IEICE TRANS. INF. & SYST., VOL.E103–D, NO.3 MARCH 2020

PAPER

Identifying Link Layer Home Network Topologies Using HTIP

Yoshiyuki MIHARA[†a)], ***Nonmember***, **Shuichi MIYAZAKI**[††], ***Senior Member***, **Yasuo OKABE**[††], ***Fellow***, **Tetsuya YAMAGUCHI**[†††], ***Member***, ***and*** **Manabu OKAMOTO**[††††], ***Senior Member***

SUMMARY In this article, we propose a method to identify the link layer home network topology, motivated by applications to cost reduction of support centers. If the topology of home networks can be identified automatically and efficiently, it is easier for operators of support centers to identify fault points. We use MAC address forwarding tables (AFTs) which can be collected from network devices. There are a couple of existing methods for identifying a network topology using AFTs, but they are insufficient for our purpose; they are not applicable to some specific network topologies that are typical in home networks. The advantage of our method is that it can handle such topologies. We also implemented these three methods and compared their running times. The result showed that, despite its wide applicability, our method is the fastest among the three.

key words: *home networks, home automation, network topology, graph algorithms, MAC address forwarding tables, HTIP*

1. Introduction

1.1 Background

Electronic devices that provide various kinds of services (called *end devices* in this paper), such as PCs, digital TVs, gaming devices, hard disc recorders, and printers, are connected to home networks, and the number of such devices is still increasing. Also, intermediate devices (or *network devices*), e.g., switches and routers, are connected to home networks using various transmission media, not only conventional UTP (Unshielded Twisted Pair) cables but also PLC (Power Line Communication), wireless, and Coax (Coaxial Cable). These trends are making configurations of home network more and more complex. Figure 1 shows an example of a home network, in which short dashed lines denote communication media other than UTP cables.

The number of IP service troubles are also increasing under these circumstances. There are a lot of possible fault points inside home networks, which include internal failures of either end devices or network devices, and troubles due to incorrect settings by users. A cable may be unplugged somewhere between the troubled end device and the access gateway (AGW), which is the gateway located at the boundary between the access line and the home network. It is hard for a user with no network expertise to detect exactly which point is faulty. This has driven a lot of inquiry calls to support centers.

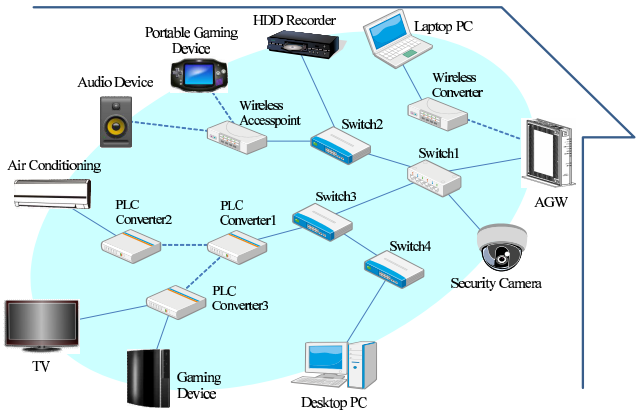

Fig. 1 An example of a home network.

In the current approach to supporting a user over the phone, the operator first asks the user to report the route from the AGW to the troubled end device in order to detect the fault point. However, most users do not know a network configuration. It takes an inordinate amount of time for the operator to grasp the situation in the home network only from the user's verbal descriptions. This increases the cost of service operating companies. Clearly operators need to identify the fault point quickly even from the remote support center in order to reduce the cost.

The link layer network topology is generally used for isolating fault points in networks. The link layer topology includes network devices which are completely transparent from the IP layer. If operators can recognize the link layer network topology of the user's home network, they can identify the network devices on the route between the troubled end device and the AGW. The operators can then detect fault points by checking connectivity to each network device on this route.

In this article, we aim to develop a method to identify the link layer home network topology automatically. We use a MAC address forwarding table (AFT), which is commonly held by network devices. An AFT consists of network de-

Manuscript received June 12, 2019.
Manuscript revised October 4, 2019.
Manuscript publicized December 3, 2019.
[†]The author is with Strategic Business Development Division, NTT, Tokyo, 100–8116 Japan.
[††]The authors are with Academic Center for Computing and Media Studies, Kyoto University, Kyoto-shi, 606–8501 Japan.
[†††]The author is with Research and Development Planning Department, NTT, Tokyo, 100–8116 Japan.
[††††]The author is with Faculty of Computer and Infomation Sciences, Sojo University, Kumamoto-shi, 860–0082 Japan.
a) E-mail: yoshiyuki.mihara.cr@hco.ntt.co.jp
DOI: 10.1587/transinf.2019EDP7161

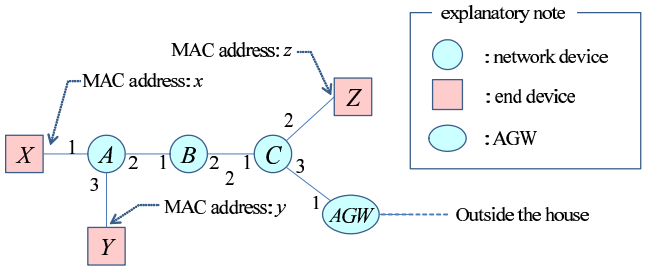

Fig. 2 An example network.

Table 1 AFTs corresponding to the network of Fig. 2.

A		
1	2	3
x	z	y
	$M(B_1)$	
	$M(C_1)$	
	$M(AGW_1)$	

B	
1	2
x	z
y	$M(C_1)$
$M(A_2)$	$M(AGW_1)$

C		
1	2	3
x	z	$M(AGW_1)$
y		
$M(A_2)$		
$M(B_2)$		

AGW
1
x
y
z
$M(A_2)$
$M(B_2)$
$M(C_3)$

vice port numbers and MAC addresses visible from each port. Figure 2 shows an example of a network, where A, B, and C are network devices and X, Y, and Z are end devices. Table 1 shows AFTs of network devices in Fig. 2, where $M(D_p)$ for a network device D denotes the MAC address of port p of device D. For example, the column corresponding to port 1 of device B includes the MAC addresses of X, Y, and port 2 of A. Note that an AFT of the AGW does not include the port information of outside the home network.

Our approach in this paper is to collect all the AFTs to one specific network device, called a *manager*, and the manager computes the network topology. In general, any network device can serve as a manager, but typically the AGW plays its role, so throughout this paper we assume that the AGW is the manager. AFTs are usually collected by the AGW using SNMP [1], which is, however, generally unsupported on network devices in home networks. To compensate this inconvenience, HTIP (Home network Topology Identifying Protocol) [2]–[5] was proposed and standardized, whose main purpose is to collect AFTs from home network devices. Note that there is no difference between the AFTs collected by SNMP and those corrected by HTIP.

1.2 Related Works and Their Problems

When we talk about a network topology, there are mainly two kinds of topologies: the IP layer topology and the link layer topology. There are a lot of commercial tools and related works for identifying IP layer network topologies,

Table 2 AFTs of Fig. 2 including only MAC addresses of end devices.

A		
1	2	3
x	z	y

B	
1	2
x	z
y	

C		
1	2	3
x	z	
y		

AGW
1
x
y
z

Table 3 AFTs of Fig. 2 including only MAC addresses of network devices.

A		
1	2	3
	$M(B_1)$	
	$M(C_1)$	
	$M(AGW_2)$	

B	
1	2
$M(A_2)$	$M(C_1)$
	$M(AGW_1)$

C		
1	2	3
$M(A_2)$		$M(AGW_1)$
$M(B_2)$		

AGW
1
$M(A_2)$
$M(B_2)$
$M(C_3)$

such as HP's Operations Manager[†], IBM's Tivoli NetView[††] and HelpSystems's InterMapper[†††]. Also there have been proposed methods which use ICMP [7] or Traceroute [8]–[12]. However, there have been relatively few works on identifying a link layer topology. We describe here a couple of major works on identifying a link layer topology and their problems.

Bejerano's method [13] uses AFTs that consist of MAC addresses of only end devices (Table 2). Briefly speaking, to determine an edge, this method computes a set inclusion of two columns of AFTs. Let us denote by $D(j)$ the set of MAC addresses contained in the jth column of the AFT of the device D. Consider the example of Fig. 2. We have that $A(1) = \{x\}$ and $B(1) = \{x, y\}$, so we know that $A(1) \subset B(1)$ and there is no set that lies between them. From this, we know that device A is connected to device B, and device B is closer to the AGW than device A. Hence we connect B's 1st port and A's port that is visible from the AGW (i.e., A's 2nd port). A major drawback of this method is that it cannot identify the topology when a network device exists that is connected to only two other network devices, such as device B in Fig. 2. This causes $B(1) = C(1)$ and hence we cannot distinguish between B's 1st port and C's 1st port. Such a device can be seen in home networks since, when extending a home network, users sometimes install an intermediate device such as B (instead of buying long cables). Therefore, this method is applicable for ordinal networks but is not enough for home networks.

Breitbart et al.'s method [14] uses AFTs consisting of MAC addresses of only network devices (Table 3). Briefly speaking, it works as follows. For each pair of AFT columns

[†]http://www8.hp.com/jp/ja/software-solutions/operations-manager-infrastructure-monitoring/
[††]http://www-03.ibm.com/software/products/ja/tivoli-netview-zos
[†††]http://www.helpsystems.com/intermapper

$D(i)$ and $D'(j)$ such that $|D(i)|+|D'(j)|$ is equal to the number of network devices, if $D(i) \cup D'(j)$ contains MAC addresses of all the network devices, then we know that port i of device D and port j of device D' are connected. In the example of Fig. 2 and Table 3, we have that $|A(2)| + |B(1)| = 4$ and $A(2) \cup B(1)$ contains MAC addresses of all of A, B, C, and AGW, so we know that A's 2nd port is connected to B's 1st port. On the other hand, $A(2) \cup C(3)$ does not contain A's MAC address even though $|A(2)| + |C(3)| = 4$, so we conclude that A's 2nd port and C's 3rd port are not connected. This method can identify network topologies for which Bejerano's method [13] fails. However, it assumes SNMP for collecting AFTs since their target is not home networks but usual computer networks, and this enables the AGW to know MAC addresses that are invisible from it. For example, consider Fig. 2 again. If the AFT of device B is sent to the AGW using SNMP, then the AGW can know $M(B_1)$, which we cannot expect if HTIP is used. This ability makes it possible for Breitbart et al.'s method to check if the union of two sets includes MAC addresses of all the network devices. If all the MAC addresses of each device are the same, we can still use Breitbart et al.'s method even if HTIP is used, because $M(B_1) = M(B_2)$ and the AGW can know $M(B_2)$. However, this condition is too strong and hence we need a new method that works without this assumption.

1.3 Our Contributions

We propose a method that resolves both problems mentioned in Sect. 1.2, that is, our method can identify networks for which two existing methods [13], [14] fail. Our method uses AFTs consisting of MAC addresses of only network devices, just as Breitbart et al.'s method [14] does, and identifies a leaf node one by one.

We also conducted experiments to compare computation times of our method with Bejerano [13] and Breitbart et al. [14] methods. To this end, we generated networks for which these two existing methods can handle, and measured computation times of three methods. The result showed that, despite its wide applicability, our method is the fastest among the three.

We have presented a preliminary version of this work at the conference IEEE CCNC 2016 [15]. In the current paper, we added an example of an execution of our algorithm (Sect. 5.2), detailed analysis of the running times of algorithms (Sect. 6), and more experiments (Sects. 7.2 and 7.3).

1.4 Structure of This Article

The structure of this article is as follows. In Sect. 2, we show preliminaries. In Sect. 3, we describe an overview of our method. In Sect. 4, we formalize the *Skeleton Tree Construction problem*, which is the main part of network topology identification, as a graph problem. In Sect. 5, we propose an algorithm for solving the Skeleton Tree Construction Problem. In Sect. 6, we analyze time-complexities of the two existing methods and our method. In Sect. 7, we give experimental results. In Sect. 8, we give brief explanations on supplementary processes used in our method. Finally, we describe a brief summary in Sect. 9, including a future work.

2. Preliminaries

2.1 Graph Representations of Networks

A network can be represented as a graph where each device is a vertex (or a node) and transmission media is an edge. Figure 3 is a graph representation of the network of Fig. 1. End device nodes are represented by dashed circles, where a label inside a circle represents a MAC address of the corresponding device. For convenience, we assume that these labels denote the names of nodes as well. For example, the node f in Fig. 3 corresponds to the TV in Fig. 1, and the MAC address of the TV is f. Network device nodes are represented by solid circles. Names of network devices are abbreviated, such as WR for Wireless Converter, $SW2$ for Switch2, and $PLC1$ for PLC Converter1. Note that in this graph representation, network devices and end devices correspond to internal nodes and leaf nodes, respectively. Each label on the port represents a port number and its MAC address; for example, "2/003" of $SW3$ means that this is the 2nd port of $SW3$ and its MAC address is "003".

2.2 Networks under Consideration

Home networks considered in this paper are within the link layer broadcast domain. We assume that there is no loop for the redundant configuration, and also that there is no virtual LAN.

We allow networks to have a device which has a point-to-multipoint interface, such as wireless and PLC. This interface can connect one port of the device to more than one

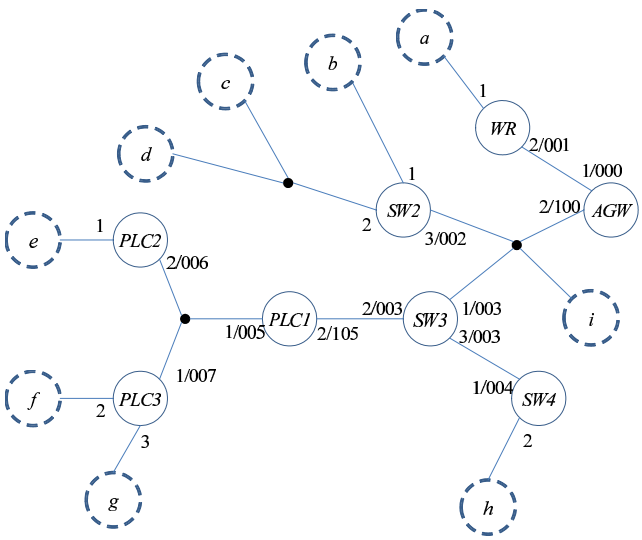

Fig. 3 A graph corresponding to Fig. 1.

Table 4 AFTs corresponding to Fig. 3.

WR (001)	
1	2
a	000
	002
	003
	004
	105
	006
	007
	b
	c
	d
	e
	f
	g
	h
	i

AGW	
1	2
001	002
a	003
	004
	105
	006
	007
	b
	c
	d
	e
	f
	g
	h
	i

*SW*2 (002)		
1	2	3
b	*c*	100
	d	001
		003
		004
		105
		006
		007
		a
		e
		f
		g
		h
		i

*SW*3 (003)		
1	2	3
100	105	004
001	006	*h*
002	007	
a	*e*	
b	*f*	
c	*g*	
d		
i		

*SW*4 (004)	
1	2
100	*h*
001	
002	
003	
105	
006	
007	
a	
b	
c	
d	
e	
f	
g	
i	

*PLC*1 (105)	
1	2
006	100
007	001
e	002
f	003
g	004
	a
	b
	c
	d
	h
	i

*PLC*2 (006)	
1	2
e	100
	001
	002
	003
	004
	005
	007
	a
	b
	c
	d
	f
	g
	h
	i

*PLC*3 (007)		
1	2	3
100	*f*	*g*
001		
002		
003		
004		
005		
006		
a		
b		
c		
d		
e		
h		
i		

devices. Therefore, more than two devices may be connected by a link, which can be seen as a hyperedge in the graph representation. In Fig. 3, the point-to-multipoint interface connecting *PLC*1, *PLC*2 and *PLC*3 is represented by a hyperedge (a black small circle).

We also allow networks to include an *uncooperative network device* which does not support HTIP. Such a device cannot be recognized by other devices since it does not send any packets/frames, and hence corresponds to a hyperedge in the graph representation. To see this, consider Fig. 1 again, where Switch1 is an uncooperative network device. It is connected to four (network and end) devices, and hence it must correspond to a vertex of degree four. But since it is not seen from other devices, each of the interfaces of four devices can see each other, as if they are connected directly. Thus Switch1 should correspond to a hyperedge consisting of four nodes. In the same example, Wireless access point is also an uncooperative device, which corresponds to another hyperedge in Fig. 3.

Table 5 Information collected by AGW.

Device Name (Chassis ID)	Port Number	The Source MAC Address of LLDP Frame	Interface Type (IANAifType number)	AFT
WR	2	001	wireless(71)	see Table 4
*SW*2	3	002	UTP(6)	
*SW*3	1	003	UTP(6)	
*SW*4	1	004	UTP(6)	
*PLC*1	2	105	UTP(6)	
*PLC*2	2	006	PLC(174)	
*PLC*3	1	007	PLC(174)	

2.3 MAC Address Forwarding Tables

Table 4 shows AFTs of the network devices in Fig. 3. One AFT corresponds to one network device. Each column of an AFT corresponds to each port of the corresponding device, and each column includes MAC addresses of visible ports of other devices from that port in an arbitrary order. For example, an AFT of *PLC*1 in Fig. 3 has two columns since it has two ports. Notice from Fig. 3 that the 2nd port of *PLC*2, the 1st port of *PLC*3, and three end devices *e*, *f*, and *g* are visible from the 1st port of *PLC*1. Therefore, the 1st column of an AFT of *PLC*1 includes MAC addresses of these five devices. Similarly, the column corresponding to the 2nd port of *PLC*1 includes eleven MAC addresses seen from it.

There is a designated network device called *AGW*, which collects AFTs of all the cooperative network devices. More concretely, an AFT is encapsulated in an LLDP frame and transmitted from a network device to the AGW. Here we remark that information other than AFTs, such as Chassis ID and IANAifType [16] number, are sent to the AGW using LLDP (See Table 5). "The Source MAC Address of LLDP Frame" in the third column is the MAC address visible from the AGW side. This is the *representative MAC address* written under the device name using parentheses in Table 4. Note that the AGW has no representative MAC address.

3. Overview of the Topology Identification Flow

Recall that our goal is to identify the whole network topology as depicted in Fig. 3. However, end devices are relatively easy to deal with because they are located on leaves of the tree. The difficult part is identification of the backbone consisting of network devices. Therefore, we treat network devices and end devices separately. We call a tree obtained by removing all the nodes corresponding to end devices a *skeleton tree* (this term is originally defined in [13]). Figure 4 shows the skeleton tree of the network in Fig. 3, and Table 6 is the AFTs corresponding to it. Note that Table 6 can be obtained from Table 4 by removing MAC addresses of all the end devices.

Our approach is illustrated in Fig. 5. Recall that all the AFTs (Table 4) are aggregated to the AGW. From this, we

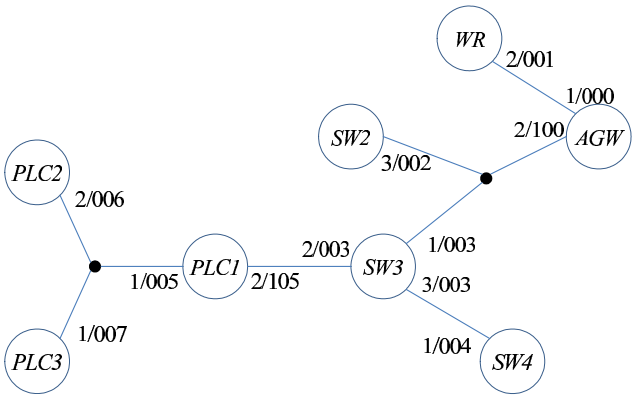

Fig. 4 A skeleton tree of Fig. 3.

Table 6 AFTs of Fig. 3 including only MAC addresses of network devices.

WR (001)
2
000
002
003
004
105
006
007

AGW	
1	2
001	002
	003
	004
	105
	006
	007

*SW*2 (002)
3
100
001
003
004
105
006
007

*SW*3 (003)		
1	2	3
100	105	004
001	006	
002	007	

*SW*4 (004)
1
100
001
002
003
105
006
007

*PLC*1 (105)	
1	2
006	100
007	001
	002
	003
	004

*PLC*2 (006)
2
100
001
002
003
004
005
007

*PLC*3 (007)
1
100
001
002
003
004
005
006

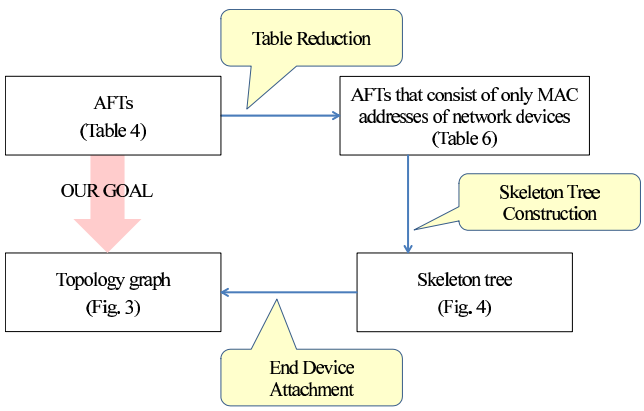

Fig. 5 A diagram of identifying network topology.

construct the AFTs corresponding to the skeleton tree (Table 6) by removing all the MAC addresses of end devices. We call this process the *Table Reduction*. Next, we construct the skeleton tree (Fig. 4) from the reduced AFTs (Table 6). This process is the main part of this work and is called the *Skeleton Tree Construction*. Finally, we construct the whole home network topology (Fig. 3) by connecting the end device nodes to the skeleton tree (Fig. 4). We call this process the *End Device Attachment*.

In the subsequent two sections, we formalize the Skeleton Tree Construction problem and give an algorithm to solve it. As mentioned above, the Table Reduction and the End Device Attachment are easy task, so we explain them briefly in Sect. 8 after the algorithm.

4. Formalization of the Skeleton Tree Construction Problem

In this section, we formalize the *Skeleton Tree Construction problem*. Let $G = (V, E)$ be an undirected graph, where V and E are the sets of vertices and edges, respectively. As mentioned previously, an underlying network is connected and does not have a loop, so in this paper G is always a tree. Also, recall from Sect. 2.2 that E may contain a hyperedge. There is a designated node $r \in V$, called the *manager node*, which corresponding to the AGW.

Let $d(v)$ denote the degree of a vertex v. For each vertex v, numbers 1 through $d(v)$ are associated with the edges incident to v without duplication. For an edge $e = (u, v)$, if e's number (with respect to v) is i then we say that e is the ith edge of v. We call the connecting point of v and e the *ith port of* v and denote it $port(v, i)$. Associated with each $port(v, i)$ is a string called the *label* and denoted $label(v, i)$, which abstracts the MAC address of this port. Since we assume that one device may have different MAC addresses for different ports, $label(v, i)$ and $label(v, j)$ may be different. However, different devices cannot have the same MAC address, so we have that $label(u, i) \neq label(v, j)$ if $u \neq v$.

For example, the skeleton tree in Fig. 4 can be represented as the graph with

$$V = \{AGW, WR, SW2, SW3, SW4, PLC1, PLC2, PLC3\}$$

and

$$E = \{(WR, AGW), (AGW, SW2, SW3), (SW3, SW4), (SW3, PLC1), (PLC1, PLC2, PLC3)\}.$$

Edges $(AGW, SW2, SW3)$ and $(PLC1, PLC2, PLC3)$ are hyperedges consisting of three vertices. $PLC1$ has two ports and their associated labels are 005 and 105. Hence $label(PLC1, 1) = 005$ and $label(PLC1, 2) = 105$ in our notation.

Let $N(v, i)$ be the set of vertices that are reachable from v via $port(v, i)$. For example, $N(PLC1, 1) = \{PLC2, PLC3\}$ and $N(PLC1, 2) = \{WR, AGW, SW2, SW3, SW4\}$. If $u \in N(v, i)$ then we say that the *visible port* of v from u is $port(v, i)$. For example, the visible port of $PLC1$ from AGW is $port(PLC1, 2)$. We may also say that $label(v, i)$ is *visible* from u.

For each vertex v, we associate AFT_v (which corresponds to an AFT of v). The first row of AFT_v consists of the name of the corresponding vertex v and its representative label (which is an abstraction of the representative MAC address explained in Sect. 2.3). AFT_v has $d(v)$ columns, each of which corresponds to each port of v. For each v, i, and $u \in N(v, i)$, the label of the visible port of u from v is stored in the ith column (i.e., the column corresponding to v's ith port) of AFT_v in an arbitrary order. We denote $AFT_v(i)$ the ith column of AFT_v. For example, AFT_{PLC1} in Table 6 has two columns since the degree of $PLC1$ is two. The 2nd column of AFT_{PLC1}, namely $AFT_{PLC1}(2)$, contains labels 100, 001, 002, 003 and 004 because these are the labels of $N(PLC1, 2) = \{AGW, WR, SW2, SW3, SW4\}$ visible from $PLC1$.

Finally, we let AFT_G be the set of AFT_v for all the vertices v in G. The Skeleton Tree Construction problem is the problem of constructing G from given AFT_G.

5. Algorithm for the Skeleton Tree Construction Problem

In this section, we present an algorithm for the Skeleton Tree Construction problem. It is easy to see that, by examining AFT_G, we can recognize the number of vertices and the name, the degree, and the representative label of each vertex. Hence we first prepare isolated vertices and ports. The task of the algorithm is to verify, using AFT_G, how the vertices are connected by edges.

5.1 Algorithm for the Skeleton Tree Construction Problem

The pseudo-code of our algorithm is shown in Algorithm 1. Here, ML (standing for Manager Labels) at Line 2 denotes the set of the labels of all the ports of the manager node. We assume that the manager node knows its own labels. Basically, our algorithm determines edges in an order from leaves to the root (where we consider the manager node as the root). To do so, Algorithm 1 removes at Lines 1 through 3 all the entries of the jth column of AFT_v, if v's visible port from the root is $port(v, j)$. This makes it easier to identify leaves.

The **while**-loop of Lines 4 through 12 is the main loop of Algorithm 1. Suppose that the correct solution is $G = (V, E)$ (which is of course unknown to the algorithm). For each $i \geq 0$, let $G_i = (V, E_i)$ be the graph that Algorithm 1 holds just after the ith loop. As mentioned before, the initial graph consists of only isolated vertices and hence $E_0 = \emptyset$. For a better exposition, we define $\tilde{G}_i = (V, \tilde{E}_i)$ where $\tilde{E}_i = E \setminus E_i$. This is the graph consisting of only edges that are not yet found by the algorithm (which is, again, unknown to the algorithm). Each vertex has one of two states; *marked* and *unmarked*. At the beginning, all the vertices are unmarked.

Before the execution of the ith loop, we have a provisional solution G_{i-1} and implicitly $\tilde{G}_{i-1}$. During the ith loop, Algorithm 1 identifies all the leaf nodes of $\tilde{G}_{i-1}$ and connect these leaves to their parents. As a result, we obtain an updated graph G_i. Finally, Algorithm 1 modifies AFTs so that the current AFT_G corresponds to $\tilde{G}_i$, which allows Algorithm 1 to identify all the leaves of $\tilde{G}_i$ at the next loop.

Algorithm 1 for the Skeleton Tree Construction Problem

```
1: for each v and j (1 ≤ j ≤ d(v)) do
2:     Remove all the entries of AFT_v(j) if AFT_v(j) contains a label in ML.
3: end for
4: while there is an unmarked vertex except for the manager node do
5:     Let L := the set of vertices v which are unmarked and every column
       of AFT_v is empty.
6:     for each AFT_v(j) containing only the representative labels of ver-
       tices in L do
7:         Let u_1, u_2, ..., u_ℓ be vertices whose representative labels are con-
           tained in AFT_v(j).
8:         Add a (hyper)edge (v, u_1, u_2, ..., u_ℓ).
9:         Mark u_1, u_2, ..., u_ℓ.
10:        Remove the representative labels of u_1, u_2, ..., u_ℓ.
11:    end for
12: end while
```

We explain the ith loop in more detail. When we simply say that a port is *visible* or *invisible*, it means that it is visible or invisible from the root. Consider a vertex v and its invisible port $port(v, j)$. We first show that $AFT_v(j)$ is empty if and only if $port(v, j)$ has no edge in $\tilde{G}_{i-1}$. We show this by induction on i. First consider the base case of $i = 1$. We need to show the following two statements: (1) If $port(v, j)$ has an edge in $\tilde{G}_0$, then $AFT_v(j)$ is nonempty. (2) If $port(v, j)$ has no edge in $\tilde{G}_0$, then $AFT_v(j)$ is empty. Note that since $\tilde{G}_0 = G$, all the ports have an incident edge in $\tilde{G}_0$. Hence the statement (2) is trivially satisfied and so we only need to show the statement (1). Each invisible port contains some labels at the beginning. Since invisible ports do not contain a label in ML, it is not processed at Lines 1 through 3. Therefore those AFTs are nonempty. Thus our claim holds at $i = 1$.

Next, we consider induction step. Suppose that our claim holds up to $i = k$. For $i = k + 1$, let $\tilde{G}_k$ be the graph updated from $\tilde{G}_{k-1}$ by executing the **for**-loop from Lines 6 to 11. For a better exposition, consider an execution of the loop for a single $AFT_v(j)$. Since $AFT_v(j)$ contains the representative labels of $u_1, u_2, \ldots, u_\ell$, $port(v, j)$ has an edge by the induction hypothesis. At Line 8, the hyperedge $(v, u_1, u_2, \ldots, u_\ell)$ is added to G_{k-1}, meaning that $(v, u_1, u_2, \ldots, u_\ell)$ is removed from $\tilde{G}_{k-1}$. At Line 10, the representative labels of $u_1, u_2, \ldots, u_\ell$ are removed from $AFT_v(j)$, so $AFT_v(j)$ becomes empty. Hence the claim holds for v's jth port. It can happen that the representative labels of $u_1, u_2, \ldots, u_\ell$ are removed from other columns of AFTs, but such columns do not become empty in this loop since they must contain the representative label of v. By doing the same argument for all the $AFT_v(j)$ processed in this ith loop, we can see that our claim holds for $i = k + 1$.

The above claim implies that the set L constructed at Line 5 coincides with the set of leaves of $\tilde{G}_{i-1}$. To see this, let $v \in L$. Then all the columns of AFT_v are empty. By the claim, all invisible ports of v have no edge in $\tilde{G}_{i-1}$ and hence v is a leaf of $\tilde{G}_{i-1}$. On the contrary, suppose that v is a leaf

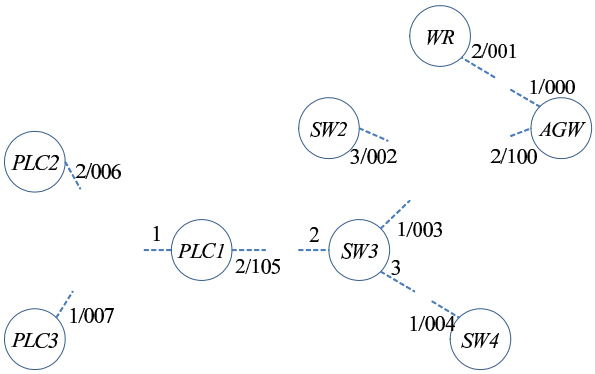

Fig. 6 A graph G_0.

of $\tilde{G}_{i-1}$. This implies that all the invisible ports of v have no edge in $\tilde{G}_{i-1}$. Then by the claim, all the columns of AFT_v corresponding to invisible ports are empty. The column of AFT_v corresponding to the visible port is empty due to the operation of Lines 1 through 3. Therefore, all the columns of AFT_v are empty and hence $v \in L$.

In the **for**-loop from Lines 6 to 11, Algorithm 1 identifies a vertex v which is a parent of a leaf in L. If $AFT_v(j)$ contains only representative labels of vertices in L, we know that $port(v, j)$ is connected to only leaves. At Line 7, Algorithm 1 identifies leaves $u_1, u_2, \ldots, u_\ell$ connected to $port(v, j)$, and at Line 8, it adds an edge connecting v and these leaves. Note that if $\ell = 1$ this is a usual edge and if $\ell \geq 2$ it is a hyperedge. At Line 9, Algorithm 1 marks $u_1, u_2, \ldots, u_\ell$, whose purpose is just to make sure that these vertices are not selected again in later rounds. The role of Line 10 is already explained above. We remark that not all vertices in L are marked at Line 9.

We also note that the manager node does not become a leaf of $\tilde{G}_i$ until the end: Note that the AFT of the manager node does not contain labels in ML and hence it is not processed at Line 2. Therefore its AFT contains information of all the vertices at the beginning, and hence never becomes empty until all other vertices are marked.

5.2 An Example of the Execution of Algorithm 1

We show an example of the execution of Algorithm 1, namely, we show how the graph of Fig. 4 is constructed from AFTs of Table 6. At the beginning, we have a graph G_0 consisting of isolated vertices (Fig. 6). For a better exposition, we place vertices in their right positions (although we do not know the solution at this moment). For each vertex, we already know its degree, name, and representative label. Since the labels of the manager node (AGW) are 000 and 100, we have that $ML = \{000, 100\}$. By executing Lines 1 through 3, Algorithm 1 removes all the columns of AFTs of Table 6 that contain 000 or 001. As a result, we have Table 7.

We now proceed to the main loop (Lines 4 through 12). From Table 7, we can calculate $L = \{WR, SW2, SW4, PLC2, PLC3\}$ and also know that their representative labels are 001, 002, 004, 006, and 007. Since $AFT_{AGW}(1)$ contains

Table 7 AFTs after Line 3.

WR (001)
2
-
-
-
-
-
-
-
-

AGW	
1	2
001	002
	003
	004
	105
	006
	007

$SW2$ (002)
3
-
-
-
-
-
-
-
-

$SW3$ (003)		
1	2	3
-	105	004
-	006	
-	007	

$SW4$ (004)
1
-
-
-
-
-
-
-
-

$PLC1$ (105)	
1	2
006	-
007	-
	-
	-
	-

$PLC2$ (006)
2
-
-
-
-
-
-
-
-

$PLC3$ (007)
1
-
-
-
-
-
-
-
-

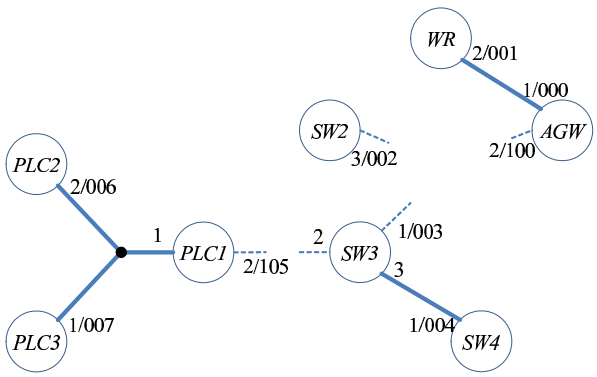

Fig. 7 A graph G_1.

only 001 and it is the representative label of WR, we know that WR is connected to $port(AGW, 1)$ and hence add an edge (AGW, WR). (It is obvious from Fig. 6 that this edge is incident to $port(WR, 2)$.) In a similar manner, we can detect an edge $(SW4, SW3)$. $AFT_{PLC1}(1)$ contains 006 and 007, the representative labels of $PLC2$ and $PLC3$ respectively, so we know that $PLC2$ and $PLC3$ are connected to $PLC1$ by a hyperedge and hence we add $(PLC1, PLC2, PLC3)$. Although $SW2 \in L$, there is no column containing only representative labels of L including 002. Hence we do not mark $SW2$. This happens because $SW2$ is connected with $SW3$ and AGW by a hyperedge, but $SW3$ is not yet a leaf of $\tilde{G}_0$. As a result of this round, we obtain the graph G_1 (Fig. 7). During this round, the representative labels 001, 004, 006, and 007 are removed from AFTs. The resulting AFTs are given in Table 8. (We give asterisk to AFTs of marked vertices.)

In the 2nd round, we have that $L = \{SW2, PLC1\}$. $AFT_{SW3}(2)$ contains only 105, which is the representative label of $PLC1$. Hence we add an edge $(SW3, PLC1)$, mark $PLC1$, and remove 105 from AFTs. Note that SW2 is still

Table 8 AFTs after the 1st round.

WR^* (001)
2
-
-
-
-
-
-
-

AGW	
1	2
-	002
	003
	-
	105
	-
	-

$SW2$ (002)
3
-
-
-
-
-
-
-

$SW3$ (003)		
1	2	3
-	105	-
-	-	
-	-	

$SW4^*$ (004)
1
-
-
-
-
-
-
-

$PLC1$ (105)	
1	2
-	-
-	-
	-
	-
	-

$PLC2^*$ (006)
2
-
-
-
-
-
-
-

$PLC3^*$ (007)
1
-
-
-
-
-
-
-

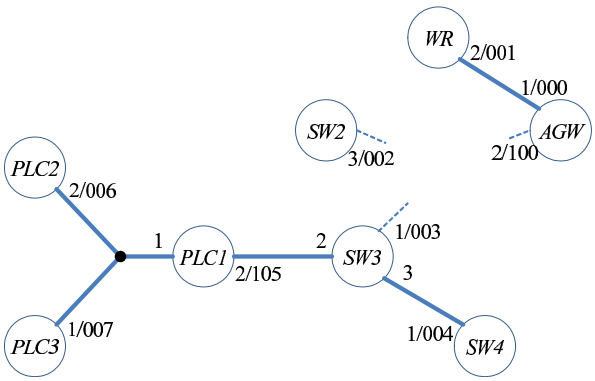

Fig. 8 A graph G_2.

Table 9 AFTs after the 2nd round.

WR^* (001)
2
-
-
-
-
-
-
-

AGW	
1	2
-	002
	003
	-
	-
	-
	-

$SW2$ (002)
3
-
-
-
-
-
-
-

$SW3$ (003)		
1	2	3
-	-	-
-	-	
-	-	

$SW4^*$ (004)
1
-
-
-
-
-
-
-

$PLC1^*$ (105)	
1	2
-	-
-	-
	-
	-
	-

$PLC2^*$ (006)
2
-
-
-
-
-
-
-

$PLC3^*$ (007)
1
-
-
-
-
-
-
-

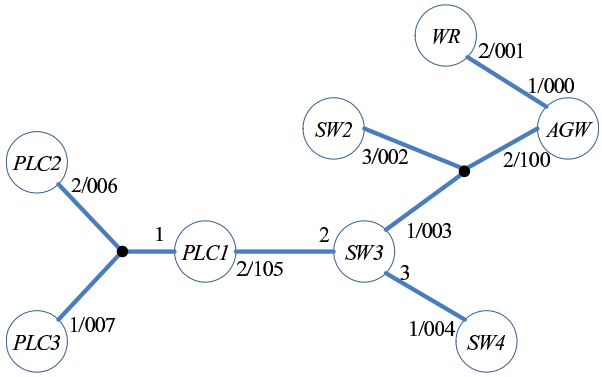

Fig. 9 Output graph.

unmarked after this round. At the end of this round, we obtain the graph G_2 in Fig. 8 and AFTs are modified as in Table 9.

In the 3rd round, we have $L = \{SW2, SW3\}$, and $AFT_{AGW}(2)$ contains the representative labels 002 and 003 of $SW2$ and $SW3$. Hence we add a hyperedge $(AGW, SW2, SW3)$ and mark $SW2$ and $SW3$. Finally, we remove 002 and 003 from AFTs. As a result, all the vertices except for AGW are marked, and hence Algorithm 1 exits the **while**-loop and terminates. The output graph is given in Fig. 9.

6. Analysis of Time-Complexities

In this section, we analyze time-complexites of two existing methods [13], [14] and our method. The analysis of this section is performed for a c-ary tree (where c is a constant such that $c \geq 2$) of n nodes. Then the height of the tree is approximately $h = \log_c n$.

6.1 Time-Complexity of Bejerano's Method

In Bejerano's method, two ports are connected if the numbers of labels they contain differ by exactly one, and one set of labels is included in the other. In the current case, the size of a port (i.e., the number of labels stored in the port) depends only on the depth of the node, so Bejerano's method checks the latter condition (i.e., inclusion) between the nodes of depth i and the nodes of depth $i+1$. The number of nodes of depth i is c^i, and each node has c invisible ports, so there are c^{i+1} ports. Similarly, in depth $i+1$, the number of ports is c^{i+2}. Hence we need to perform $c^{i+1} \times c^{i+2} = c^{2i+3}$ check of inclusion. For checking inclusion, we use *containsAll* method, which takes time proportional to the product of the numbers of elements in two sets. The number of labels each invisible port of depth i node contains is

$$\sum_{j=0}^{h-i-1} c^j = O(c^{h-i-1}),$$

and that of depth $i+1$ node is $O(c^{h-i})$. Therefore, the time for checking inclusion for one pair of ports is $O(c^{h-i-1}) \times$

$O(c^{h-i}) = O(c^{2h-2i-1})$. The number of pairs of ports is c^{2i+3} as calculated above, hence the time-complexity for checking depth i and depth $i+1$ is $c^{2i+3} \times O(c^{2h-2i-1}) = O(c^{2h+2})$. Consequently, the whole time-complexity is

$$\sum_{i=1}^{h-1} O(c^{2h+2}) = O(hc^{2h}) = O(n^2 \log n).$$

6.2 Time-Complexity of Breitbart's Method

In Breitbart's method, if $AFT_v(i) \cup AFT_{v'}(j)$ contains MAC addresses of all the network devices, and $AFT_v(i) \cap AFT_{v'}(j) = \emptyset$, then we connect $port(v, i)$ and $port(v', j)$. This condition is equivalent to $|AFT_v(i)| + |AFT_{v'}(j)| = n$ and $AFT_v(i) \cap AFT_{v'}(j) = \emptyset$. In our implementation, we first sort the ports according to their sizes, and we check the latter condition for port pairs that satisfies the former condition. As in Sect. 6.1, the size of a port depends only on the depth of the node, so a pair satisfying the condition $|AFT_v(i)| + |AFT_{v'}(j)| = n$ is an invisible port of depth i node and a visible port of depth $i+1$ node. The number of nodes of depth i is c^i, and each node has c invisible ports, so there are c^{i+1} such ports. The number of nodes of depth $i+1$ is c^{i+1}, and each node has 1 visible port, so there are c^{i+1} such ports. Hence there are $c^{i+1} \times c^{i+1} = c^{2i+2}$ combinations. We implemented a subroutine for checking for $AFT_v(i) \cap AFT_{v'}(j) = \emptyset$, which can be done in $O(n)$ time, so the whole time-complexity is

$$\sum_{i=0}^{h-1} c^{2i+2} \times O(n) = O(c^{2h}n) = O(n^3).$$

6.3 Time-Complexity of Our Method

In our method, the most costly computation is Step 6, i.e., identifying $AFT_v(j)$ containing only the representative labels of vertices in L. Hence we estimate this time-complexity.

For $i = 1, 2, \ldots, h$, consider an ith iteration of the algorithm. The size of L is c^{h-i+1}. For each node v of depth j, the number of labels contained in v's AFT is

$$\sum_{k=1}^{h-j} c^k \leq c^{h-j+1}.$$

For such node v, we check the inclusion of two sets of sizes at most c^{h-i+1} and c^{h-j+1}. Note that for two sets A and B, checking if $A \subseteq B$ can be done in time $O(|A| + |B|)$ if the elements in A and B are sorted. In our implementation, we used this technique. Therefore, checking this inclusion can be done in $O(c^{h-i+1} + c^{h-j+1})$. The number of nodes of depth j is c^j, so the time-complexity for all the nodes of depth j is $O(c^{h-i+1} + c^{h-j+1}) \times c^j = O(c^{h-i+j+1} + c^{h+1})$.

In the ith iteration, the depth of the nodes we consider is from 0 to $h - i$, so we take the sum of the above time-complexity for these depths.

$$\sum_{j=0}^{h-i} O(c^{h-i+j+1} + c^{h+1}) = O(c^{2h-2i+2} + (h-i+1)c^{h+1}) = O(c^{2h-2i+2}).$$

Finally, by taking the sum for iterations i from 1 to h, we have the whole time-complexity of our algorithm as follow:

$$\sum_{i=1}^{h} O(c^{2h-2i+2}) = O(c^{2h+2}) = O(n^2).$$

7. Experiments

To evaluate the performance of our method, we implemented it and conducted three experiments. The purpose of the first experiment is to compare running times of our method with two existing methods [13], [14]. For this experiment, we generated input trees randomly. However, as mentioned in Sect. 1.2, the existing methods are not applicable to arbitrary networks, so we posed restrictions on inputs that allow all three methods run correctly. The purpose of the second experiment is to see the effect of the restrictions posed at the first experiment. Therefore, we removed the restrictions, ran only our algorithm on them, and compared the running times with the results of the first experiment. The purpose of the last experiment is to investigate the influence of the number of branches on running times. In this experiment, we prepared trees with a fixed number c of branches for several c, and compared the running times of our algorithm for them.

We implemented our method using Java. We were not able to obtain source codes of two existing methods, so we have implemented them by ourselves (also using Java) referring to their papers. For a routine of checking set inclusion, we used *containsAll* method of Java. All the experiments were conducted on a laptop PC with Windows 7, 2.30GHz Intel®Core™i5, and 8.0GB memory.

7.1 Running Times of Three Methods

For the first experiment, we generated inputs in the following way: To construct a skeleton tree, we first determine the number n of nodes and construct the root node r. We then generate an integer $c \in \{2, 3, 4, 5\}$ uniformly at random, and add c children to r. We then repeat the following procedure until the number of nodes reaches n. We choose a leaf node v uniformly at random from the current tree, generate an integer $c \in \{2, 3, 4, 5\}$ uniformly at random, and add c children to v. (In case adding c children makes the number of nodes exceed n, we cut off unnecessary nodes.) We then add end devices (recall that Bejerano's method [13] needs MAC addresses of end devices). To do so, for each leaf node v of the skeleton tree, we generate an integer $c \in \{2, 3, 4, 5\}$ uniformly at random and add c end devices to v. Note that the constructed tree does not have a degree-two internal node, which makes Bejerano's method [13] applicable. Next, we give labels to ports in such a way that each node has the same

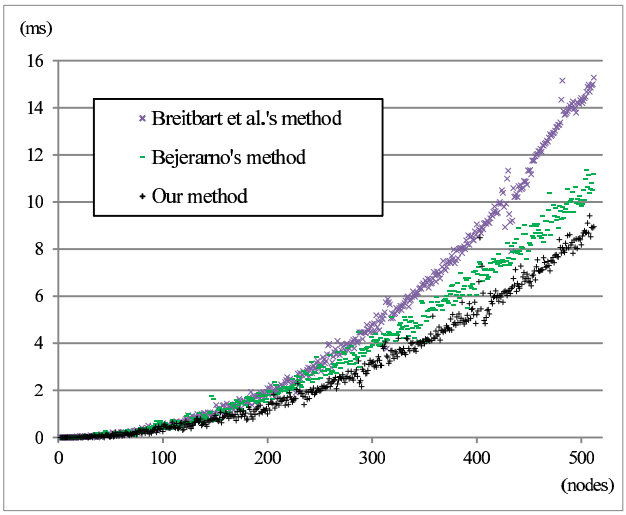

Fig. 10 Running times of three methods.

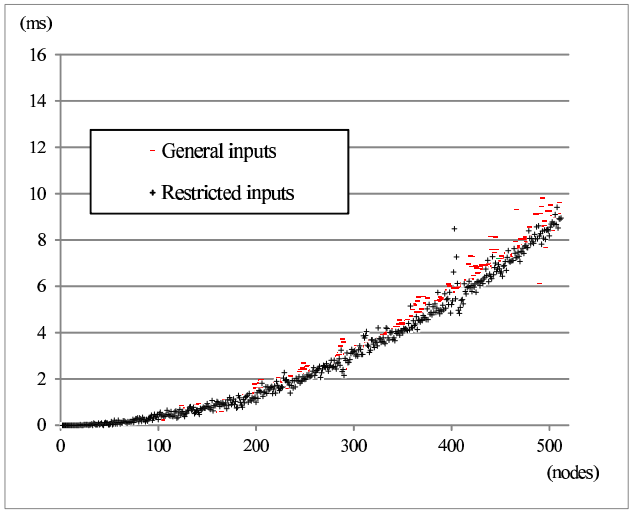

Fig. 11 Running times of our method for general and restricted inputs.

label for its all ports, by which Breitbart et al.'s method [14] is also made applicable. Finally, we construct the set of AFTs corresponding to the constructed tree, which is given to the algorithms as an input.

Figure 10 shows running times of three methods for each n varying from 2 to 512. For each n, we constructed 500 inputs and took the average of running times. It shows that our method is the fastest, followed by Bejerano's and Breitbart et al.'s methods. This order of Fig. 10 matches the analysis of time-complexities in Sect. 6.

7.2 Effect of the Degree-Two Internal Nodes

In this experiment, we removed the restrictions posed on inputs at the first experiment. The restriction to avoid degree-two internal nodes is removed by letting c be chosen from $\{1, 2, 3, 4, 5\}$ instead of $\{2, 3, 4, 5\}$. Note that the second restriction, that is, all the labels of each internal node are the same, does not affect our algorithm because our algorithm uses only one label from each node. Therefore, we did not change the input generation program for this part.

Figure 11 compares running times of our method for two different types of inputs. Results for restricted inputs, plotted in Fig. 11 by the black "+" symbols, are simply taken from the first experiment. The red "-" symbols are results for unrestricted inputs, newly taken this second experiment, which are also average on 500 executions. Running times for restricted inputs are slightly shorter than general ones, but there seems to be no big difference.

7.3 Running Times for Trees with a Fixed Number of Branches

In this last experiment, we fixed the number of branches of c to a constant, taken from $\{1, 2, 3, 4, 5\}$. The number of nodes n is varied from 2 to 512, as the previous experiments. The construction algorithm is also the same as before.

Figure 12 plots the running times of our algorithm,

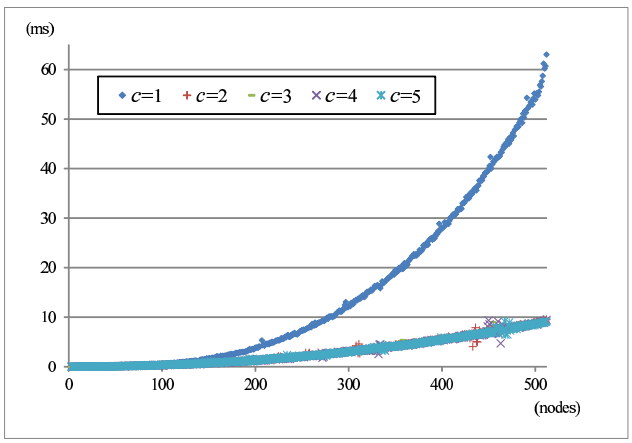

Fig. 12 Running times of our method for each tree with a fixed number of branches.

where each plot is an average of 500 trials. As shown in Fig. 12, running times are fairly large when $c = 1$, compared to the cases of $c = 2, 3, 4, 5$ where almost no differences are observed. This phenomenon can be interpreted as follows. When $c \geq 2$, the time-complexity of the algorithm is $O(n^2)$ as analyzed in Sect. 6. When $c = 1$, however, the previous analysis does not hold since the height of the tree is n. In the **for**-loop in Algorithm 1, the nodes, whose representative labels are $u_1, u_2, \ldots, u_\ell$, are connected with the parent node. Moreover, $u_1, u_2, \ldots, u_\ell$ are removed from each AFTs. When $c \geq 2$, the number of executions of the **for**-loop is approximately $\log_c n$, while when $c = 1$, the number of the execution is n. Consequently, the running time became longer when $c = 1$.

8. Table Reduction and End Device Attachment

In this section, we give brief explanations on how the Table Reduction and the End Device Attachment are performed.

8.1 Table Reduction

Recall that the Table Reduction is a procedure to remove the MAC addresses of all the end devices. We use the ARP scanning to collect them. In the ARP scanning, the AGW sends ARP request packets to all the IP addresses within the same network, except for itself, that can be calculated easily from the network mask of the IP address of the AGW. For example, if the IP address of the AGW is 192.168.11.3/24, the AGW sends ARP request packets to 253 IP addresses from 192.168.11.1 to 192.168.11.254. By receiving responses to these requests, the AGW can collect MAC addresses of all the end devices.

8.2 End Device Attachment

The End Device Attachment is a process to attach end devices to the skeleton tree obtained in the Skeleton Tree Construction. We use a previously used example to show how this process can be performed. Our task is to construct the network of Fig. 3 from Fig. 4 and Table 4. Note that during the Table Reduction we have already collected MAC addresses of all the end devices, so we already know that $a, b, c, d, e, f, g, h, i$ are those MAC addresses.

We consider a column of Table 4 that consists of only MAC addresses of the end devices. We know that only end devices are connected to that port. For example, the 1st column of WR's AFT contains only a, so we know that the device a is connected to the 1st port of WR (recall that we are using the same symbol for both the device name and its MAC address). In the same manner, we can detect the ports which b, h, e, f, and g are connected to. Column 2 of AFT_{SW2} contains two MAC addresses c and d, so we know that there is a hyperedge containing c, d, and $SW2$.

The problematic case is the device i. Note that there is no column that contains only end devices including i. We know that in this case i is connected to one of the hyperedges. To identify this hyperedge, we observe that if i is connected to a hyperedge e, then all the ports faced with e must contain i in its corresponding columns of the AFT. In our current example, consider the hyperedge $(AGW, SW2, SW3)$. This hyperedge is faced with the 2nd, 3rd, and 1st ports of AGW, $SW2$, and $SW3$, respectively. In Table 4, all columns corresponding to these ports contain i, so we know that i is connected to this hyperedge.

9. Conclusion and Future Work

In this paper, we have presented an algorithm to construct a network topology from AFTs, motivated by the importance of detecting user's link layer home network topology in customer services. Our future work is to integrate this algorithm into the existing management system, which is provided by the company three of the authors belong to.

References

[1] J. Case, M. Fedor, M. Schoffstall, and J. Davin, Simple network management protocol (SNMP), RFC 1157, 1990.
[2] Y. Mihara, T. Yamazaki, M. Okamoto, and A. Sato, "Designing HTIP which identifies home network topology and applying HTIP to a troubleshooting application," Information Processing Society of Japan Trans. Consumer device and system, vol.2, no.3, pp.34–45, 2012.
[3] Y. Mihara, T. Yamazaki, and A. Takehiro, "Designing HTIP: home network topology identifying protocol," Proc. IEEE International Conference on Communications, pp.1–6, 2011.
[4] ITU-T G.9973, Protocol for identifying home network topology, 2011.
[5] TTC JJ-300.00, HTIP: Home network topology identifying protocol, 2011.
[6] IEEE Computer Society, 802.1AB-2009: Local and metropolitan area networks - station and media access control connectivity discovery (LLDP), 2009.
[7] J. Postel, Internet control message protocol, RFC 792, 1981.
[8] G. Malkin, Traceroute using an IP option, RFC 1393, 1993.
[9] H. Burch and B. Cheswick, "Mapping the Internet," IEEE Computer, vol.32, no.4, pp.97–98, 1999.
[10] R. Govindan, and H. Tangunarunkit, "Heuristics for Internet map discovery," Proc. IEEE INFOCOM, pp.1371–1380, 2000.
[11] R. Siamwalla, R. Sharma, and S. Keshav, Discovering Internet topology [Online] Available: http://www.cs.cornell.edu/skeshav/papers.html
[12] N. Spring, R. Mahajan, and D. Wetherall, "Mesuring ISP topologies with Rocketfuel," Proc. ACM SIGCOMM, pp.133–146, 2002.
[13] Y. Bejerano, "Taking the skeletons out of the closets: a simple and efficient topology discovery scheme for large Ethernet LANs," IEEE/ACM Transactions on Networking, vol.17, no.5, pp.1385–1398, 2009.
[14] 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, pp.401–414, 2004.
[15] Y. Mihara, S. Miyazaki, Y. Okabe, T. Yamaguchi, and M. Okamoto, "Identifying link layer home network topologies using HTIP," Proc. IEEE CCNC, pp.891–898, 2017.
[16] Internet Assigned Numbers Authority (IANA), IANAifType-MIB definitions, ver. 201409240000Z 2014.
[17] ISO/IEC 29341-1, Information technology - UPnP Device Architecture - Part 1: UPnP Device Architecture Version 1.0, Edition 1.0, 2008.
[18] UPnP Forum, Internet Gateway Device (IGD) V 1.0, 2001.
[19] UPnP Forum, MediaServer:1 and MediaRenderer:1, 2002.
[20] UPnP Forum, Device Management:1, 2010.
[21] IEEE Computer Society, 802.3-2008, Part 3: Carrier sense multiple access with Collision Detection (CSMA/ CD) Access Method and Physical Layer Specifications, 2008.

Yoshiyuki Mihara is a Manager, NTT Strategic Business Development Division. He received the B.Sc. and M.Sc. degrees from Tokyo Institute of Technology in 2004 and 2006, respectively. He also received the Ph.D. from Kyoto university in 2017. Since joining NTT in 2006, he has been engaged in R&D of a home network management service. He has achieved standardization of the home network management protocols that he designed in UPnP, ITU-T, and Japan's TTC. He is currently promoting the key protocols with a view to launching a home network management service.

Shuichi Miyazaki is an associate professor at Academic Center for Computing and Media Studies, Kyoto University. He received B.E., M.E., and Ph.D. degrees from Kyushu University in 1993, 1995 and 1998, respectively. His research interests include discrete algorithms and computational complexity theory.

Yasuo Okabe received M.E. from Department of Information Science, Kyoto University in 1988. From 1988 he was an Instructor of Faculty of Engineering, from 1994 he was an Associate Professor of Data Processing Center, and from 1998 he was an Associate Professor of Graduate School of Informatics, Kyoto University. He is now a Professor of Academic Center for Computing and Media Studies, Kyoto University. Ph.D. in Engineering. His current research interest includes Internet architecture, network security and distributed algorithms. He is a member of IPSJ, ISCIE, JSSST, IEEE, and ACM.

Tetsuya Yamaguchi is a Senior Manager, R&D Produce Group, NTT Research and Development Planning Department. He received his B.E., M.E., and Ph.D. degrees in information engineering from Osaka University, Osaka, in 1997, 1999, and 2008, respectively. He joined NTT in 1999 and engaged in R&D of content distribution, navigation for IPTV services, home network services and ultra-realistic communications. Since moving to NTT Research and Development Planning Department in 2018, he has been engaged in promoting advanced media services.

Manabu Okamoto received the M. Design degree from Kyushu Institute of Design, Japan, in 1991 and Doctor of Design from Kyushu University, Japan, in 2007. In 1991 he joined NTT Electrical Communication Laboratories and researched the acoustic design of various kinds of teleconferencing systems and the system design of ICT services. He moved to Sojo University as a Professor in 2019. He is a senior member of the Institute of Electronics, Information and Communication Engineers of Japan, and a member of the Acoustical Society of Japan and IEEE.