Regular Paper

The Equal Deepest Vertex First Reboot: Rebooting Network Edge Switches in a Campus Network

Motoyuki Ohmori[1,a) Koji Okamura[2,b)

Received: June 17, 2019, Accepted: November 29, 2019

Abstract: Even in the era of Software Defined Network (SDN) or Software Defined Infrastructure (SDI), network edge switches still require to be rebooted for some reasons, e.g., updating a firmware, configuring a special behavior and so on. It may be necessary to clarify how one can shorten downtime of a campus network when many switches in the network require reboots. To this end, this paper proposes the *equal deepest vertex first reboot with vertex contraction* that can simultaneously reboot many network switches with less overhead downtime. This paper tries to express a campus network in a graph theory fashion, reduce downtime overhead by *vertex contraction*, and proves that all switches can be rebooted within a finite number of rebooting procedures. This paper presents an implementation of the proposed procedures, and evaluates the proposed method in an actual campus network. The *equal deepest vertex first reboot with vertex contraction* has appeared to reboot all switches by only 16-second additional overhead out of 109-second downtime in total where the ideal minimum downtime was 93 seconds in an actual campus network where there were more than 300 network switches installed.

Keywords: Network switch, availability, firmware update, certificate replacement and maintenance reboot.

1. Introduction

Even in the era of Software Defined Network (SDN) or Software Defined Infrastructure (SDI), network edge switches require to be rebooted in some situations, e.g., updating a firmware, configuring a special behavior and so on. We, Tottori University, replaced almost all network switches in our campus network with newer network switches made by AlaxalA in September 2017. We are now facing a serious problem that an edge switch requires a reboot to activate a new digital certificate for web authentication after installing the certificate. As a digital certificate expires within few years or even few months, we have to periodically install a new digital certificate into all edge switches, and reboot them. When an edge switch is rebooted, an end host becomes disconnected, and this downtime should be minimized.

In order to minimize this sort of downtime, the order and timing to reboot an edge switch is important. Let us take an example. In our campus network, some edge switches are *upstream* switches that accommodate another *downstream* edge switch. If we reboot an *upstream* edge switch first, a *downstream* edge switch is inaccessible while the *upstream* edge switch is rebooting. The *downstream* edge switch should be then rebooted after the *upstream* edge switch reboots and becomes up again. An end host may be then disconnected longer in total. On the other hand, we can shorten this downtime if we reboot a *downstream* edge switch first. While the *downstream* edge switch is rebooting, the *upstream* edge switch is still accessible. The *upstream* edge switch can be, therefore, rebooted right after the *downstream* edge switch is rebooted before the *downstream* edge switch becomes up. The order and timing to reboot an edge switch may be, thus, important. We have actually experienced that we could not access to a downstream edge switch when we rebooted upstream and downstream edge switches at the same time. The better order and timing may not be, however, clarified enough yet.

In order to dig into the better order and timing to reboot an edge switch, this paper proposes the *equal deepest vertex first reboot with vertex contraction* to reboot edge switches. This method represents a campus network as an unweighted directed rooted spanning tree graph, and contracts vertices of *non-reboot switches* from a graph. This method then obtains a set of vertices of edge switches that are in the same depth from a root vertex. This method reboots edge switches in the *deepest vertex first* fashion, i.e., rebooting the farthest edge switches from a core switch. This method simultaneously reboots edge switches in the same depth, and then can avoid unnecessary waiting time for a switch reboot. This method can minimize the total *downtime* of a network.

Note that this paper focuses on the case where almost all of edge switches require reboots but *non-reboot switches* such as core, distribution and some edge switches do not. It is out of the scope of this paper to reboot core and distribution switches that may require a longer time to reboot. Also note that this paper is an extended version of our preliminary papers [1], [2].

The rest of this paper is organized as follows. Section 2 describes the proposed rebooting procedure, the *equal deepest vertex first reboot with vertex contraction*, and proves that all switches can be rebooted within a finite number of rebooting procedures. Section 3 explains our implementation of the proposed

[1] Tottori University, Tottori 680–8550, Japan
[2] Kyushu University, Motooka, Fukuoka 819–0395, Japan
a) ohmori@tottori-u.ac.jp
b) oka@ec.kyushu-u.ac.jp

rebooting procedure including how to obtain a campus network graph. Section 4 then evaluates the proposed method using the prototype implementation in an actual campus network. Section 5 discusses the validity of pre-conditions in this paper and further the applied usage of the proposed method. Section 6 refers to related work. Section 7 finally concludes this paper.

2. Equal Deepest Vertex First Reboot with Vertex Contraction

This section proposes the *equal deepest vertex first reboot with vertex contraction*, and proves that reboots finish within a finite number of rebooting procedures.

This section firstly presents pre-conditions of a campus network in this paper. This section represents a campus network in a graph theory fashion. This section explains *vertex contraction* in order to simplify a graph of a campus network. This section then presents how to compute a depth of a vertex in a contracted graph using matrix computations, and proves that a vertex contraction can reduce matrix computations and the computations finish within less than the maximum depth of vertices. This section proposes the procedure to reboot switches.

2.1 Pre-Condition

Suppose a campus network holds the following pre-conditions.

(1) a core switch is installed in each campus as a root,
(2) distribution switches are directly connected to a core switch, and accomodate edge switches,
(3) edge switches accomodate end hosts,
(4) an *upstream* edge switch, which is nearer toward a core switch and directly connected to a distribution switch, may also accommodate another *downstream* edge switch, which is farther toward the core switch,
(5) *non-reboot switches*, i.e., core, distribution and some edge switches, do not require to be rebooted,
(6) all switches are connected in the spanning tree, i.e., there is no alternative path from an *upstream* edge switch to a *downstream* edge switch, the *upstream* edge switch has no other blocked port to the *downstream* edge switch, the *downstream* edge switch has no other discarding port to the *upstream* switch,
(7) an edge switch can be accessed only via the main data network, there is no additional management network that separately accomodates all edge switches and that is dedicated for management connections only in addition to the main data network,
(8) it is unpredictable whether an upstream edge switch is running during enough time to reboot a downstream edge switch right after "reboot command" is entered to the upstream edge switch due to network stability or other reasons,
(9) an edge switch is better to be rebooted in daytime, not at midnight,
(10) an edge switch cannot be scheduled to reboot autonomously or with other external equipment,
(11) an IP address of a core switch is given,
(12) IP addresses of other switches are not, and
(13) an IP address of a neighboring switch can be obtained by Link Layer Discovery Protocol (LLDP) [5], Cisco Discovery Protocol (CDP) or static manual definition as described in Section 3.

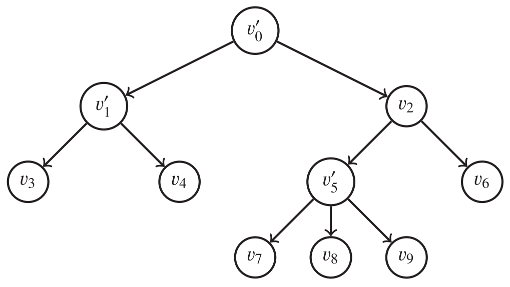

Fig. 1 An example of a representation of a campus network.

Note that our proposal may be able to be applied to a campus network where an edge switch has multiple paths toward a core switch. We, however, assume condition (6) that assumes the spanning tree because we do not have enough multiple paths in our actual campus network and the multiple paths cannot be evaluated enough in this paper. The case where an edge switch has multiple paths toward a core switch is out of scope of this paper, and is future work.

It is difficult to make a whole campus network stable at the same time. Our proposal described in later sections does not reboot all edge switches at the same time. Instead, we divide all edge switches into several groups in accordance with their *depths* as described later, and reboot edge switches for each depth in turn. In this way, we may be able to support a campus network that is not stable at the same time.

Regarding condition (9), one may consider why not reboot an edge switch at midnight. When we reboot an edge switch, the edge switch may never come up again, and be broken down. In this case, we might not able to replace the broken edge switch at midnight when we cannot physically access to the edge switch at midnight. This broken edge switch may incur a very longer downtime until tomorrow morning. We, therefore, assume condition (9) in this paper.

Regarding condition (11)–(13), a network topology map and port list of switches are not accurately maintained in our campus network. It is then almost unknown which port of which switch is connected to which port of which switch. We, therefore, need to be able to automatically collect these information with a few given information such as an IP address of a core switch.

2.2 Campus Network in Graph Theory

This section presents notation of a campus network in a graph theory fashion in this paper.

Figure 1 depicts an example of a campus network, and note that end hosts are omitted. In Fig. 1, a campus network is represented as an unweighted directed rooted spanning tree graph G. $V(G)$ and $E(G)$ then denote a set of vertices and arcs, respectively. A switch is a vertex, and i-th switch is denoted by $v_i \in V(G)$. Suppose *non-reboot switches* may not require to be rebooted due to some reasons such that they accommodate no end host or do not implement web authentication. *Non-reboot switches* depicted in Fig. 1 are v_0, v_1 and v_5, and they are marked by prime ($'$). Let v'_0 be

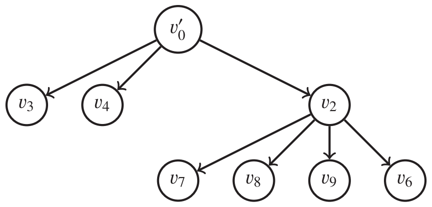

Fig. 2 An example of vertex contraction.

a vertex of a 0-th switch, i.e., a *root*. Let $indeg(\cdot)$ and $outdeg(\cdot)$ denote indegree and outdegree of a vertex, respectively. Let v'_0 be the only source vertex with $indeg(v'_0) = 0$ in G, i.e., a core switch, while there are sink vertices v_i with $outdeg(v_i) = 0$ which are connected to no *downstream* switch. *Downstream* here is a direction from the *root* v'_0 to other vertices. On the other hand, *upstream* is a direction toward a *root* v'_0. An upstream switch v_i is then *adjacent* (i.e., physically and directly connected) to another downstream switch v_j by an unweighted arc $v_iv_j \in E(G)$ where $i < j$. On the other hand, suppose the downstream switch v_j has no arc back to the upstream switch v_i for simplicity in discussions and computational efficiency. Note that traffic may actually go through from v_j to v_i in an actual network.

2.3 Vertex Contraction

As described in Section 2.1, some of switches, *non-reboot switches*, in a graph G do not require reboots. In order to consider an order to reboot switches, we can *contract* vertices of *non-reboot switches*. Note that a *vertex contraction* [3] here is the same as an edge contraction in this paper since a graph G is a spanning tree and two vertices to be contracted are always connected. **Figure 2** depicts a graph $G/V'(G)$ that contracts vertices of *non-reboot switches* from a graph G depicted in Fig. 1. Note that $V'(G)$ is a set of vertices of *non-reboot switches*, and "$/V'(G)$" represents an operation of vertex contraction of all *non-reboot switches*. In Fig. 2, v'_1 and v'_5 are contracted. A vertex contraction of a vertex v'_j is an operation to:
(1) remove an arc $v_iv'_j$,
(2) union arcs to downstream vertices of v'_j to v_i, and
(3) delete v'_j.
Note that a root vertex v'_0 may also be able to be contracted in theory. A root vertex v'_0 is, however, not contracted for simplicity in discussions here. Note that $G/V'(G)$ is then a spanning tree as well.

2.4 Computation of Depth of Vertex

A *depth* of a vertex is the number of arcs from a root in a graph, and a depth of a root is 0. Let us focus on a depth in a contracted graph $G/V'(G)$, not in a graph G. In Fig. 2, the depths of $\{v_0\}$, $\{v_2, v_3, v_4\}$ and $\{v_6, v_7, v_8, v_9\}$ are 0, 1 and 2, respectively. A depth of each vertex can be obtained as follows.

Let A be an adjacency matrix for $G/V'(G)$, and a_{ij} be an element (i, j) of A where i and j denote indices of v_i and v_j, respectively. Let a^n_{ij} be an element (i, j) of A^n. Note that indices of contracted vertices are not included in A and A^n.

Corollary 1. For an index of each vertex in $G/V'(G)$ except for a root v_0, there exists one and only one $n \in \mathbb{N}$ such that $a^n_{0j} = 1$, and n will be a depth of v_j.

Proof. $a_{ij} \in \{0, 1\}$ since $G/V'(G)$ is an unweighted graph. In accordance with a nature of an adjacency matrix, if $a^n_{ij} \neq 0$ then n is the number of vertices of a *simple path* from v_i to v_j and a^n_{ij} is the number of *simple paths*. Actually $a^n_{ij} \in \{0, 1'\}$ because $G/V'(G)$ is an unweighted spanning tree, and there exists only one *simple path* from v_i to v_j if exists. There also exists one and only one *simple path* from v_0 to v_j. There, hence, exists one and only one $n \in \mathbb{N}$ such that $a^n_{0j} = 1$, and n is then a depth of a vertex v_j. □

Corollary 2. There exists $m \in \mathbb{N}$ such that $A^m \neq O$ and $A^{m+1} = O$. m is then the maximum depth of vertices in a graph $G/V'(G)$.

Proof. A is an upper triangular matrix because $G/V'(G)$ is a directed graph and there are no back arc to an upstream vertex. The main diagonal components a_{ii} of A are all zeroes since an arc from v_i to v_i itself never exists. A is then a strictly upper triangular matrix, and nilpotent, say with $A^r = 0$ [4]. A then holds $\{{}^\exists m \in \mathbb{N} \mid A^m \neq O \wedge A^{m+1} = O\}$. The maximum depth of vertices is then m. □

Corollary 3. Depths of all vertex can be obtained by computing A^n while $n \leq m$.

Proof. $A^n = O$ where $n > m$. Since $G/V'(G)$ is a spanning tree, there exists one and only one *simple path* from v_0 to v_j. For each vertex in $G/V'(G)$ except for a root v_0, there exists one and only one n such that $a^n_{0j} = 1$ and $n \leq m$. Depths of all vertices can be then obtained by computing A^n while $n \leq m$. □

Let us take a look at an example of an adjacency matrix of $G/V'(G)$ depicted in Fig. 2. The adjacency matrix A is expressed in Eq. (1). The square and cube of A are Eq. (2) and Eq. (3), respectively. The depth of $\{v_2, v_3, v_4\}$ and $\{v_6, v_7, v_8, v_9\}$ is 1 and 2, respectively. The maximum depth of vertices is then 2.

$$A = \begin{pmatrix} a_{00} & a_{02} & a_{03} & a_{04} & a_{06} & a_{07} & a_{08} & a_{09} \\ a_{20} & a_{22} & a_{23} & a_{24} & a_{26} & a_{27} & a_{28} & a_{29} \\ a_{30} & a_{32} & a_{33} & a_{34} & a_{36} & a_{37} & a_{38} & a_{39} \\ a_{40} & a_{42} & a_{43} & a_{44} & a_{46} & a_{47} & a_{48} & a_{49} \\ a_{60} & a_{62} & a_{63} & a_{64} & a_{66} & a_{67} & a_{68} & a_{69} \\ a_{70} & a_{72} & a_{73} & a_{74} & a_{76} & a_{77} & a_{78} & a_{79} \\ a_{80} & a_{82} & a_{83} & a_{84} & a_{86} & a_{87} & a_{88} & a_{89} \\ a_{90} & a_{92} & a_{93} & a_{94} & a_{96} & a_{97} & a_{98} & a_{99} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & & & & \end{pmatrix} \quad (1)$$

$$A^2 = \begin{pmatrix} 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & & & & \end{pmatrix} \quad (2)$$

$$A^3 = 0 \quad (3)$$

2.5 Equal Deepest Vertex First Reboot Procedure

We propose the *equal deepest vertex first reboot* to reboot the *deepest* switches first. The *deepest* here means that the depth from a root v_0 in a contracted graph $G/V'(G)$ is the deepest among vertices of switches that are not rebooted yet. Switches whose vertices are in the same depth are *simultaneously* rebooted. *Simultaneously* here implies *concurrently* or *parallely*, and it depends upon an implementation. The switches must be then confirmed if the switches receive *reboot commands*, e.g., whether TCP acknowledgement for a segment containing *reboot commands* is received, whether connections are closed or otherwise. The next deepest vertices are then examined to be rebooted. In this way, the *equal deepest vertex first reboot* ensures that upstream switches are not rebooted before downstream switches are. This avoids the worst case where a downstream switch waits to be rebooted after an upstream switch is rebooted and becomes up, and then minimizes a *downtime*.

3. Implementation

This section presents our prototype implementation to reboot edge switches in a campus network to activate a new digital certificate for web authentication. This section firstly explains an implementation of the *Networking Utilities* that connect to a switch and execute a Command Line Interface (CLI) command on the switch. This section explains how to simultaneously find and reboot switches in different campuses. This section then presents how to automatically find a switch and produce a campus network graph. This section then presents how to determine whether a switch requires a reboot or not. This section also presents how to commit to reboot a switch.

3.1 Networking Utilities

In order to control a switch, we have implemented the *Networking Utilities* that connect to a switch, and execute a command on a switch. The utilities are written in Ruby, and can be run with Ruby 1.9.3 or later. The utilities require net/telnet and net/ssh library, and then connect to a switch by telnet or SSH. The utilities are *free software* licensed with 2-clause BSD license, and available on https://github.com/ohmori7/netutils.

The motivation to implement the *Networking Utilities* with traditional telnet or SSH is that recent REST or other APIs of network equipment do not support all commands, especially, complicated but frequently employed commands in an actual network such as Virtual Routing and Forwarding (VRF) related commands. The characteristics of the utilities can be summarized as follows.

(1) automatic network equipment discovery: the utilities support discovering a neighboring equipment using LLDP and CDP. Simple Network Management Protocol (SNMP) [6] is not supported because SNMP tends to burden much on a network and network equipment. An operator then must give at least an IP address of one of root switches. Other switches are, however, not necessary to be statically defined in advance.

(2) static neighbor definition: network equipment may not be able to run both of LLDP and CDP. The utilities support statically defining such network equipment by an outgoing interface, host name and IP address.

(3) vendor lock-in free: the utilities currently supports Cisco router C1812J, Cisco catalyst 6500, 3560, 2960, AlaxalA AX8600, AX3800, AX3650, AX2530, AX2200, AX620, NEC IX2215, Palo Alto Networks PA-5220, PA-3020, PA-850 and Aruba mobility controller 7210.

(4) automatic maker and product detection: the utilities can detect a maker of network equipment when connecting to the equipment. The utilities can then detect a product of the equipment.

(5) multiple accessing methods support: the utilities can connect to network equipment using telnet, SSH or both of them. This feature is motivated by the fact that some switches are configured to accept telnet only while others are SSH only due to different system integrators or other reasons.

(6) multiple accounts support: the utilities can try multiple accounts to connect to network equipment. This feature is useful for privilege separation among different operators or system integrators.

3.2 Different Graph per Campus

A campus network in a university can be represented as a single graph, i.e., all network switches can be rooted to a single core switch in a single campus even though there may be multiple campuses. We, however, divide a campus network per campus, and each campus has each root of a core switch in each campus since a core switch cannot be a neighboring switch of another campus due to a link issue where LLDP or CDP cannot be implemented. Our implementations, however, regard switches at the same depth from their root in their campus as being at the same depth even though their own root core switches are different. That is, all switches in all our campuses are simultaneously rebooted.

3.3 Automated Network Graph Production

In our campus network, a network topology map and port list of switches are not accurately maintained. It is then almost unknown which port of which switch is connected to which port of which switch. In addition, the prototype implementation focuses on the case where a switch should be installed a new digital certificate and then rebooted. There is no document that clearly states which switch runs web authentication. In other words, we do not know which switches should be rebooted.

We then decide to automatically build the network topology map. To this end, we employed either LLDP or CDP on all switches except for switches that could not run both of LLDP and CDP. We also supported statically defining such switches as a *static* neighbor of other switches.

As an IP address of a core switch in each campus is given, IP addresses of neighboring switches are obtained by examining the output of LLDP or CDP. All other switches are then obtained by examining neighboring switches one by one. In order to avoid an infinite loop, a switch is identified by a host name, and already examined switches, i.e., upstream switches, are examined only once.

Note that telnet or SSH connections used for a network graph production are kept for further procedures described in later sections.

3.4 Non-Reboot Switch Detection

We here focus on rebooting edge switches in a campus network to activate a new digital certificate for web authentication. We decided to regard network equipment as being rebooted if and only if the following conditions hold:

(1) the network equipment is a switch,
(2) web authentication is enabled, and
(3) a switch is not rebooted after a new certificate is installed.

The former condition (2) is examined by checking to see if a configuration of a switch includes below line:

```
web-authentication system-auth-control
```

This may require a longer time to detect since outputting a configuration requires a longer time, and there would be a better way to detect. This improvement will be future work.

The latter condition (3) is examined by comparing the output of following commands:

```
show web-authentication ssl-crt
show system
```

The former output shows when a certificate is installed. The latter output shows when a switch boots.

3.5 Rebooting Switches

After a network graph is produced and non-reboot switches are detected, vertices of non-reboot switches are contracted. A depth of each vertex from a root is then computed, all vertices of switches are grouped by depth. In order to minimize a delay of a reboot and avoid a network failure of SSH or telnet on a reboot in advance, all edge switches are kept to be logged in after switches are detected as described in Section 3.3. If a switch cannot be logged in during a detection, reboots of all switches can be canceled.

The switches of the deepest vertices are then *concurrently* examined in the *equal deepest vertex first* fashion using threads. The reason why this is *concurrently*, not *parallely*, is just because of a Ruby implementation of threads that can run only one thread at the same time.

When a switch is examined, its configuration is saved if necessary because there may be an unsaved configuration left. If a reboot command is executed even though there is an unsaved configuration, a switch may interactively ask via CLI if one can discard the unsaved configuration or not. Because it would be difficult to automatically and properly handle this interactive dialogue, we here save a configuration before rebooting.

A switch is then rebooted, i.e., a reboot command is sent to a switch. A switch is confirmed to accept a reboot command if the switch holds one of the following conditions:

- CLI prompt is returned back to the *Network Utilities*, or
- a switch closes a TCP connection of a telnet or SSH connection.

In our environment, above both cases can be observed when a switch is rebooted.

All switches of the deepest vertices to be rebooted are waited to be confirmed to accept reboot commands. This careful confirmation avoids a switch to fail to be rebooted as much as possible. After all switches are confirmed, the next deepest vertices are then examined. The detailed process of rebooting switches is shown

```
# connect to a switch via a network in advance.
for v in vertices do
        v.connect
end
# reboot switches at the same depth.
for depth in max_depth(vertices) .. 0 do
        #
        # concurrently or parallely reboot a switch
        # in order to minimize downtime.
        #
        threads = []
        vertices.get(depth).each do |v|
                threads <<= Thread.new do
                        #
                        # save configuration for
                        # safety.
                        #
                        v.save_configuration
                        #
                        # send a reboot command
                        # to a switch.
                        #
                        v.reboot
                        #
                        # make sure that a switch
                        # accepts a reboot.
                        #
                        v.wait_for_acknowledgement
                end
        end
        #
        # ensure that all switches at the same depth
        # are being rebooted.
        #
        threads.each do |t|
                t.join
        end
end
```

Fig. 3 A pseudo code of the equal deepest vertex first reboot.

in **Fig. 3**.

4. Evaluations

This section evaluates the *equal deepest vertex first reboot with vertex contraction* using the prototype implementation described in Section 3 in an actual campus network. This section firstly presents a campus network configuration and an evaluation environment. This section then presents time to produce a network graph, and the overhead of wating time to reboot all switches. This section also presents the downtime before all switches are booted.

4.1 Evaluation Environment

We, Tottori University, have three main campuses, Koyama, Hamasaka, Yonago and tree branch offices. Koyama and Yonago were connected by 10 Gbps. Koyama and Hamasaka were connected by 1 Gbps by commercial VLAN service. The branch offices were connected to Koyama by 1 Gbps. The number of network equipment supported by the *Network Utilities* in our campus network was 317 in total. The network equipment broken-down

Table 1 Network equipment breakdown.

maker	model	number
AlaxalA	AX8600S08	1
AlaxalA	AX8600S16	1
AlaxalA	AX260A-08T	8
AlaxalA	AX2530S-08P	29
AlaxalA	AX2230S-24P	65
AlaxalA	AX2530S-24T	85
AlaxalA	AX2530S-24T4X	3
AlaxalA	AX2530S-48T	87
AlaxalA	AX2530S-48T2X	8
AlaxalA	AX2530S-48P2X	8
AlaxalA	AX3650S-20S6XW	5
AlaxalA	AX3650S-48T4XW	1
AlaxalA	AX3830S-32X4QW	1
AlaxalA	AX3830S-44XW	2
NEC	IX2215	4
Aruba	Aruba7210	2
Palo Alto	PA-5220	1
Palo Alto	PA-3020	1
Palo Alto	PA-850	1
Cisco	C3560E	1
Cisco	C2960C	3
Total		317

Table 2 Networking utilities server specification.

CPU	Intel(R) Xeon(R) CPU E5-2697 v2 2.70 GHz (allocted one core)
memory	2 GB
OS	CentOS 7.4.1708 (64 bit)
Programming Language	ruby 2.5.1p57 (2018-03-29 revision 63029) [x86_64-linux]

by model is shown in **Table 1**. 13 switches and network equipment were then defined as a root. The number of edge switches to be rebooted was 288. All switches synchronized time using Network Time Protocol (NTP) [7] and its clock error would be within 10 ms. All switches also run Rapid Spanning Tree Protocol (RSTP) [8] per VLAN in order to avoid a loop.

We implemented the *Network Utilities* on a server running on an ESXi as a Virtual Machine (VM) as shown in **Table 2**. The server was connected to a root core switch via a non-reboot switch, and there was no rebooted switch between the server and the core switch. We also implemented to measure and record time in *Network Utilities*.

4.2 Time to Produce Network Graph

As described in Section 3.3, a network graph is automatically produced. If a network graph is produced within a short time, the network graph production can be done when rebooting all switches. If its production takes a long time, its production should be done in advance. In order to clarify this issue, we measured time to produce a network graph.

It took about 34.19 seconds in our campus network to automatically produce a network graph. This time can be considered to be short enough to produce a network graph at the same time when rebooting all switches. Our implementation then produces a network graph at the same time. Our implementation is, however, limited to connect to 64 switches at the same time. This limitation is intended to avoid memory and CPU consumptions that may incur a longer delay resulting from swap out. It would be possible to increase this number or remove this limit, and these are future work.

Table 3 Overhead to reboot switches.

depth	switches	waiting time (sec.)
1	100	2.82
2	128	2.86
3	42	5.74
4	18	5.40
total	288	16.82

4.3 Overhead Waiting Time to Reboot Switches

The *equal deepest vertex first reboot* simultaneously reboots switches at the same depth from a root in the *deepest vertex first fashion*. The *equal deepest vertex first reboot* then waits for all switches at the same depth to surely receive a reboot command before examining the next less deeper vertices. Since this waiting time could be overhead, we measured this waiting time. As shown in **Table 3**, the maximum depth was 4 while the actual maximum depth, i.e., without vertex contraction, was 6. Note that root switches, all of which were *non-reboot switches*, are omitted in Table 3. All switches were waited to be rebooted for about 16.82 seconds in total. This waiting time was shorter than the time to detect all switches and produce a network graph, 34.19 seconds, because the telnet and SSH connections were reused when rebooting switches. Note that all switches are concurrently rebooted as described in Section 3.5.

It can be then said that this waiting time was shortened by connecting to all switches before rebooting them.

In addition, the time was interestingly not correlated to the number of switches. The number of switches at the depth 2 and 3 were 42 and 18, and they required waiting time of 5.74 and 5.40 seconds, respectively. On the other hand, the number of switches at depth 0 and 1 are 100 and 128, and they required waiting time of only 2.82 and 2.86 seconds, respectively. As shown Table 1, there were many models of switches, and the difference of the models could be the cause.

It can be also said that the vertex contraction reduces the total waiting time because more than two seconds seems to be required for each depth.

4.4 Downtime

When switches are rebooted, a *downtime* is the most important. In this paper, a *downtime* is the time required to reboot all switches requiring reboots. To be more specific, a *downtime* is the time between:

- when the first rebooted switch stops forwarding packets and
- when the last rebooted switch becomes up and starts to forward packets.

We then measured the downtime by RSTP logs of switches. An upstream switch of a firstly rebooted switch could log the time when a packet forwarding was blocked on a port to which the firstly rebooted switch was connected. The time was regarded as the beginning of downtime. On the other hand, the time that a switch lastly started to forward packets on the last VLAN was regarded as the end of downtime. The downtime was 109 seconds. Note that accuracy of timestamps of logs was in second.

Let us compare this downtime and the ideal minimum downtime. The ideal reboot operation is rebooting all edge switches requiring reboots at once. This operation ideally minimize the

Table 4 Reboot time of switches.

model	reboot time (sec.)
AX260A-08T	63
AX2530S-08P	74
AX2230S-24P	69
AX2530S-24T	70
AX2530S-24T4X	68
AX2530S-48T	75
AX2530S-48T2X	73
AX2530S-48P2X	93

downtime that is equal to $\max\{v_i \in G/V'(G)|T_{reboot}(v_i)\}$ where $T_{reboot}(\cdot)$ denotes the time that the edge switch, v_i, reboots and becomes up to start to forward packets. Note that this operation is ideal, and not feasible in an actual environment because it is almost impossible for an operator to directly connect to all switches via physical console cables, not a network. If an operator accesses to all switches via a network, an operator may not be able to reboot all switches at the same time because an order of arrivals of packets containing a reboot command will not be ensured. In order to find $\max\{T_{reboot}(v_i)\}$, we measured the downtime of model of rebooted switches as shown in **Table 4**. Note that models in Table 4 are filtered from those in Table 1. Table 4 shows only the downtime of rebooted switches that require reboots in our environment because downtimes of other switches that do not require reboots are not related to the downtime of *equal deepest vertex first reboot* in our environment. It could be said that $\max\{T_{reboot}(v_i)\}$ was at least more than 93 seconds. Suppose that 93 seconds was $\max\{T_{reboot}(v_i)\}$, and it could be said that the overhead of *equal deepest vertex first reboot* was about 16 seconds. It can be said that *equal deepest vertex first reboot with vertex contraction* can reboot switches only by approximately 17.2% downtime increase. 16 seconds almost similar to 16.82 seconds of waiting time as shown in Table 3. It could be, therefore, said that overhead of *equal deepest vertex first reboot with vertex contraction* is resulting mainly from waiting time to confirm if a reboot command is surely sent to a switch on a reboot.

4.5 Comparison with Manual Operation

We here compare our proposal with manual operation briefly. Before we implemented our proposal, we needed a few days to build recent correct network topology maps and detect edge switches that required reboots because network maps were not properly maitained. After that, we needed to determine the order to reboot edge switches, and it took several hours. On the other hand, our propsal can reduce the required time of the above operation to 34.19 seconds as described in Section 4.2. It can be said that our proposal can reduce the required time of preparation operations described above.

In addition, a human operation to manually reboot one edge switch required 1 second or more including switching a screen of an operation terminal, and its required time depended upon operators skills. Human operations might require approximately 288 seconds for 288 switches at least in our environment even though we could not properly measure the required time because we could not finish rebooting all switches without other operations. It can be said that our proposal can reduce the downtime from 288 seconds to 109 seconds as described in Section 4.4.

5. Discussions

5.1 RSTP Convergence Time and Downtime

As mentioned in Section 4.4, our proposal, *equal deepest vertex first reboot with vertex contraction*, requires the overhead of 16 seconds. This overhead may include the RSTP convergence time. The RSTP convergence time may depend upon how fast a rebooted edge switch can be found by other upstream or downstream edge switches. In our environment, the RSTP hello interval was 2 seconds. The first message, Bridge Protocol Data Unit (BPDU), of RSTP can be delayed when an edge switch is booting. The RSTP convergence time, therefore, may be more than few seconds. The RSTP convergence time, however, may not be most of the overhead, 16 seconds. If we can eliminate RSTP, the RSTP convergence time can be reduced, and our proposal can reboot all edge switches with less downtime.

5.2 Simultaneous Reboot versus Segment Reboot

One may think of organizing a reboot schedule for a department or a section in order to avoid a critical downtime. It may be, however, very difficult and take a long time to organize a reboot schedule for each department or section. In addition, all ports of all switches should be tracked, i.e., which port is connected to which room. We actually have been unable to track all ports of all switches for a long time, and a new switch is being installed on an on-demand basis.

On the other hand, this paper has proposed to simultaneously reboot all required switches. The downtime of the proposed procedure is just 109 seconds, and it could be short enough. The proposed procedure can be executed during a lunch break or a day off with prior notification.

It may be, however, necessary to consider a wireless Access Point (AP) powered by PoE. If an AP is accommodated by a PoE capable network switch and its switch should be rebooted, a wireless network may incur a longer downtime. We actually have experienced a longer downtime of a wireless network. This downtime can be avoided or reduced by considering a network configuration, and this is our future work.

5.3 Autonomous Reboot Scheduling

One may conceive to schedule a reboot by time when rebooting many switches. A Cisco edge switch, indeed, has a command, `reload at HH:MM`, for that purpose. Because an error of NTP [7] is within 10 ms, this scheduling method is more accurate than the proposed method. The proposed method may be, however, useful for a network equipment that does not support to schedule a reboot. Our network switches, actually, do not have an autonomous reboot scheduling function. In addition, the proposed method may be still useful for a firmware update or a special configuration that cannot be scheduled.

5.4 Core and Distribution Switch Reboot

Regarding core or distribution switches, they may require a reboot for a firmware update but they may require no downtime using Non-Stop Forwarding (NSF) or Non-Stop Routing (NSR) with redundant control planes. On the other hand, They may require a reboot when configured to change the number of entries in ARP, NDP, flow and/or filtering tables. Core or distribution switches require a relatively longer downtime than edge switches, and their reboots may require a special consideration. In addition, core or distribution switches usually do not implement web authentication.

6. Related Work

Calvert et al. review the basic topological structure of the Internet, and present a modeling method [9]. Their method is mainly for a simulation experiment to generate a network topology, and does not focus on representing a campus network in a graph theory fashion.

Alderson et al. analyzed the Internet with a first-principles approach [10]. They analyzed the Internet using an actual data from actual ISPs. Their work is very useful for understanding the Internet topology. It, however, mainly focused on a tier-1 network, and did not model an enterprise network or a campus network.

Fujitsu Limited released Converged Fabric (C-Fabric) on 2013 that was virtualizable network switches and was dedicated for a data center [11]. C-Fabric has a function called *rolling reset* to reset all switches. *Rolling reset*, however, requires 5 minutes for each switch, and resets switches one by one [12]. *Rolling reset*, therefore, requires a longer time to complete to reset all switches.

7. Concluding Remarks

This paper has proposed the *equal deepest vertex first reboot with vertex contraction* that can simultaneously reboot many network switches with less overhead downtime. This paper has expressed a campus network in a graph theory fashion, has reduced downtime overhead by *vertex contraction*, and has proven that all switches can be rebooted within a finite number of rebooting procedures. This paper has presented an implementation of the proposed procedures, and has evaluated the proposed method in an actual campus network. The *equal deepest vertex first reboot with vertex contraction* has appeared to reboot all switches by only 16-second additional overhead out of 109-second downtime in total where the ideal minimum downtime was 93 seconds in an actual campus network where there were more than 300 network switches installed.

References

[1] Ohmori, M. and Higashino, M.: The Equal Longest Path First Reboot: Rebooting Network Edge Switches in a Campus Network, *IEICE Technical Report*, Vol.118, No.204, pp.83–89 (2018).

[2] Ohmori, M., Miyatal, N. and Suzuta, I.: AXARPS: Scalable ARP Snooping Using Policy-Based Mirroring of Core Switches, *Proc. 33rd International Conference on Advanced Information Networking and Applications* (*AINA-2019*), pp.667–676 (2019).

[3] Pemmaraju, S. and Skiena, S.: *Computational Discrete Mathematics: Combinatorics and Graph Theory with Mathematica*, Cambridge University Press, Cambridge, England (2003).

[4] Hesselink, W.H.: A Classification of the Nilpotent Triangular Matrices, *Compositio Mathematica*, Vol.55, No.1, pp.89–133 (1985).

[5] IEEE Std. 802.1ab 2004: *Local and Metropolitan Area Networks: Virtual Bridged Local Area Networks: Station and Media Access Control Connectivity Discovery*, The IEEE Standards Association (2004).

[6] Harrington, D., Presuhn, R. and Wijnen, B.: An Architecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks, RFC 3411 (Standard) (2002), Updated by RFCs 5343, 5590.

[7] Mills, D.: Network Time Protocol (Version 3) Specification, Implementation and Analysis, RFC 1305 (Draft Standard) (1992), Obsoleted by RFC 5905.

[8] IEEE Std. 802.1w-2001: *Local and Metropolitan Area Networks. Rapid Reconfiguration of Spanning Tree* (2002).

[9] Calvert, K.L., Doar, M.B. and Zegura, E.W.: Modeling Internet topology, *IEEE Communications Magazine*, Vol.35, pp.160–163 (1997).

[10] Alderson, D., Li, L., Willinger, W. and Doyle, J.C.: Understanding Internet Topology: Principles, Models, and Validation, *IEEE/ACM Trans. Netw.*, Vol.13, No.6, pp.1205–1218 (online), DOI: 10.1109/TNET.2005.861250 (2005).

[11] Fujitsu Limited: FUJITSU Intelligent Networking and Computing Architecture (2013), available from ⟨https://pr.fujitsu.com/jp/news/2013/05/8.html⟩ (accessed 2019-01-31).

[12] Fujitsu Limited: Converged Fabric Configuration Example (2013), available from ⟨http://jp.fujitsu.com/platform/server/primergy/manual/peripdf/ca92344-0344-05.pdf⟩ (accessed 2019-01-31).

Motoyuki Ohmori was born in 1976. He received his B.S. and M.S. degrees in Computer Science and Communication Engineering from Kyushu University in 1999 and 2001, respectively. He joined the Information Processing Society of Japan in 2001. He had been a lecturer at Chikushi Jogakuen University since 2004. He has been an associate professor at Tottori University since 2013. His research interest includes network architecture, multicasting, routing, mobile networking and energy efficient network operation. He is a member of the IPSJ, IEICE, JSSST, IEEE CS/ComSoc and ACM.

Koji Okamura was born in 1965. He received the BS, MS and Ph.D. degrees from Kyushu University in 1988, 1990, and 1998. He worked as an Associate Professor of Computer Center and Graduate School of Information Science and Electrical Engineering, Kyushu University. Since 2011, he has been a Professor of Kyushu University. He is the Director of Cybersecurity Center and the Vice Director of Research Institute for Information Technology, Kyushu University. He is also Vice CISO of Kyushu University. His current research interests are Cybersecurity for information network and social infrastructure and advanced operation technologies for Internet and Future Internet such as Openflow and Virtual Network. He is also researching power-aware and security-aware network operation and developing green power and secure network equipment system.