Design and Implementation of Support System for Network Testing with Whitebox Switches

Megumi Shibuya[†], Hidehiko Kawakami[†], Teruyuki Hasegawa[†] and Hirozumi Yamaguchi[‡]
[†]KDDI Research, Inc.
Saitama, JAPAN
e-mail: {shibuya, hi-kawakami, teru}@kddi-research.jp

[‡]Osaka University
Osaka, JAPAN
e-mail: h-yamagu@ist.osaka-u.ac.jp

***Abstract*—In this paper, we design and implement an automated network testing system that enables network testing operators to observe the behavior of networks under a variety of failures. We aim at automating the network failure testing of commercial networks, which is often time consuming since much effort is needed for network configuration, test scenario execution, evaluation report creation, and cause analysis of erroneous behavior. Moreover, we consider such an environment where whitebox switches and legacy switches co-exist. Whitebox switches are composed of commodity-based hardware components where Open Source Software (OSS), such as Chef can be used to manage them, while legacy switches are usually operated by vendor-dependent tools. Our system hides such difference and provides a seamless and automated management scheme in such heterogeneous environment. We have implemented the proposed system to confirm its effectiveness and feasibility.**

Keywords – Whitebox Switch; Chef; Automated Verification; OSS; DevOps

I. INTRODUCTION

Network providers need to assure a certain level of services in providing commercial networks. Such assurance would be necessary every time their elements (e.g., links and routers) are added, removed, updated, or replaced. As testing methodologies for IP networks, interconnect testing, stress testing, and failure testing are well-known and have been utilized. Those testing procedures contain test scenario generation, network testing environment preparation, test result analysis, and many other tasks.

In particular, failure testing is significant as a failure of network component often occurs. However, failure testing for commercial networks has several issues to be addressed. Firstly, it requires a large amount of time and efforts that should be dedicated to network equipment preparation, network cabling, network configuration including connectivity assessment, execution of a huge number of testing items, and analysis of the evaluation results. Furthermore, with increased size and complexity of commercial networks in recent years, cabling and setting errors are prone to occur when the network to be tested is constructed. This is because the man-hours for the testing has increased and testing is often manually undertaken by operators. Consequently, there are many cases in which a few months' work is required until they become ready for use [1][2]. This might be an obstacle to rapid deployment of the services.

In order to support network testing operators engaged in such time- and effort-consuming tasks of network failure testing, many network testers [3][4] and network simulators [5]-[7], which serve as automated network testing tools, have been manufactured. However, they are very costly and operations differ from each other. Thus, efficiency depends on the network testing environment of network equipment.

Recently, in contrast to legacy switches (denoted by *Legacy-SW*) where hardware and network OS are manufactured by conventional network equipment vendors, whitebox switches (denoted by *WB-SW*), which are composed of commodity-based hardware components and selectable network OSs, are becoming increasingly widespread [8]-[10]. As these network OSs are generally based on Linux OS, various testing processes can be automated by utilizing Open Source Software (OSS), such as Chef [11]. DevOps [12]-[15] is an emerging paradigm for fostering collaboration among system development and operations and it can improve the life cycle of software deployment and management processes. Besides, Chef and some other tools enable automated configuration of servers and clouds.

From a standpoint of network providers focusing on carrier-grade networks, we expect that Linux-based innovative WB-SWs will co-exist with Legacy-SWs in commercial networks. However, as the network OS of Legacy-SWs is not usually Linux-based, different automation techniques are necessary if these different SWs are co-used. Even in such heterogeneous environment containing both WB-SWs and Legacy-SWs, automation should be available to mitigate the workload of networking testing operators (*operational workload*).

In this paper, we assume future-generation commercial networks where Linux-based WB-SWs and Legacy-SWs co-exist. Then, we design an automated network testing system. The system is able to observe the behavior of the networks to be tested under given intended failures, collect testing results, and analyze them automatically. Heterogeneity due to different types of SWs is completely hidden and seamless testing operations are provided. The designed system has been implemented to confirm its effectiveness and feasibility. Specifically, we have implemented the automated network testing system that can automatically construct and configure the test target network containing both WB-SWs and Legacy-SWs, and have observed how the target network behaves in failure scenarios.

 ISBN: 978-1-61208-517-3

Since Legacy-SWs do not allow execution of OSS, a proxy to translate the Chef commands into Legacy-SW's is newly introduced. The proposed system automatically executes the testing procedure to confirm the influence of link failure and path switching on end-to-end communications. In addition, the system has a function to apply randomly-generated failures that occur in different times and locations. The results of such random tests are informative to find potential risks that may violate the fault tolerance of the target networks.

The contributions of this work are two-fold. Firstly, our system can deal with such an environment that contains both WB-SWs and Legacy-SWs. Although this is considerably important in assuring robustness and trustworthiness of carrier-grade commercial networks with lesser amount of human effort, no existing approach has explicitly and formally addressed this issue. Secondly, we show the effectiveness and feasibility of such a system by real implementation.

This paper is organized as follows. Section II summarizes related works and Section III explains the proposed automated network testing system. Section IV presents the experimental results of the proposed system. We conclude this work in Section V.

II. RELATED WORK

Recently, the concept of DevOps, which is an emerging paradigm to actively foster collaboration between system development and operations, is becoming popular. In order to speed up the improvement in the development quality, some automated tools, such as Chef, Puppet [16], and Ansible [17], have emerged for realizing DevOps, which are able to construct a server- and/or cloud-based infrastructure environment [2][12][13]. Using these tools, a testing environment is constructed automatically, which saves time and effort for testing operations.

As related work of automated network testing, the L1 patch [18], where whole OpenFlow switches are virtually seen as a single L1 patch panel using an OpenFlow technique [19], is proposed. By combining this approach with Mininet [20], an OSS-based network testing tool, the operational workload was mitigated and the dedicated time was just a few minutes per operator, while a half or an hour by two operators was necessary in the conventional work procedure. Reference [18] also reports that the number of test types was increased to 194, which had been just 90 in the conventional approach. Reference [21] proposes a method of constructing an automated test platform for Virtual Network Functions (VNF). In such an environment where virtual machines from different vendors are controlled by different virtual infrastructure managers, such as OpenStack and VMware Esxi, a network testing tool that automatically performs a series of tasks, such as test scenario selection and verification testing, is implemented using the commercial products CloudShell and TestShell [22]. Such comprehensive tests, which had required a lot of manual operations so far, can be automatically performed by using the tool, and 2,736 types of tests can be completed in 40 hours. However, this approach does not support Legacy-SWs, and thus applicability is rather limited until the ISP completely migrates into the NFV environment as described above.

WB-SWs have recently become increasingly prominent, and they are less expensive than Legacy-SWs. In addition, unlike the software-based virtual SWs, WB-SWs are generally implemented with ASICs in order to transfer the packets at high speed. Since most of the network OSs [8][23] for WB-SWs are based on Linux, it is expected that various types of tasks are automated using OSS. For example, Chef can be used to construct the environment automatically. In addition, Zabbix [24] and iperf [25] are known as a network monitoring tool and a traffic generator, respectively. However, considering such a situation where WB-SWs and Legacy-SWs co-exist, we also need to support Legacy-SWs that cannot execute OSS. Therefore, it is crucial that WB-SWs and Legacy-SWs can be operated seamlessly.

For these reasons, we assume that Legacy-SWs are still used with WB-SWs during the migration period. We also assume that there is a growing demand for automated network testing to mitigate management costs. Therefore, we design the automated network testing system enabling seamless operations with heterogeneous SWs. The system can create the network to be tested (denoted as *NtbT* hereinafter) and execute the failure scenarios automatically. In addition, as the size of the networks is still growing and configurations are becoming more complicated, comprehensive failure testing should also be supported where network problems are found by random failure generations with testing.

III. AUTOMATED NETWORK TESTING SYSTEM

In this section, we explain how our proposed automated network testing system is designed for commercial networks with WB-SWs and Legacy-SWs.

A. Design Principle

In general, a conventional test procedure consists of four steps as shown in Fig. 1 (a); create test items, construct NtbT, conduct tests, and confirm test results. Here, in order to reduce the operational workload, our proposed system automatically executes the following steps as shown in Fig. 1 (b); construct NtbT other than physical network cabling as step 2-2), conduct tests as step 3), and collect and analyze the results as step 4-1).

Step #	Workflow	Operation
1	Create test items	
2	Construct NtbT	
3	Conduct test	
4	Confirm results	

(a) Conventional network testing.

Step #	Workflow	Operation
1	-Select NtbT and scenarios -Schedule execution time	
2-1	-Physical NW cabling	
2-2	-Construct (include logical NW cabling) -Initialize -Collect NW info. -Confirm connectivity	Auto.
3	-Execute test scenarios -Collect NW info. -Confirm reachability	Auto.
4-1	-Collect logs and routing info. -Visualize results	Auto.
4-2	-Analyze results	

(b) Network testing by automated network testing system.

Figure 1. Comparative defintions of network testing procedures.

Figure 2 shows our proposed automated network testing system that consists of (1) NtbT with network equipment (WB-SWs (physical and/or virtual), Legacy-SWs, and hosts), and (2) a network testing control server (NW-CS) that sends to NtbT instructions, such as "construct NtbT" and "occurrence of failure". NtbT and NW-CS are connected via Control Plane (C-Plane), and SWs and hosts forward the traffic via Data Plane (D-Plane).

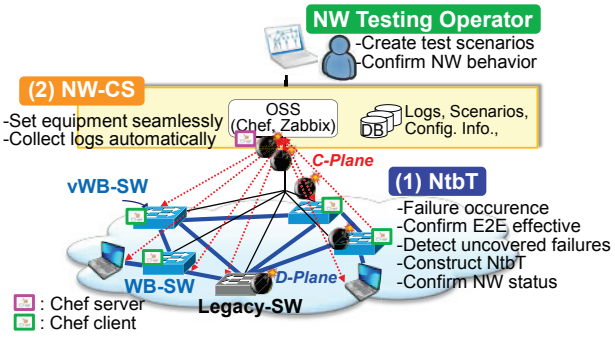


Figure 2. Automated network testing system.

We describe the design principles for each function below.

1) Automatic Construction of NtbT (for Step 2-2)

In order to construct NtbT automatically, Chef, which is an OSS infrastructure environment construction tool, is used. After completing the physical network cabling on NtbT, the Chef server instructs the Chef clients (i) to create the virtual WB-SWs (vWB-SWs) if necessary, (ii) to initialize and configure NtbT, and (iii) to start the routing protocol. This instruction is called *NtbT construction scenario*. To confirm the connectivity of the constructed NtbT, the Chef server conducts bidirectional reachability assessment on every link and between every host pair using *ping*.

2) Automatic failure occurrence (for Step 3)

The *failure scenario* specifies a series of failures in network testing. More concretely, the failure scenario consists of multiple *failure recipes*. A failure recipe corresponds to failure occurrence (e.g., link down and/or up). The location of failure in a recipe can be specified by the network testing operator (e.g., failures occur at SW#1) or can be randomly chosen. Then, the failure scenario specifies the execution ordering of those recipes. Network testing can be conducted repeatedly according to this failure scenario with the NtbT construction scenario. Both scenarios are stored in the database (DB). This is called *testing scenario*, which is explained in Section III-B. Furthermore, by the scheduling function, the testing scenario is executed at a specified time and date. Hence, it can be executed automatically at any time (e.g., nighttime or holidays).

3) Provisioning seamless operation of WB-SWs and Legacy-SWs (for Steps 1, 2-2 and 3)

To avoid complicated settings and operations due to coexistence of WB-SWs and Legacy-SWs, the settings and executing commands are made uniform for both types of SWs. However, Legacy-SWs cannot execute the Chef client because the network OS for Legacy-SWs is not Linux-based. Therefore, the proxy to translate the Chef commands to Legacy-SW's is prepared. We describe this in detail in Section III-C.

4) Network connectivity assessment (for Steps 2-2, 3 and 4-1)

In order to confirm bidirectional connectivity between adjacent equipment pair (via a certain link), Link Layer Discovery Protocol (LLDP) is used to collect the address information of the equipment pair, such as IP and MAC addresses. If LLDP is not applied to the target network, Address Resolution Protocol (ARP) is used instead.

5) Visualization of network behavior during and after network testing (for Steps 3 and 4-1)

To simplify the confirmation, the network behavior during network testing can be visualized in a real-time manner with network information in the above 4). Moreover, in order to confirm it after network testing, this network behavior can be replayed visually.

6) Assessment of failure influence on end-to-end communication (for Step 3)

During network testing, each host continuously sends packets to other hosts by *iperf* to confirm the connectivity between any pair of hosts. This is significant to observe the capability of the network to guarantee a certain level of service quality even when failures occur.

7) Automatic log collection and analysis (for Step 4-1)

Statistical log information, such as the traffic volume and CPU utilization, is collected by Chef and Zabbix. This log information is then analyzed and the results are shown in lists and/or plots on graphs.

8) Detection of uncovered failures (for Step 3)

Chaos Monkey [26][27] provides a function to generate failures randomly at any testing point. This often helps to find system errors and faults that have not yet been found in the system. Focusing on this feature, we integrate this function into our network testing system, i.e., failures occurrence order and/or locations that are randomly chosen. Concretely, failure candidates are categorized into some groups, such as links accommodated in the same switch and routers in the same OSPF area. The system must choose a candidate within the same group, which makes this random choice much more effective.

B. Testing Scenario

As we introduced earlier, a *testing scenario* consists of an NtbT construction scenario and a failure scenario. It is a testing procedure that is composed of seven *scenario recipes* as shown in Table I. This testing scenario is created by the network testing operator.

Again, the failure scenario consists of one or more failure recipes. We may select recipes from DB and some examples of those recipes are; (a) traffic sending/receiving with start and end time, (b) failure occurrence/recovery, such as interface (IF) down/up, (c) latency test, (d) load test, and (e) end-to-end quality measurement. The testing scenario is stored as the NtbT construction scenario and the failure scenario. After creating the testing scenario, the execution

 ISBN: 978-1-61208-517-3

time is scheduled at the scheduler. The execution is started automatically at the specified time and stopped when the failure scenario is completed.

TABLE I. SCENARIO RECIPES IN THE TESTING SCENARIO

Testing Scenario	No.	Scenario Recipes
NtbT construction scenario	1	Selection of NtbT topology
	2	Reconstruction of NtbT (Yes/No)
	3	Initialization of network equipment (WB-SWs, Legacy-SWs, hosts) (Yes/No)
	4	Confirmation of connectivity (Yes/No)
	5	Execution schedule (date and time, or at specific execution time)
	6	Collection of interval time of the network information and logs
Failure scenario	7	Failure recipes

C. System Implementation

Figure 3 shows the implementation of the automated network testing system. NW-CS consists of a Web server, databases that store logs, scenarios and configurations of NtbT topologies, a Chef server, and a Zabbix server on a PC. NtbT consists of an NtbT PC with vWB-SWs and virtual hosts (vhosts), physical WB-SWs, and Legacy-SWs. Chef clients run on the NtbT PC.

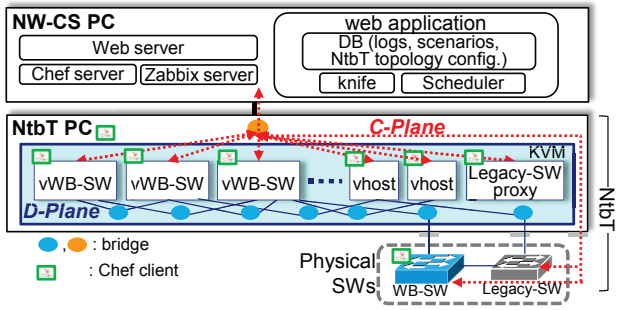


Figure 3. Implementation of automated network testing system.

When NtbT is reconstructed (if a recipe of "Reconstruction of NtbT" is "Yes"), vWB-SWs are constructed automatically in accordance with the selected NtbT. Furthermore, every adjacent SW (or host) pair is connected with a virtual link through a different bridge as logical NW cabling. The physical SW and virtual SW are also connected in the same way.

At the specified execution time, the Chef clients on each SW download the corresponding recipes from the Chef server via C-Plane. Then, the various settings are executed automatically based on the downloaded recipes. Furthermore, logs and routing tables of network equipment during the testing are collected automatically and stored in DB.

In order to separate C-Plane and D-Plane strictly where the packets in C-Plane must not leak into D-Plane and vice versa, each WB-SW arranges interfaces (IFs) and routing tables dedicated to C-Plane separated from those for D-Plane, by using two independent namespaces *mgmt* (for C-Plane) and *default* (for D-Plane) inside WB-SW [28]. Consequently, the Zabbix server collects the D-Plane information, such as routing tables of network equipment through *mgmt*.

To use uniform descriptions of the setting commands and method, it would be better that both WB-SWs and Legacy-SWs could be controlled according to the framework of Chef. Since the Chef client cannot be executed on Legacy-SW, a proxy that translates the Chef commands into the Legacy-SW commands and executes it automatically is created (called *Legacy-SW proxy*).

IV. EXPERIMENTAL EVALUATION

This section presents the evaluation results of the proposed system expressed in Section III.

A. Experimental Setup

In order to validate the effectiveness of our system implementation, we constructed NW-CS and NtbT on different PCs as shown in Fig. 3. The Chef server, Zabbix server, DB stored logs, and scenarios are implemented on NW-CS. We arranged three types of VMs as stand-alone on KVM [29] for vWB-SWs, the vhosts, and the Legacy-SW proxy composing NtbT. One WB-SW (DELL S4048-on, OS: Cumulus Linux) and one Legacy-SW (Catalyst 3750-24TS-E) also participate in NtbT via a physical interface. TABLE II summarizes the specifications of PCs for NW-CS and NtbT.

TABLE II. PARAMETERS OF THE AUTOMATED NETWORK TESTING SYSTEM

Parameter	NW-CS PC	NtbT PC
OS	Ubuntu 14.04 64bit	
Memory	64GB	
CPU	Intel® Xeon® CPU E5-2650+ v3@1.80GHz	
Chef	Server 12.4.0	Client 12.5.1
Zabbix	2.2.2(server)	2.2.2 (agent)
SW	-	Cumulus VX 2.5.6
Traffic generator	-	iperf
Routing software	-	quagga(ospf)

TABLE III. CONFIGURATION OF TESTING NETWORKS

NW No.	Network equipment (#equipment)	#Links	Failure recipes
1	vWB-SW(4), vhost(2)	7	IF down
2	vWB-SW(13), Catalyst(1), vhost(5)	25	1) IFdown 2) Silent failure
3	vWB-SW(13), DELL(1), vhost(5)	25	IF down
4	vWB-SW(12), Catalyst(1), DELL(1), vhost(5)	26	Change OSPF cost

TABLE IV. TESTING SCENARIO (NW NO. 2)

Testing Scenario	Recipe No.(*Rcp#*)	Scenario Recipes	*T* [sec]
NtbT construction scenario	1	Reconstruction of NtbT (Yes)	0
	2	Initialization of network equipment (Yes)	0
	3	Confirmation of connectivity (Yes)	0
Failure scenario	4	Start traffic send/receive	0
	5	Failure1 occurrence: wbs-core01 swp1 down	30
	6	Failure1 restoration: wbs-core01 swp1 up	30
	7	Failure2 occurrence: cat-agg20 FastEthernet1/0/2 down	30
	8	Failure2 restoration: cat-agg20 FastEthernet1/0/2 up	30
	9	End traffic send/receive	30
	10	Collection and analysis of logs	0

T: waiting time after completion of previous recipe.

 ISBN: 978-1-61208-517-3

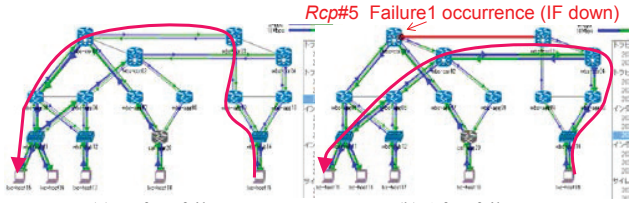
To evaluate our proposed system, we created the four types of experimental networks shown in TABLE III where NtbT is constructed automatically and all hosts generated traffic by iperf between every host pair. We then conducted each scenario 10 times. Note that physical SWs are cabled by hand preliminarily.

As an example, the testing scenario of NW No. 2 is shown in TABLE IV.

B. Verification of System Functions

Firstly, we verify that our proposed system can conduct network testing automatically. In order to confirm it, we executed network testing using four types of testing networks and corresponding failure scenarios in TABLE III. We verified that our proposed system can conduct all scenarios, and every scenario was executed without error. We validated that changing the routing paths when the link failure occurs and the OSPF cost changes, the disconnect time of end-to-end connections, the transition of the traffic volume of each interface (IF) at SWs, and the transition of the routing tables by visualization. Moreover, we verified that the Legacy-SW proxy works correctly.

As an example, Fig. 4 shows our visualization of end-to-end traffic paths before/after scenario recipe *Rcp*#5 of NW No. 2 in the failure scenario. We can observe the network behavior of path restoration using this visualization.



(a) Before failure (b) After failure

Figure 4. Example of routing path restoration after failure occurrence.

C. Performance Evaluation of Processsing Time

To evaluate the performance of our proposed system, we evaluated the processing time of each scenario recipe in the testing scenario during network testing. More concretely, the time to process each recipe was collected and analyzed.

The processing time of each recipe using NW No. 2 is shown in Fig. 5. The processing time differs according to the scenario recipe. The processing time has a small variance over recipes, and the maximum variance was 1.50 when the scenario recipe is the confirmation of connectivity (*Rcp*#3). The processing time of reconstruction (*Rcp*#1) was 110.0 seconds, while that of initialization (*Rcp*#2) was 138.1 seconds on average. The processing time of *Rcp*#1 was less than that of *Rcp*#2. The processing time of a failure occurrence / restoration recipe is the time by which the Chef client is executed till completion of the executed failure command at the corresponding SW.

Here, we consider two cases; Case #1 that consists of *Rcp*#5 and *Rcp*#6, and Case #2 that consists of *Rcp*#7 and *Rcp*#8. The processing time of Case #1 was 15.19 seconds and 21.94 seconds in Case #2. These two cases are similar in the sense that they have one failure and recovery, but the time is different. Cases #1 and #2 are different since #1 is an interface down failure and #2 is a "silent" failure that is difficult to detect. As seen in these cases, the processing time differs according to the failure recipe, but it is important to know such processing time of each recipe to estimate the time to conduct tests for large-scale networks. More detailed analysis of execution time is necessary, which is part of our future work.

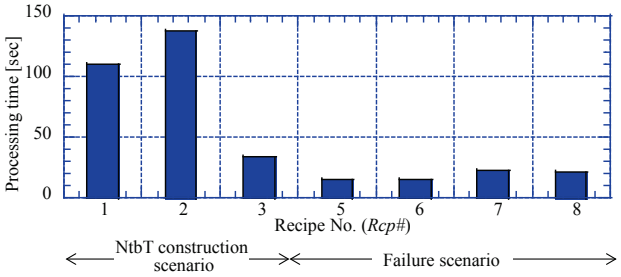


Figure 5. Processing time of each recipe (*Rcp*#).

D. Performance Evaluation of NtbT Construction Time

As our proposed system constructs NtbT including the construction and initialization of network equipment (SWs and hosts) in NtbT, it is important to perceive the processing performance of the construction NtbT for the number of SWs.

To evaluate it, we focused on the SWs because the construction time of SWs is longer than that of the hosts. In the evaluated network, vWB-SWs are connected in series on NtbT and hosts are connected to the edge of a path. Varying the number N of SWs from 5 to 60 units, we measured the processing time for constructing N vWB-SWs 10 times in each case.

Figure 6 shows the construction time on average of each recipe in the NtbT construction scenario. The processing time of each event increases in proportion to N. When N was 60, the total time for constructing NtbT was 480.4 seconds on average, and the processing time of construction increased in the following order; initialization > construction > connectivity. However, the increase rate of the processing time of *Rcp*#1 was larger than that of *Rcp*#2. Therefore, we estimate that when N is over 60, the processing time of *Rcp*#1 exceeds that of *Rcp*#2.

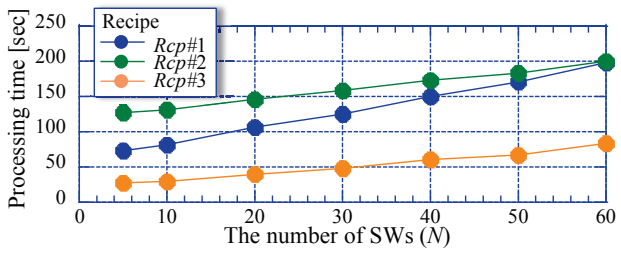


Figure 6. Number of WB-SWs and NtbT configuration time.

The reconstruction of NtbT (*Rcp*#1) creates VMs of vWB-SW only, while initialization of network equipment (*Rcp*#2) installs some software, such as OSS. Furthermore,

 ISBN: 978-1-61208-517-3

confirmation of connectivity (*Rcp#*3) is conducted via ping between both interfaces (IFs) of a link for all links and all hosts. Therefore, the processing time depends on the number of links and hosts. In this topology, due to the fixed number of hosts (=2), the processing time increases in proportion to the number of SWs.

E. Effectiveness in Reducing Work Hours

To evaluate how our proposed system contributes to reducing the operational workload, we observed the work hours during the network testing process by comparing manual and automated operations. Using NW No. 1, we prepared the testing scenario where an interface (IF) is down and then the routing path is switched, including network cabling, initialization, and failure occurrence. In Case-A, this scenario was conducted manually, and automatically conducted by the proposed system in Case-B. We measured them 3 times in each case.

The work hours were 16.03 and 7.37 minutes in Case-A and Case-B, respectively. Namely, the work hours were reduced by 54.1%. Note that each work hour includes 30-second intervals of each recipe. From this result, we expect more efficiency in large-scale networks, and we can say that our proposed system can reduce the work hours.

V. CONCLUSION

In this paper, we assumed future-generation commercial networks where Linux-based WB-SWs and legacy-SW co-exist. So, we proposed a method of designing and implementing an automated network testing system. This system is able to observe the behavior of the network to be tested under given intended failures, collect testing results and analyze them automatically. Heterogeneity due to different SWs is completely hidden and seamless operations are provided. We implemented the automated network testing system that can automatically construct and configure the test target network containing WB-SWs and Legacy-SWs, and observed how target network performs in failure scenarios. The experimental results show the effectiveness and feasibility of the system. In particular, it is confirmed that our implementation can automate failure testing on the network with WB-SWs and Legacy-SWs and can reduce the work hours by 54.1%. In future, we intend to verify the effect of the work hours using larger networks and various scenarios.

REFERENCES

[1] J. Kim, C. Meirosu, I. Papafili, R. Steinert, S. Sharma, F. Westphal, M. Kind, A. Shukla, F. Nemeth, and A. Manzalini, “Service Provider DevOps for Large Scale Modern Network Services,” IFIP/IEEE IM 2015 Workshop: 10th International Workshop on Business-driven IT Management (BDIM), pp.1391-1397, 2015.

[2] L. E. Lwakatare, T. Karvonen, T. Sauvola, P. Kuvaja, H. H. Olsson, and J. Bosch, “Towards DevOps in the Embedded Systems Domain: Why is it so Hard?,” 2016 49th Hawaii International Conference on System Sciences (HICSS), pp.5437-5446, Jan. 2016.

[3] Spirent TestCenter, http://www.spirent.com/Products/TestCenter, accessed Sep. 7, 2016.

[4] IXIA, https://www.ixiacom.com/, accessed Sep. 7, 2016.

[5] OPNET Modeler, http://www.riverbed.com/products/steelcentral/opnet.html, accessed Sep. 7, 2016.

[6] Cisco Modeling Labs, http://www.cisco.com/c/en/us/products/cloud-systems-management/modeling-labs/index.html, accessed Sep. 7, 2016.

[7] GNS3, https://www.gns3.com/, accessed Sep. 7, 2016.

[8] Cumulus, https://cumulusnetworks.com/, accessed Sep. 7, 2016.

[9] Quanta, http://www.networld.co.jp/product/quanta/, accessed Sep. 7, 2016.

[10] DELL, http://www.dell.com/us/business/p/networking-s-series-10gbe/pd, accessed Sep. 7, 2016.

[11] Chef, https://www.chef.io/chef/, accessed Sep. 7, 2016.

[12] C. Ebert, G. Gallardo, J. Hernantes, and N. Serrano, “DevOps,” IEEE SOFTWARE, Vol 33. No.3, pp.94-100, Apr., 2016.

[13] J. Allspaw and P. Hammond, “10+ Deploys Per Day: Dev and Ops Cooperation at Flickr,” O’REILLY Velocity Web Performance and Operations Conference (Velocity 2009), Jun. 2016, http://conferences.oreilly.com/velocity/velocity2009/, accessed Sep. 7, 2016.

[14] J. Wettinger, V. Andrikopoulos, and F. Leymann, “Automated Capturing and Systematic Usage of DevOps Knowledge for Cloud Applications,” IEEE International Conference on Cloud Engineering (IC2E), pp.60-65, Mar. 2015.

[15] A. Csaszar, W. John, M. Kind, C. Meirosu, G. Pongracz, D. Staessens, A. Takacs, and F. Westphal, “Unifying Cloud and Carrier Network EU FP7 Project UNIFY,” IEEE/ACM 6th International Conference on Utility and Cloud Computing, pp.452-457, 2013.

[16] Puppet, https://puppet.com/, accessed Sep. 7, 2016.

[17] Ansible, https://www.ansible.com/, accessed Sep. 7, 2016.

[18] http://www.sdnjapan.org/2015/1411_torii.pdf (in Japanese), accessed Sep. 7, 2016.

[19] OPEN NETWORKING FOUNDATION (ONF), “OpenFlow,” https://www.opennetworking.org/sdn-resources/openflow, accessed Sep. 7, 2016.

[20] Mininet, http://mininet.org/, accessed Sep. 7, 2016.

[21] http://www.okinawaopenlabs.org/wp/wp-content/uploads/20160205_tanabe.pdf (in Japanese), accessed Sep. 7, 2016.

[22] QualiSystems CloudShell / TestShell, http://www.qualisystems.com/products/cloudshell-add-ons/testshell-overview/, accessed Sep. 7, 2016.

[23] PicOS, http://www.pica8.com/products/picos, accessed Sep. 7, 2016.

[24] Zabbix, http://www.zabbix.com/, accessed Sep. 7, 2016.

[25] Iperf, https://iperf.fr/, accessed Sep. 7, 2016.

[26] NETFLIX, “Chaos Monkey Released Into The Wild,” http://techblog.netflix.com/2012/07/chaos-monkey-released-into-wild.html, accessed Sep. 7, 2016.

[27] A. Basiri, N. Behnam, R. Rooij, L. N. Hochstein, L. Kosewski, J. Reynolds, and C. Rosenthal, “Chaos Engineering,” IEEE SOFTWARE, May/Jun. 2016.

[28] Cumulus Networks, “Configuration Management Namespace,” https://support.cumulusnetworks.com/hc/en-us/articles/202325278-Configuring-a-Management-Namespace, accessed Sep. 7, 2016.

[29] KVM, http://www.linux-kvm.org/page/Main_Page, accessed Sep. 7, 2016.

 ISBN: 978-1-61208-517-3