of the number of hosts for all cases we tested. For example, 1000 to 2000 SSDs are needed to detect a network topology with 100 hosts.

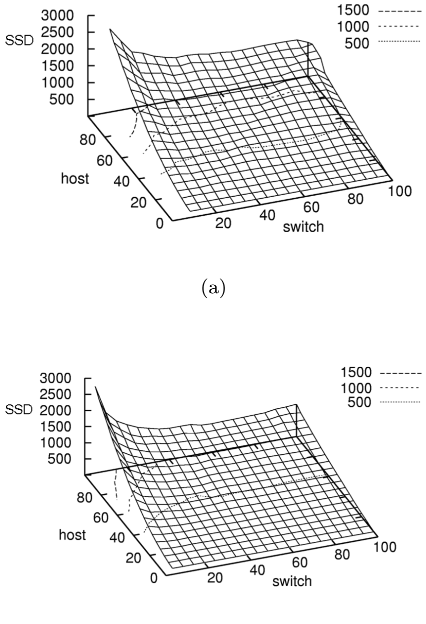
5. Evaluation on Real-World Testbed

To validate our technique in a real-world environment, we constructed a testbed on a PC. We used this testbed to also evaluate the time taken to detect a network topology.

5.1 Real-World Testing

We constructed a network comprising six hosts and two switches, as shown in Fig. 9. Each host was a Linux 2.6 PC connected via 1000Base/T Ethernet. We used Ethernet switches with an integrated Broadcom chipset.

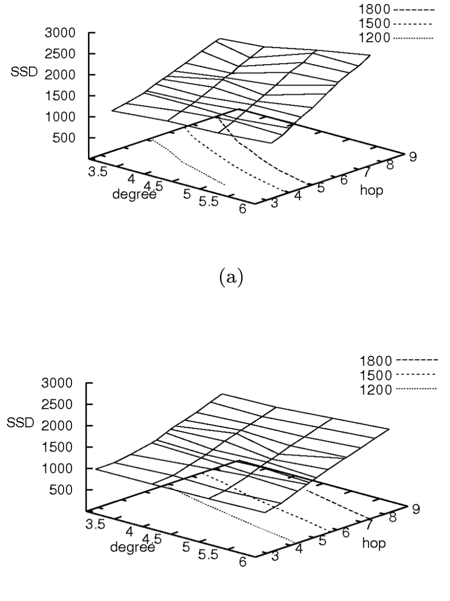
The ICMP echo request was used for multicast addresses (all hosts in the network) to obtain a list of the unknown hosts in the network. SSD was then run to detect the network topology. The SSD was run twice, and the results were accepted only when the same results were obtained for



(a)

(b)

Fig. 7 Number of SSDs for tree network with height of (a) eight and (b) four.



(a)

(b)

Fig. 8 Number of SSDs (a) when degree was constant and (b) when degree was exponentially distributed.

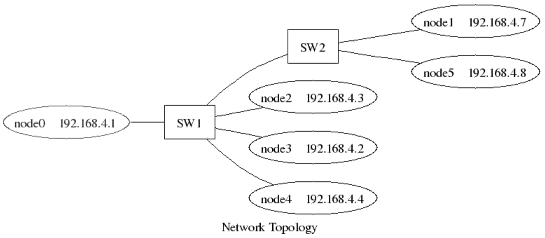


Fig. 9 Network testbed.

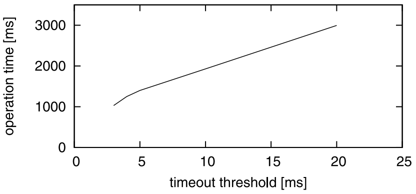


Fig. 10 Detection time vs. timeout setting.

Table 3 Effect of background TCP traffic on accuracy.

Background Traffic	Throughput	Accuracy
TCP Recv (RWIN = 85.3 KB)	780 Mbps	0%
TCP Recv (RWIN = 4 KB)	131 Mbps	80%
TCP Recv (RWIN = 2 KB)	90 Mbps	94%
TCP Send (RWIN = 4 KB)	131 Mbps	96%
TCP Send (RWIN = 2 KB)	90 Mbps	76%
TCP Send (RWIN = 4 KB)	50 Mbps	94%
TCP Recv Send (RWIN = 2 KB)	40 Mbps	74%

both runs as a means to improve accuracy.

The network topology (Fig. 9) was detected correctly after about 220 packets had been transferred through the network.

5.2 Operation Time Evaluation

As shown in Fig. 10, the time taken to detect the network topology correctly strongly depended on the timeout setting with which the tester judged ARP and ICMP packet reception. We measured the round-trip time in a LAN with hundreds of hosts and tens of switches. All of the times were less than three ms, and most were within one ms, indicating that our technique can detect network topology within one second.

5.3 Effect of Background Traffic on Accuracy

Our detection technique might be unable to detect the correct network topology if there is background traffic because it temporarily borrows MAC addresses from hosts in the network. If the original owner of a borrowed MAC address sent a packet, it could overwrite the MAC address acquisition set by the tester, resulting in incorrect detection of the network topology. Since our testbed accepted the SSD results only when the results were the same for two consecutive SSDs, if the same incorrect results were obtained twice in a row, the detection would be incorrect.

When the timeout for SSD was set to 3 ms, it took 15 ms for one SSD. Therefore, our testbed would misdetect the network topology if a host from which the MAC address was borrowed by the tester sent packets at an interval of 15 ms or less. Traffic with a 15 ms packet interval would flow at 800 kbps if full-segment 1500 byte packets were used.

VoIP/UDP, for example, traffic usually comprises packets sent at 50 ms intervals. This means that the probability of the same incorrect results being obtained twice in a row is 9% ($15/50 \times 15/50 = 0.09$). According to the simulation results in Fig. 7 and Fig. 8, 1000 to 2000 SSDs are needed to detect the network topology with 100 hosts. Since our testbed repeat SSD twice, total number of SSDs are 4000. If we simply assume that each MAC address is used for 40 SSDs (total 4000 SSDs/100 hosts = 40) in one topology detection, the possibility when our technique completes topology detection without SSD's retry is 15%($(1 - (15/50))^{40} = 0.152$).

Background traffic generated by TCP cannot be simply modeled because of its complicated behavior in terms of rate control, timeout, etc. Therefore, we evaluated the accuracy of our testbed using TCP background traffic. We generated the traffic using the iperf software tool. It was between a host from which the MAC address was used for SSD and a host that had been eliminated from the list of known hosts.

The results are summarized in Table 3, which shows the type of background TCP traffic, the average throughput of the traffic for 10 seconds, and the accuracy of topology detection. The type of traffic represents the direction of the traffic from the view of the host in the network to be detected. We adjusted these size of the TCP receive window (RWIN) to control the background throughput. Accuracy represents the probability that the topology was detected correctly in 50 times tests.

As shown in the table, the accuracy was sufficiently high when the throughput of the background traffic was around 100 Mbps. When the throughput was 780 Mbps, the accuracy was zero because the host did not respond to the SSD ARP requests due to packet loss. The PCI bus bandwidth in the PC was the bottleneck.

On the other hand, our topology detection may degrade the performance of other traffic flows because it temporarily borrows host MAC addresses. We will discuss case for UDP and TCP respectively.

When background traffic was VoIP/UDP, We assume 40 SSDs/host to detect topology as same as the above discussion. Since we showed that one SSD can be done in 15 ms, our technique will need 60 s (4000 SSDs × 15 ms = 60 s) to detect the whole network. Each host will exprience 30%($15/50 = 0.3$) packet loss for total 0.6 s (40 SSDs/host × 15 ms) during the 60s topology detection.

As for TCP background traffic, due to its rate control and timeout behavior, degradation of TCP throughput may be longer than the topology detecting period. We measured the performance of the TCP traffic without our topology detection. The throughput was 148.8 Mbps (RWIN = 4 KB) and 95.4 Mbps (RWIN = 2 KB), corresponding to degradation of 12%$(1 - 131/148.8 = 0.12)$ and 6%$(1 - 90/95.4 = 0.06)$, respectively. The packet loss rates were supposed to be less than 12% and 6%, respectively.

These tests demonstrate that our topology detection has sufficient accuracy in small network despite the existence of background traffic. They also showed that it does not significantly affect background traffic.

6. Conclusion

We have described three techniques for detecting the topology of Ethernet networks. They can be run on a single host without the assistance of network nodes and other hosts. Simulation showed that they can detect the topology at reasonable cost even for a network with 100 hosts. Testing on a real-world testbed showed that the topology of a network with six hosts and two switches could be detected within 1 second. The accuracy was sufficiently high when the throughput of the background traffic was around 100 Mbps.

Compared to other topology detection techniques, our techniques are easy deployable and quickly operational because they can be run on a single host. They have other potential applications, such as emergency LAN topology detection when an intrusion is detected.

We are planning to test them in LANs with various kinds of switches to further evaluate their performance. Our objective is to improve their accuracy, especially for situations in which there is background traffic.

References

[1] R. Siamwalla, R. Sharma, and S. Keshav, Discovering Internet Topology, Cornell University, 1998.

[2] R. Black, A. Donnelly, and C. Fournet, “Ethernet topology discovery without network assistance,” IEEE International Conference on Network Protocols (ICNP 2004), Oct. 2004.

[3] N. Spring, R. Mahajan, and D. Wetherall, “Measuring ISP Topologies with Rocketfuel,” IEEE SIGCOMM 2002, Aug. 2002.

[4] H. Burch and B. Cheswick, “Mapping the Internet,” Computer, vol.32, no.4, pp.97–98, April 1999.

[5] Y. Breitbart, M. Garofalakis, C. Martin, R. Rastogi, and A. Silberschatz, “Topology discovery in heterogeneous IP networks,” IEEE INFOCOM 2000, March 2000.

[6] M. Son, B. Joo, B. Kim, and J. Lee, “Physical topology discovery for metro Ethernet networks,” ETRI J., vol.27, no.4, pp.355–366, Aug. 2005.

[7] R. Caceres, N.G. Duffield, J. Horowitz, F. Lo Presti, and D. Towsley, “Loss-based inference of multicast network topology,” IEEE Conference on Decision and Control, Dec. 1999.

[8] N.G. Duffield, J. Horowitz, and F.L. Presti, “Adaptive multicast topology inference,” IEEE INFOCOM 2001, April 2001.

[9] ITU-T Recommendation Y.1731, OAM Functions and Mechanisms for Ethernet based Network, 2006.

[10] IEEE Computer Society, “Virtual bridged local area networks — Amendment 5 connectivity fault management,” P802.1ag, Draft, Work in Progress.

[11] “Ethernet service OAM: Overview, applications, deployment, and issues,” Fujitsu Network Communications, 2006.

[12] J. Case, M. Fedor, M. Schoffstall, and J. Davin, “A simple network management protocol (SNMP),” IETF, RFC-1157, May 1990.

[13] A. Bierman and K. Jones, “Physical topology MIB,” IETF, RFC-2922, 2000.

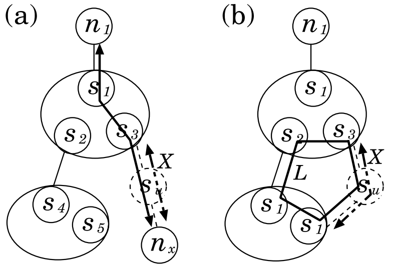


Fig. A· 1 Example for proof of theorem 1: (a) segment edge is connected to a host; (b) both segment edges are connected to the network.

Appendix: Coverage in Tree Network

In a tree network, paths from any one host to the other hosts can pass through any switch or link in the network. Suppose there is a segment X (a link or switch) that is not included in any of the paths. It is clear that one edge of the segment must be connected to a switch in the network given that it is a part of the network. Therefore, we need consider only the edge on the other side. If the other edge is a host, n_x, as shown in Fig. A· 1(a), a path to n_x will include segment X. On the other hand, if both edges are connected to switches, as shown in Fig. A· 1(b), the network has a loop L, which is incompatible with the definition of a tree network. In this case, segment X cannot exist. This means that the paths from any one host in a network to the other hosts cover all switches and links in the network.

Yohei Hasegawa received an M.E. degree from in computer science from Waseda University, Tokyo, Japan, in 1999. He joined NEC Corporation in 1999 and has since been engaged in research on active network architectures, overlay networks, and network measurement. He was a visiting scientist in the Computer Science Department of the University of Massachusetts at Amherst in 2007.

Masahiro Jibiki received a Ph.D. degree in systems management from the University of Tsukuba, Tokyo, Japan. He is currently a researcher in the Central Research Laboratories, NEC Corporation, and, since 2006, a visiting professor at the University of Wakayama, Wakayama, Japan. His research interests include networking, distributed systems, and software science.