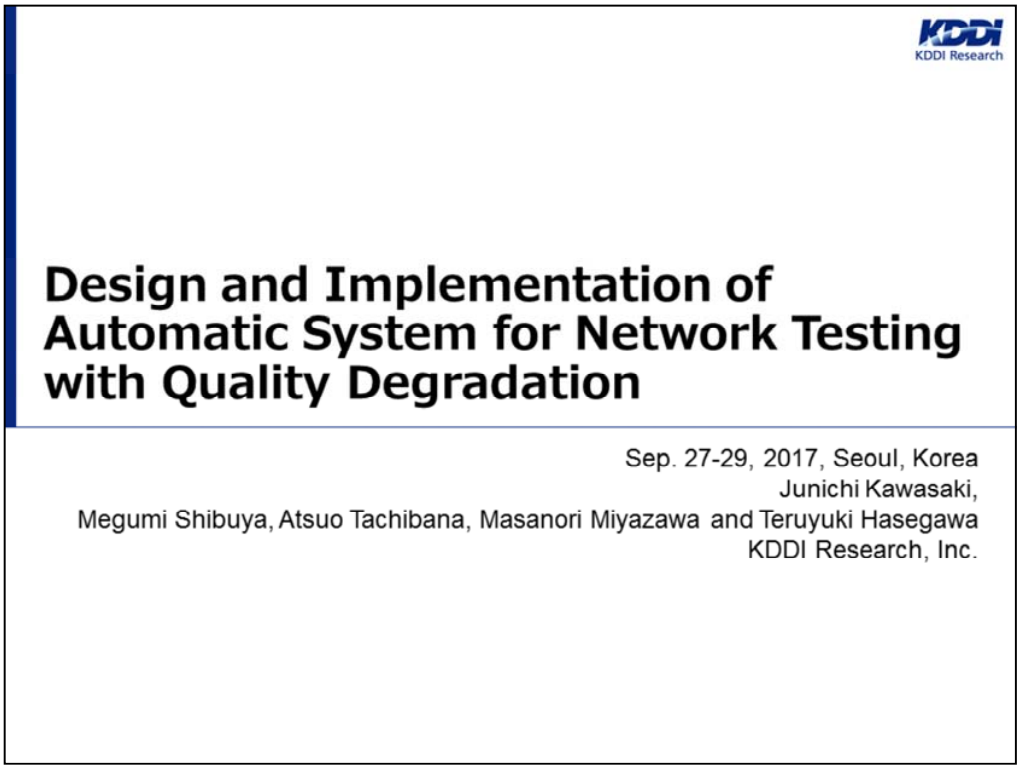
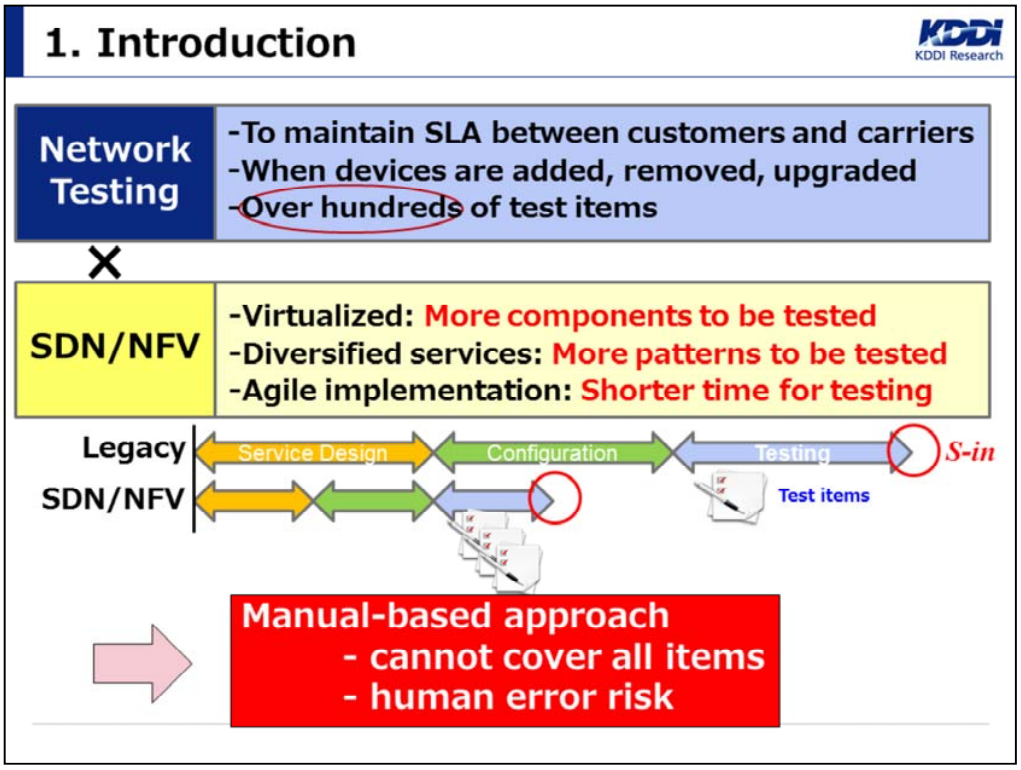


Abstract

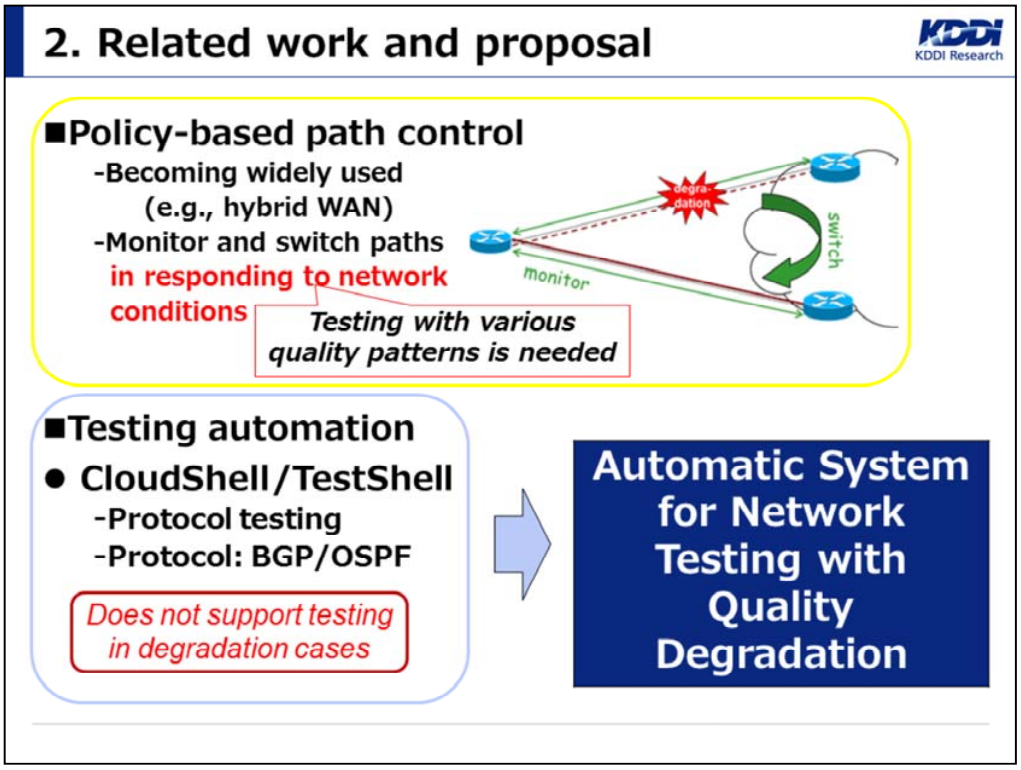
One of the key issues for telecom operators is how to efficiently achieve network testing in order to localize problems (e.g., software bugs) before deployment because both the number of testing scenarios which operators have to cover for improvement of service quality and the number of network equipment items have been increasing due to emerging advanced network technology such as policy-based path control in responding to network conditions (e.g., packet loss, delay and jitter), SDN and NFV. Hence, the increase in number of test scenarios is accelerating, and in future, such a test scenario explosion could lead a fatal limitation of the current manual-based approach, thereby making automatic network testing more valuable. To tackle this issue, we investigated an automatic system for network testing, and proposed a verification mechanism with quality degradation such as packet loss, delay and jitter supported by Open Source Software. We evaluated the proposed system using an experimental network of IP-Sec Gateway and the demonstration results show that the workload was successfully reduced by half, compared to that of the conventional approach.



1. Introduction

Telecommunication carriers need to provide a certain level of services in their commercial networks in order to maintain service level agreements (SLAs) with customers. To maintain reliable and stable network services, it is important to test the networks when new equipment is introduced and when existing devices are upgraded (e.g., next-up software version) or removed. The procedure for network testing basically consists of three steps; preparation of the testing environment, execution of the test, and analysis of the test results. The number of test items frequently exceeds several hundred; therefore, it is always a burden on the network testing team.

Furthermore, the number of components and the number of patterns to be tested will be much larger with the emergence of new advanced technology providing policy-based path control and software-based networking such as software-defined network (SDN) and network function virtualization (NFV), though the available time for testing will be shorter to cycle agile implementation of the services. Therefore, we will see a fatal limitation of the current manual-based approach: it could not cover verification of all test items and could cause human error.

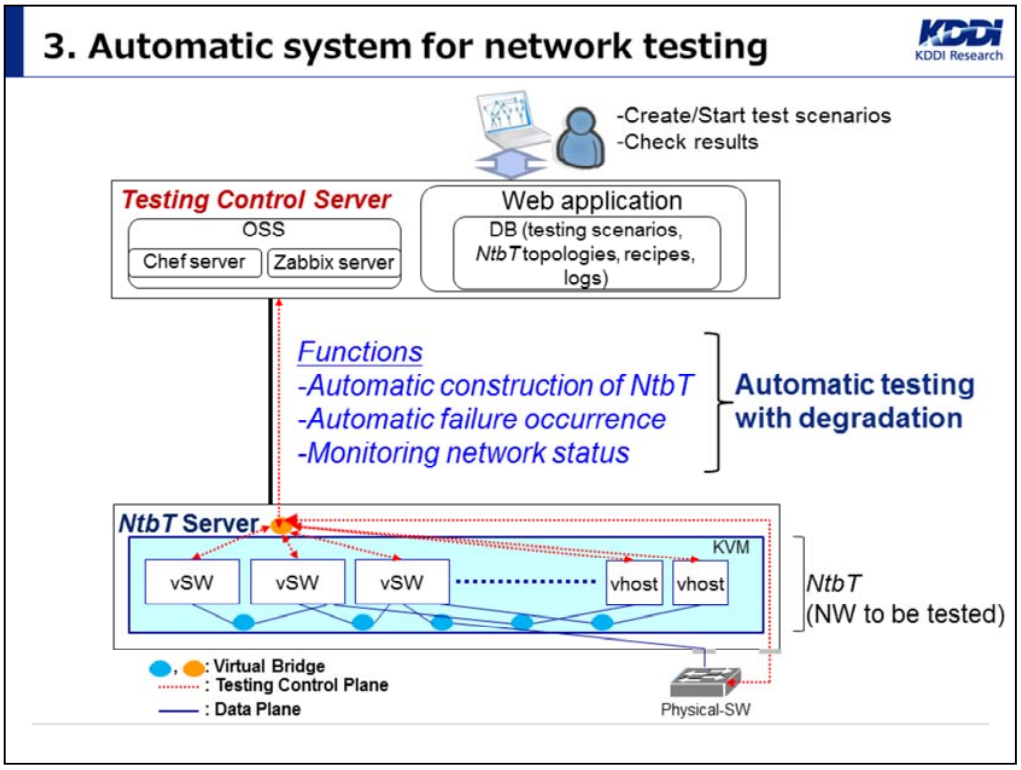


2. Related work and proposal

As one of the advanced network technologies, policy-based path control in responding to network quality is becoming widely used (e.g., hybrid WAN [1][2]). These types of devices have some parameters for switchover, and monitor the actual network conditions (e.g., packet loss, latency and jitter) and switch paths based on pre-defined thresholds. As verifying these networks requires repeated testing taking into account various quality patterns, it will be more time-consuming.

It is challenging to effectively verify the network under such advanced environments. In order to tackle that situation, we proposed a support system for network testing [3]. In our previous research, we implemented automation of failure testing to observe network behavior when a link failure occurs; however, that system does not support testing in degradation cases. Even though testing automation products (e.g., CloudShell/TestShell [4]) are becoming commonly used for network testing, they have not been applied to testing with latency and packet loss.

In this paper, we propose and demonstrate an automatic system for network testing which includes network quality degradation in order to support verification of policy-based path control.

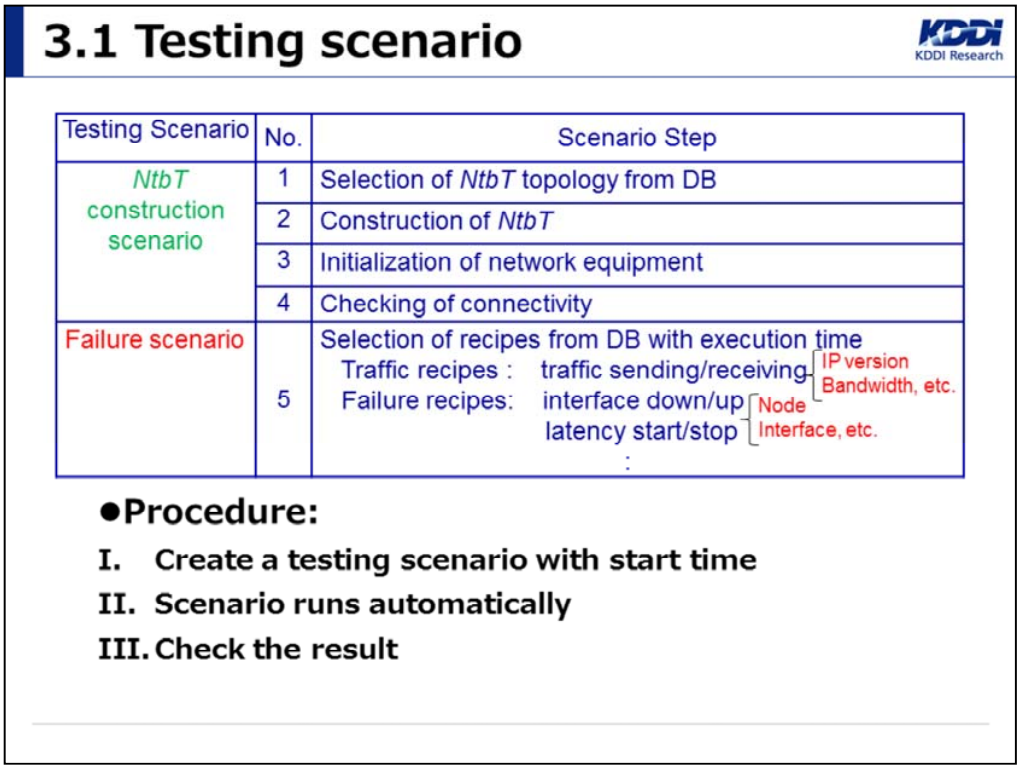


3. Automatic system for network testing

The picture above shows our proposed automatic system for network testing; it consists of a testing control server and a network to be tested (hereinafter "*NtbT*") server. The testing control server manages automatic testing with network quality degradation and the test runs on the NtbT implemented on the NtbT server. Three functions are provided by the control server to achieve the automation: automatic construction of the NtbT to prepare for the test environment; automatic failure occurrence to execute the failure testing command; and monitoring network status to collect the network information for analysis. The control server and the NtbT server are connected via the testing control plane and Open Source Software (OSS), Chef [5] and Zabbix [6] server on the control server exchange messages between the clients on the NtbT server through this control plane to support the automation.

The proposed system provides a Web user interface (UI) for operators to create and start a testing scenario and check the testing result on a Web browser. The database (DB) in the control server stores the testing scenarios, NtbT topologies and recipes that are necessary for creating scenarios (explained in details in the next slide), and result logs.

The NtbT consists of virtual switches and hosts on KVM and physical switches to establish the testing environment necessary for verification regardless of physical and virtual. They are connected to the testing control plane and the data plane by a virtual bridge. Thus, our proposed system can be used on any test environment (physical, virtual, and coexistence network) of carrier access network where policy-based control is implemented.



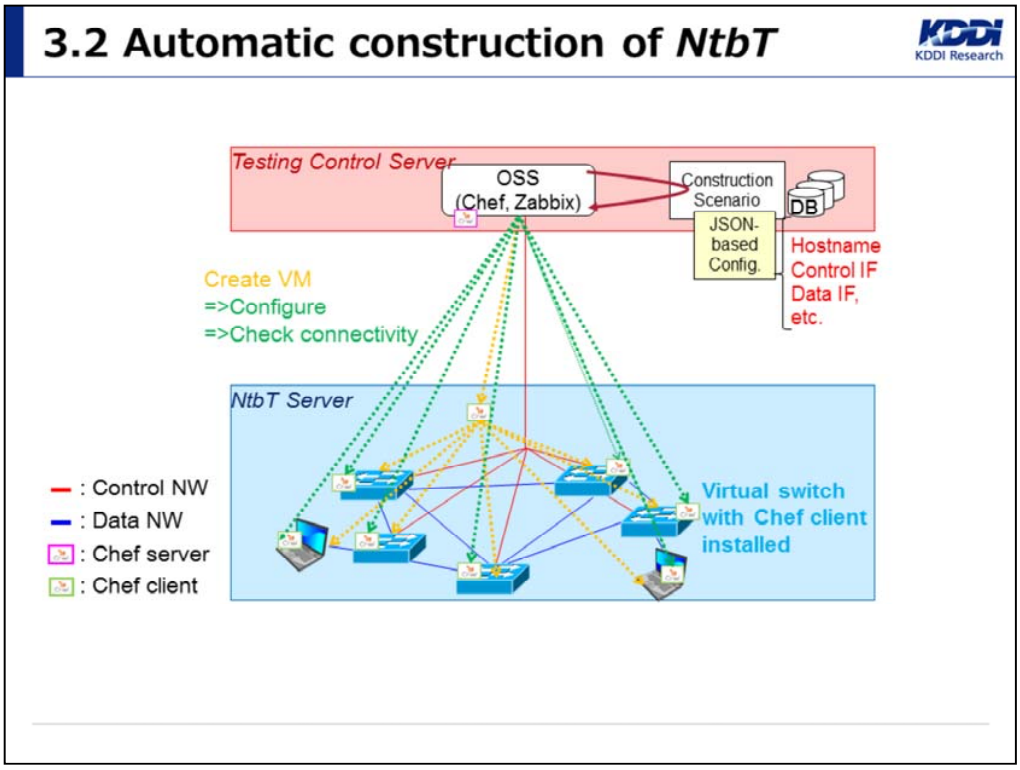
3.1 Testing scenario

This slide shows a testing scenario created by operators on a Web browser for automatic testing. A testing scenario consists of an NtbT construction scenario and a failure scenario.

The construction scenario has four scenario steps. One is selection of NtbT topology which can be selected from DB and determines an NtbT topology. The second step is construction of NtbT-creating virtual machines and bridges based on the selected topology. Third is initialization of network equipment which configures the virtual machines. And, the last one is checking of connectivity by executing a ping test.

The failure scenario consists of traffic recipes and failure recipes. Recipes can be selected from the DB with execution time and other parameters such as IP version and bandwidth for a traffic recipe and node and interface where a failure is caused for a failure recipe. Examples of failure recipes are interface down/up and latency start/stop.

In using this system, the procedure is as follows. Firstly, operators create a testing scenario with a start time. Secondly, the scenario is started automatically at the specified time and runs until the failure scenario is completed. Finally, operators check the results on a browser.



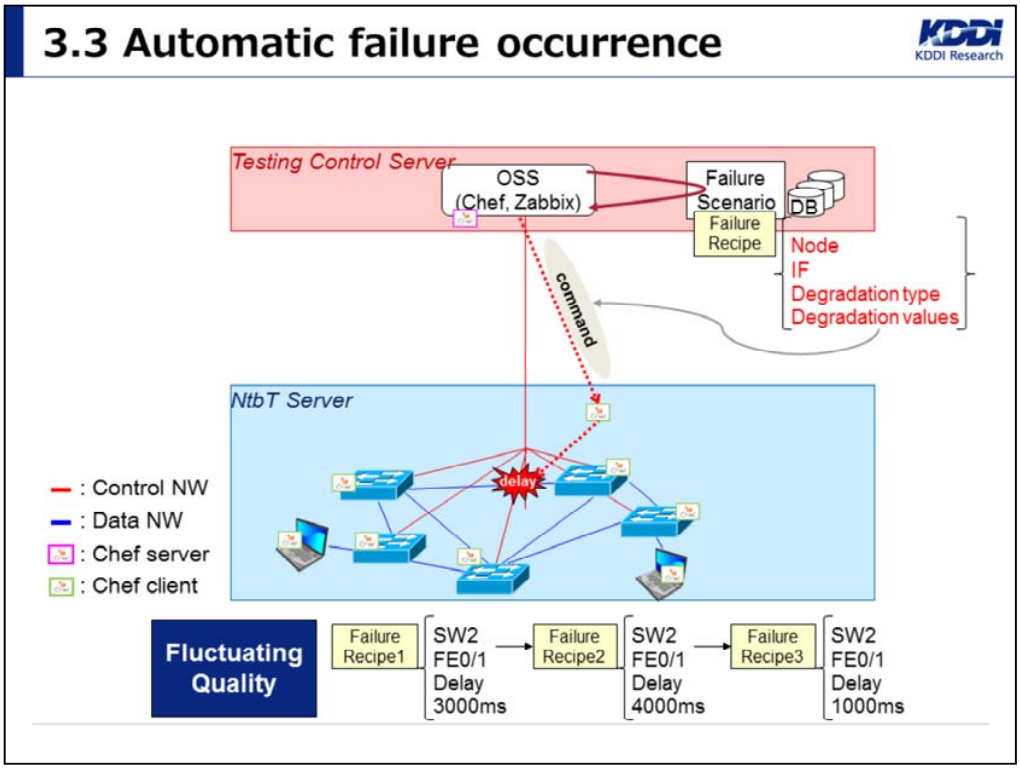
3.2 Automatic construction of NtbT

The Chef, an infrastructure environment construction tool, constructs the NtbT based on the configuration files of the topology selected from the DB in a construction scenario. The files are written in the JSON language and they define the configuration of each node, including a hostname, a control interface connected to the testing control plane, and data interfaces connected to the data plane. Each interface is defined with an IP address, a virtual bridge, and other information such as routing.

As the first step of automatic construction, the Chef server on the testing control server instructs the Chef client on the NtbT server to create virtual machines based on the information of a hostname and a control interface in the configuration file. We use Cumulus [7] as a network OS to build a virtual switch where a Chef client has been installed for the next step.

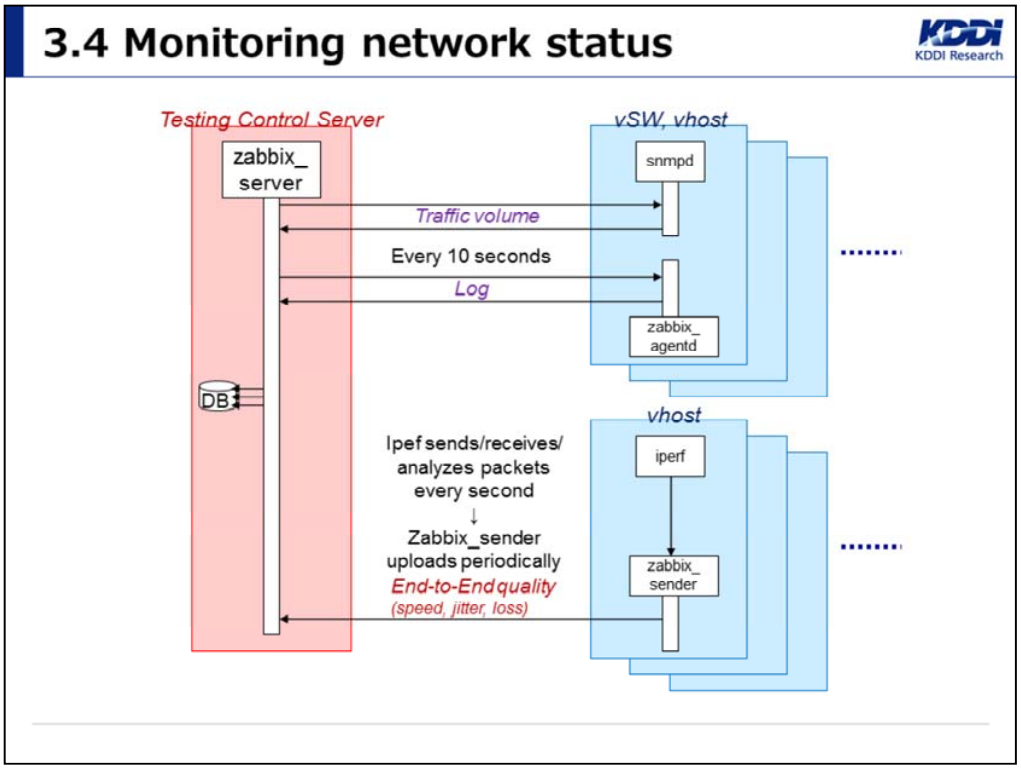
Secondly, the Chef server instructs the Chef client on each node to configure the IP address, routing. etc. that are also defined in the configuration file.

Once construction is completed, a ping test is performed on every link and between every host pair. The instructions of the ping test are also sent from the Chef server to the Chef client on each node.



3.3 Automatic failure occurrence

The Chef server on the testing control server reads failure recipes in a failure scenario, and executes the corresponding command in sequential order. The command is created based on the failure recipe that defined the failure location, such as node and interface, degradation type such as latency and packet loss, and degradation values. The created command is sent from the Chef server to the Chef client on the NtbT server and executed to cause the degradation. Eventually, it enables network testing with fluctuating quality through a combination of several failure recipes having different parameters.



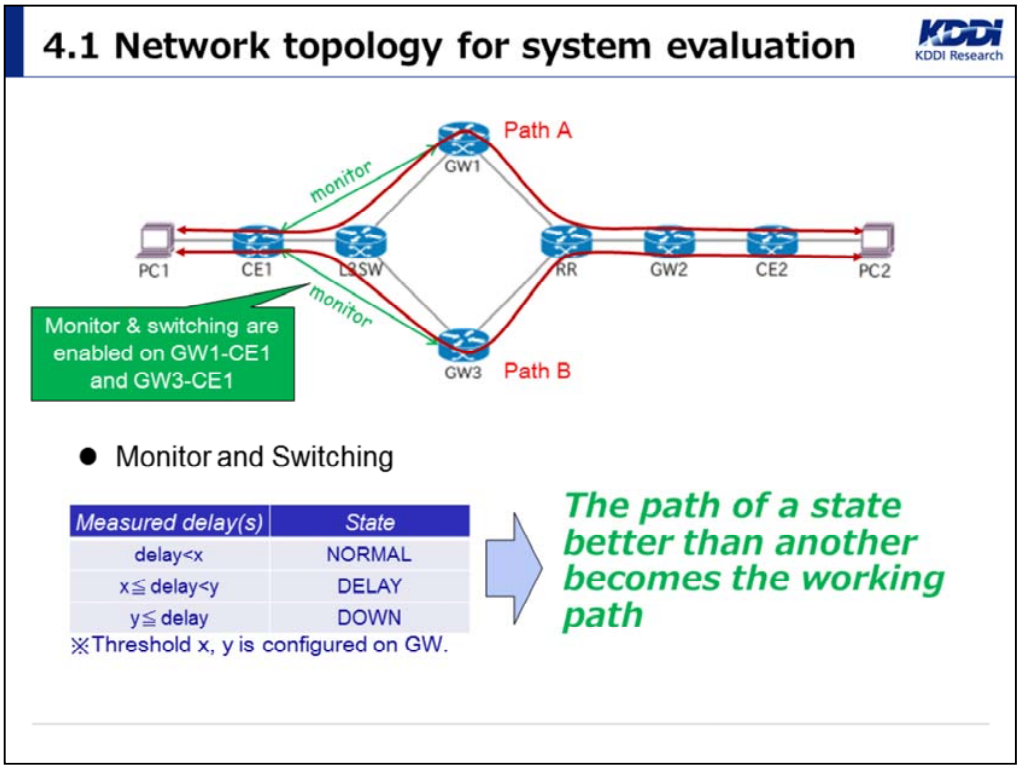
3.4 Monitoring network status

This slide explains how the testing control server monitors each node in the NtbT during network testing for verification. Zabbix, a network monitoring tool, periodically collects network information such as traffic volume on each interface, logs of the devices and the End-to-End communication quality of every host pair.

Every 10 seconds, the Zabbix server on the control server sends an inquiry for traffic volume to snmpd and that for logs to zabbix_agentd, and each daemon replies accordingly.

Iperf [8] and zabbix_sender on each host are used to collect the information of End-to-End quality. Iperf sends, receives and analyzes packets between every host pair and measures the End-to-End quality, such as speed, jitter and packet loss, every second. Zabbix_sender periodically uploads the measured data to zabbix_server.

The collected data are stored in the DB on the control server, and operators are able to easily confirm the test results via the browser.

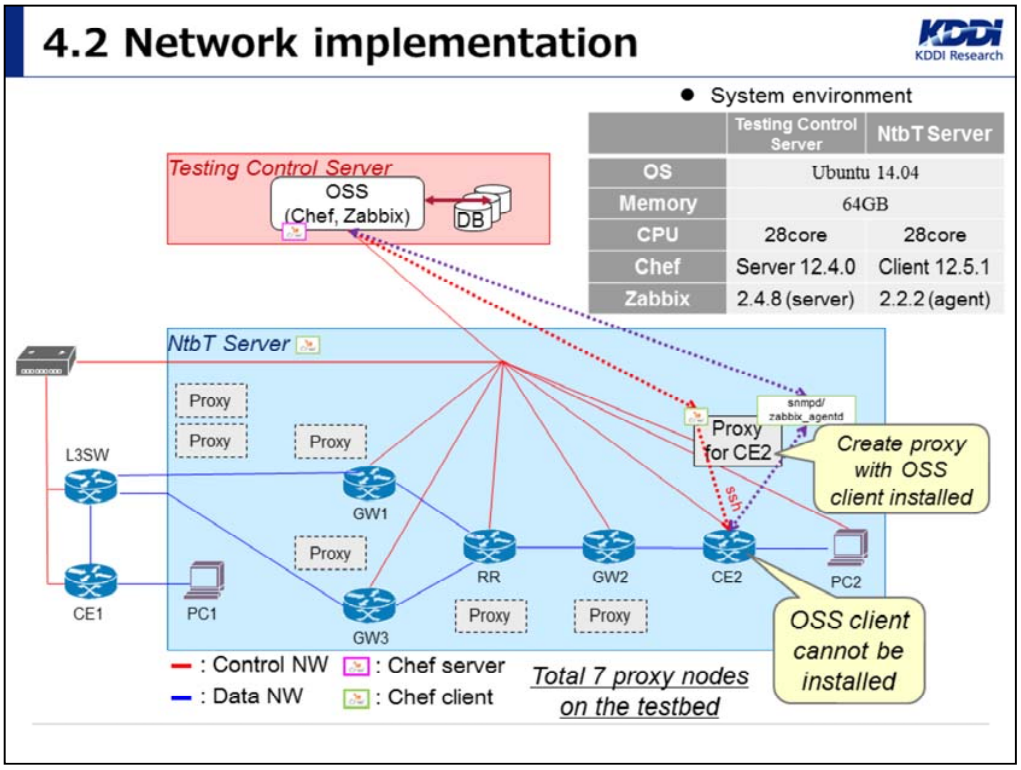


4. System evaluation

4.1 Network topology

To evaluate the availability of our proposed system, we set up a network testbed consisting of IP-Sec Gateway (GW) as a example of the target network, which has the function of policy-based path control. Both GW1 and GW3 monitor each link between GW and the Customer Edge router (CE) and they switch paths by comparing the measured delay values against the thresholds in the configuration.

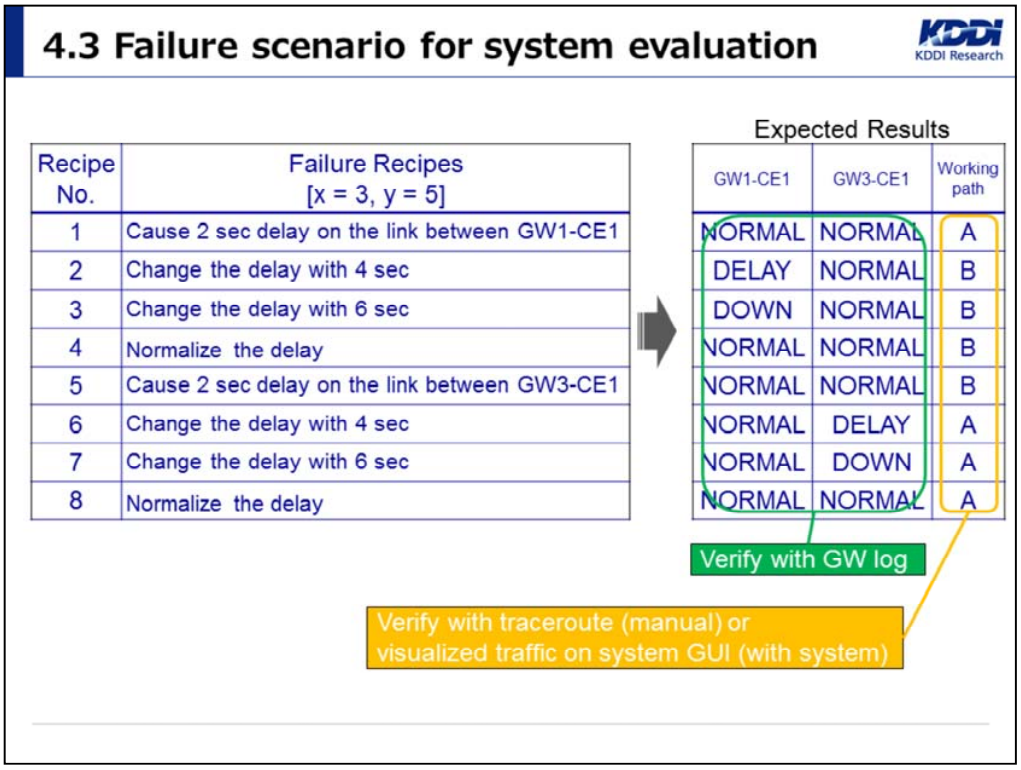
The state of the link is defined as either NORMAL, DELAY or DOWN with two thresholds x and y configured on GW1 and GW3. When the measured delay is less than x, the state is NORMAL. When the delay is x or longer and less than y, the state is DELAY. When the delay is y or longer, the state is defined as DOWN. The path of a state which is better than another is selected as the working path.



4.2 Network implementation

The picture above shows the implementation of the experimental environment. The table of system environment summarizes the specifications of servers for testing control server and NtbT server.

Chef and Zabbix server are installed on the testing control server, and Chef client is installed on the NtbT server. On the other hand, the appliance-based devices in NtbT such as “CE2” cannot install OSS client, therefore Linux containers (LXC) with Chef and Zabbix client installed are created and they work as a proxy node to receive a packet for the appliance-based devices and take necessary actions. When the Chef server on the testing control server sends an instruction to a device, the proxy node for that device receives it and executes the corresponding command on the actual device via ssh connection. When the Zabbix server on the control server sends an inquiry for a device, the proxy node for that device receives and transfers it to the actual device via Network Address Translation (NAT) to obtain network information such as traffic volume and log. The reverse data is also transferred to the control server by the proxy.

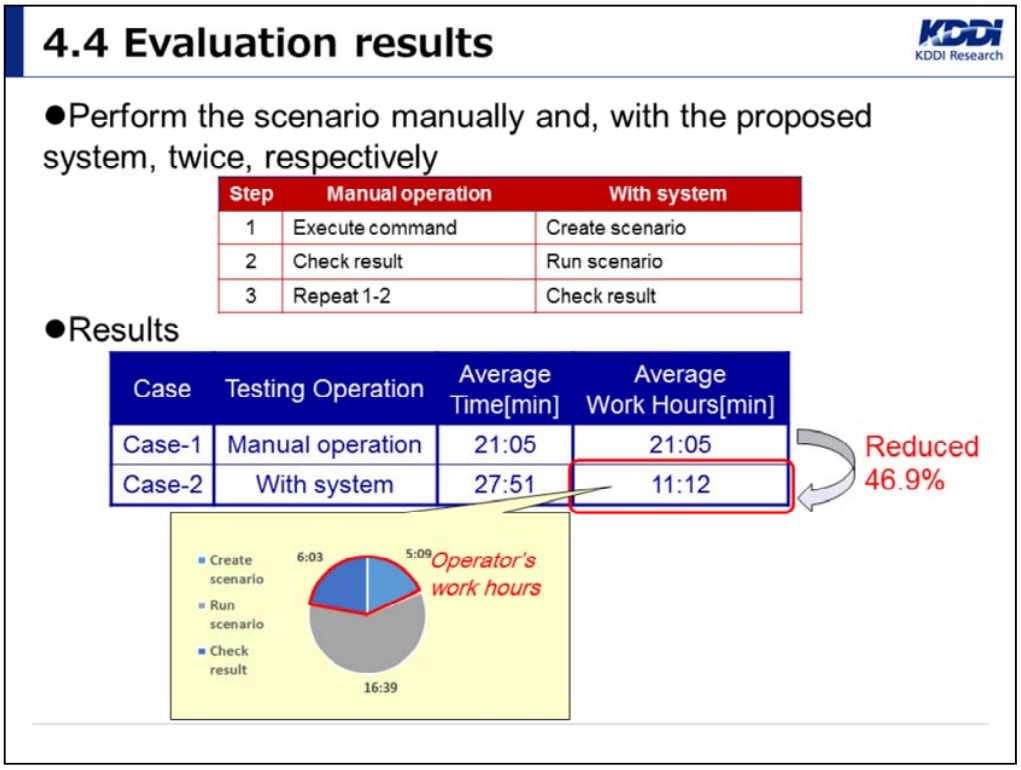


4.3 Failure scenario

The failure scenario used for system evaluation is as follows.

1) Cause a 2 second delay on the link between GW1-CE1
2) Change the delay to 4 seconds
3) Change the delay to 6 seconds
4) Normalize the delay
5) Cause a 2 second delay on the link between GW3-CE1
6) Change the delay to 4 seconds
7) Change the delay to 6 seconds
8) Normalize the delay

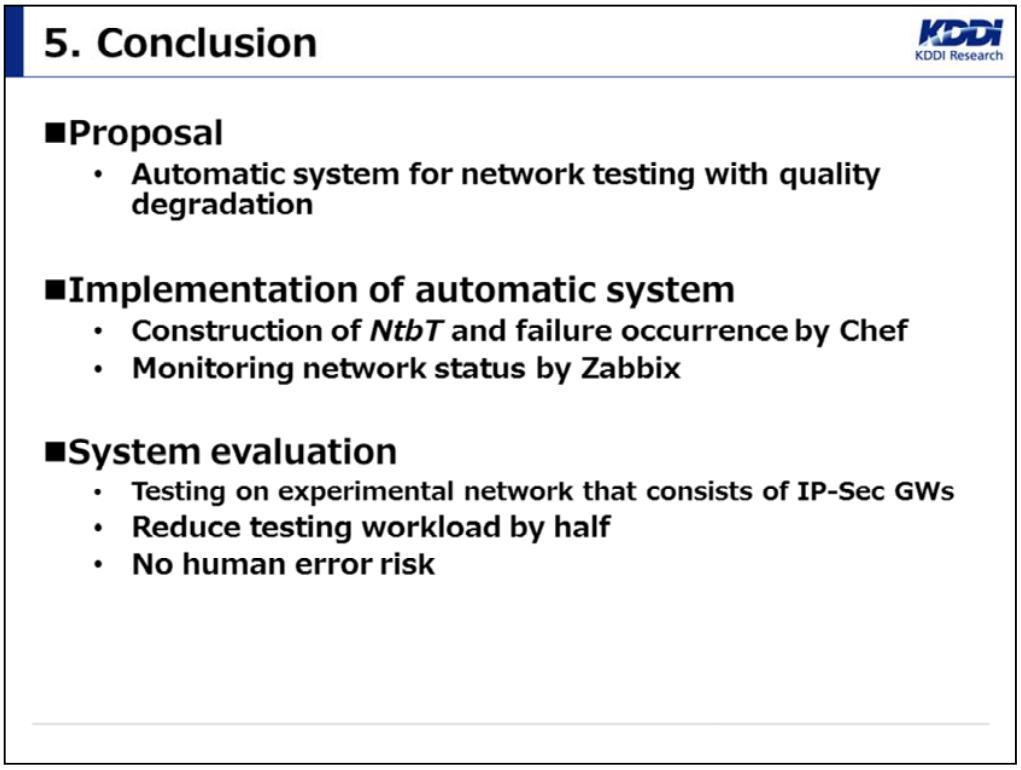
Expected results are described in the table at right. When the caused delay is four seconds, the state should be DELAY; in the case of six seconds, it should be DOWN. The working path should change and the traffic should be rerouted when GW1 becomes DELAY while GW3 is NORMAL and when GW3 becomes DELAY while GW1 is NORMAL. In verification, the states of GW1-CE1 and GW3-CE1 are confirmed on the GW log. The working path is confirmed by a traceroute command in the case of manual operation or by visualized traffic on GUI in the case of system operation.



4.4 Evaluation results

We evaluated the proposed system using the failure scenario described in the previous slide. We performed the scenario manually and with the proposed system, twice, respectively, and measured the time taken to complete all steps and the operator's work hours. In the case of manual operation, the time includes that for executing commands and checking results, and the work hours are the same since the operator needs to work continuously. In the case of system operation, the time includes that for creating the scenario, running the scenario, and checking results, and the work hours include the time taken in creating the scenario and checking the scenario (excluding that for running the scenario because it runs automatically).

In case-1, the average time and the average work hours were 21:05 minutes. On the other hand, in case-2, the average time was 27:51 minutes, and the average work hours were 11:12 minutes. The results showed that our proposed system reduced the testing workload by 46.9% compared to a manual-based approach.



5. Conclusion

We proposed an automatic system for network testing with quality degradation. We implemented the proposed system using OSS, Chef and Zabbix. Chef supports automatic construction of NtbT and failure occurrence, and Zabbix supports periodic monitoring of network status periodically during testing. We evaluated our proposed system using an experimental network consisting of IP-Sec GWs, and the demonstration results showed that the proposed system reduced testing workload by half. In addition, there was no human error risk in testing using the system, which is another advantage of our proposal.

References

[1] Silver Peak White Paper, “Dynamic Path Control: The Foundation for Your Hybrid WAN,” https://www.silver-peak.com/sites/default/files/infoctr/silver-peak_wp_dpc.pdf.

[2] Broadband Forum TECHNICAL REPORT, “TR-348 Hybrid Access Broadband Network Architecture,” https://www.broadband-forum.org/technical/download/TR-348.pdf.

[3] M. Shibuya, H. Kawakami, T. Hasegawa, and H. Yamaguchi, “Design and implementation of support system for network testing with whitebox switches,” COLLA 2016, pp.39-44, Nov. 2016.

[4] QualiSystems CloudShell / TestShell, http://www.qualisystems.com/products/cloudshell-add-ons/testshell-overview/

[5] Chef, https://www.chef.io/chef

[6] Zabbix, http://www.zabbix.com

[7] Cumulus, https://cumulusnetworks.com

[8] Iperf, https://iperf.fr


KDDI
KDDI Research