Automated Performance Evaluation of Intent-based Virtual Network Systems

Kazuki Tanabe*, Tatsuya Fukuda* and Takayuki Kuroda*
*NEC Corporation
1753, Shimonumabe, Nakahara-ku, Kawasaki, Kanagawa 211–8666, Japan
Email: {kazuki-tanabe, t_fukuda1987, kuroda}@nec.com

***Abstract*—We propose an automation method for the testing phase of system integration. The method automatically generates an evaluation program for determining if the target system design satisfies the system requirements. We model both system requirements and system design as graph structures and define the test templates that contain abstract test commands and search methods of their syntax parameters. After referring to the test templates and acquiring parameters from the system design graph according to search methods, test commands are translated into concrete test scripts, which are compatible for different environments such as OS type and network configuration, by referring to the command templates. We conducted evaluation experiments on the effectiveness of our method. The results indicate that the method can flexibly generate an evaluation program in a short time and the evaluation program can determine the satisfaction of system requirements for the deployed system design based on their evaluation units.**

I. Introduction

According to the recent rapid growth in virtualization technologies of network infrastructures, the demand for ICT system integration (SI) based on virtualization technologies has been increasing. On one hand, these technologies are applicable to various scales of network environments, and enable easier and more flexible configuration and deployment for clients. On the other hand, such technologies may put a heavier burden of engineers and lead to a network configuration depended on the engineers' specific proficiency because management of such systems requires a wide range of advanced knowledge.

To reduce the duration of system designing and configuration, there have been several studies on automation technologies for each phase of SI. These studies have mainly focused on the requirement-definition, designing and deployment phases. We developed an automated network configuration designer called Weaver [1] and an automated provisioning planner and deployment tool [2]. Weaver follows the concept of Intent-based Networking (IBN), in which network management is based on users' abstract intent for output ICT systems. Meanwhile, for application-level testing phase, automation technologies have been actively studied and several CI/CD testing tools such as Jenkins [3] have been developed. However, there have been few studies of network-level system-test automation technology.

For application-level automation of the testing phase, there have been studies of Web frameworks targeted at Web-oriented services [4] [5]. Wang et al. proposed an automation framework to execute input test scenarios and determine the testing results [4]. Guo et al. proposed a test automation web framework [5] in which performance tests of web applications are automated by referring to test scenarios, which are defined in XML-based Web Services Description Language (WSDL) format.

In the general testing phase of SI, system requirements of clients are divided into several evaluation units, which generally correspond to clients' intents in the SI for IBN-based systems. To determine that the deployed network system satisfies the requirements, evaluation programs are created and executed for each evaluation unit. When these programs are created from abstract requirements on the IBN-based network environment, both the system requirements and deployed network topology (system design) need to be referred and some commands of the evaluation program need to be modified according to the system design (e.g. available packages on each OS and arguments of commands, such as IP address or URL). Therefore, different evaluation programs need to be created for each system design, which puts a heavy burden on engineers.

Net-Tester [6] is a conventional opensource tool for automating the network-level testing phase. It is based on Cucumber [7], which is a framework of Behavior Driven Develop (BDD), in which test scenarios are described with the behavior of the system in a natural language. However, it still has some room for improvement from the viewpoint of reusability because some information, such as IP addresses and port numbers, still needs to be manually described at the test scenario. Li et al. have had a survey on network verification and testing methods by using formal methods in Software Defined Networking (SDN) [8]. They concluded that the verification of Boolean invariants e.g. the reachability on Data plane (D-plane) is fast enough for applying to large networks [9] [10] [11], but the verification of quantitative invariants, e.g. latency, packet loss, and bandwidth is still a challenge for future work.

In this study, we propose an automation method of the network-level testing phase of SI for IBN-based ICT systems. Our method automatically generates evaluation programs from system requirements, which consist of clients' abstract intents, and a system design derived from the requirements. The target systems are modeled as graph structures, and the method compares the system requirements and system design and searches the parameters of the evaluation program in the system design graph. Thus, it enables flexible generation of appropriate evaluation programs according to each client's intent.

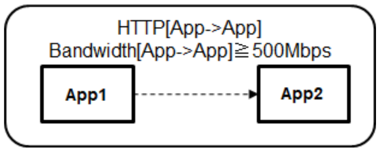

Fig. 1. Example of system requirement graph G^{req}.

```
HTTP[App->App]:
  src: App1
  dest: App2
Bandwidth[App->App]≥500Mbps:
  src: App1
  dest: App2
```

Fig. 2. Example description of G^{req}.

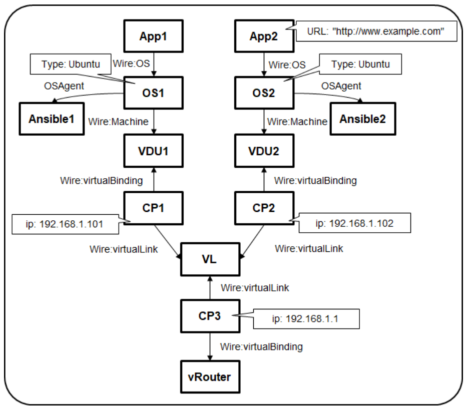

Fig. 3. Example of system design graph G.

```
App1:
  type: App
  requirements:
    OS: [OS1, Wire:OS]
App2:
  type: App
  properties:
    URL: http://www.example.com/
  requirements:
    OS: [OS2, Wire:OS]
OS1:
  type: OS
  properties:
    Type: Ubuntu
  requirements:
    Ansible: [Ansible, OSAgent]
    Machine: [Machine1, Wire:Machine]
...
```

Fig. 4. Example description of G.

The rest of this paper is organized as follows. In Section II, we describe the model definition of our target ICT systems and purpose of this study. In Section III, we explain our proposed method, which is based on a graph-based parameter-search algorithm. In Section IV, we present the results of an evaluation experiment on the effectiveness of our proposed method. We conclude this paper and discuss future work in Section V.

II. Model Settings

In this section, we give the model definition of our target ICT systems and explain the purpose of this study.

The network topology and configuration of an ICT system (system design) is modeled as a directed graph $G = (V, E)$. The G is composed of a set of nodes V and set of edges E. Set V consists of system element v_i, which includes not only network appliances, such as routers, switches, and firewalls, but also physical machines (PMs), virtual machines (VMs), OSes running on machines, middleware, and applications running on the OSes. Set E consists of a connection $e_{i,j} = \{v_i, v_j\}$ from a node $v_i \in V$ to another node $v_j \in V$. In addition, each v_i consists of properties such as the configuration value of applications, IP address of a network interface, and OS type.

System requirements consist of abstract intents that the client requires for the ICT system. These intents are composed of functional requirements and non-functional requirements, e.g., reliability, performance, and security. We also model the system requirements as a directed graph $G^{req} = (V^{req}, E^{req})$. The G^{req} is composed of a set of nodes V^{req} and set of edges E^{req}. Set V^{req} consists of a system element v_i^{req}, and E^{req} consists of a connection $e_{i,j}^{req} = \{v_i^{req}, v_j^{req}\}$ from a node v_i^{req} to another node v_j^{req}.

Examples of G^{req} and G are shown in Figs. 1 and 3, respectively. System requirements consist of an HTTP connection from a client application (App1) to a server application (App2) and minimum bandwidth of 500 Mbps. A system design is a concrete network configuration on an Openstack [12] virtual environment and is automatically generated from the system requirements in Fig. 1 by using Weaver. App1 and App2 are hosted on different Ubuntu Linux OSes on independent VMs (Virtual Development Units: VDUs) respectively. Each VDU connects to the same virtual subnet (Virtual Link: VL), which is operated by a virtual router (vRouter) via each virtual network interface (Connection Point: CP). Ansible provisioning tools [13] have also been installed on both Ubuntu OSes and App2 contains the Web URL of the website as a property.

Finally, the definitions of G^{req} and G are shown in Figs. 2 and 4. The definition of G^{req} describes the elements and connections that are required on the ICT system. For example, the G^{req} in Fig. 1 contain two requirements as items, and each item defines the source (`src`) and destination (`dest`) of the connection. The definition of the G is based on the TOSCA format [14], which is a standard by OASIS for defining ICT system configurations. The G is described as a list of nodes v_i that contains supplemental information as their properties. As shown in Fig. 4, the `type` field defines the type of each node and `properties` field defines the properties of each node. The `requirements` field defines the ID of node instance that can be connected from the node and type of connection.

We model target network systems as graph structures with the above settings. Our goal is to automate the network-level testing phase of SI by automatically generating evaluation programs that determine the G's satisfaction of G^{req}, from abstract input intents. To automate evaluation-program generation, an evaluation script needs to be generated for each evaluation unit, and the agents and parameters of the evaluation script need to be determined by referring to the relationship between the system requirements and the system design. Our proposed method searches and acquires the required agents and parameters in the G for each evaluation unit, as discussed in the next section.

III. Proposed Method

In this section, we explain our proposed method to generate evaluation programs for the network-level testing phase.

First, an evaluation program is composed of several evaluation scripts, each of which refers to each evaluation unit. The method requires two inputs: the G^{req} and a G, which is derived from the G^{req}. The method then refers to the evaluation template, which is defined for each evaluation unit in advance, and generates an abstract evaluation command related to each evaluation script.

Table I shows the definition example of evaluation templates. Each evaluation unit has two types of arguments: type of target

TABLE I
DEFINITION EXAMPLE OF EVALUATION TEMPLATES

Evaluation Unit		Value	Search Method
`HTTP[<a:App> -> <b:App>]`	Command	`Run http(Agent, OS.Type, App2.URL)`	
	Parameter	`Agent`	`<a>(reference, HostedOn, OS)+(service, OSAgent, Agent)`
		`OS.Type`	`<a>(reference, HostedOn, OS)`
		`App2.URL`	`<b>(reference, HostedOn, App)`
`Bandwidth[<a:App> -> <b:App>, x[Mbps]]`	Command	`Run bandwidth(Agent1, OS1.Type, Agent2, OS2.Type, CP2.ip, x)`	
	Parameter	`Agent1`	`<a>(reference, HostedOn, OS)+(service, OSAgent, Agent)`
		`OS1.Type`	`<a>(reference, HostedOn, OS)`
		`Agent2`	`<b>(reference, HostedOn, OS)+(service, OSAgent, Agent)`
		`OS2.Type`	`<b>(reference, HostedOn, OS)`
		`CP2.ip`	`<b>(reference, HostedOn, VDU|Container)+(service, HostedOn, CP)`

TABLE II
DEFINITION EXAMPLE OF COMMAND TEMPLATES

Commands	Order	Conditions	Agents	Scripts
`http(Agent, OS.Type, URL)`	1	`OS.Type == "Windows"`	`Agent`	`Invoke-Restmethod -url {{URL}}`
		`OS.Type == "Ubuntu"`	`Agent`	`curl {{URL}}`
`bandwidth(Agent1, OS1.Type, Agent2, OS2.Type, CP2.ip, x)`	1	`OS2.Type == "Windows"`	`Agent2`	`iperf -sD`
		`OS2.Type == "Ubuntu"`	`Agent2`	`iperf -sD`
	2	`OS1.Type == "Windows"`	`Agent1`	`iperf -c {{CP2.ip}}`
		`OS1.Type == "Ubuntu"`	`Agent1`	`iperf -c {{CP2.ip}}`

TABLE III
DESCRIPTION EXAMPLE OF EVALUATION PROGRAM

Evaluation Units	Order	Agents	Scripts	Constraints
`HTTP[App->App]`	1	`Ansible1`	`curl http://www.example.jp/`	
`Bandwidth[App->App]`$\geq$`500Mbps`	1	`Ansible2`	`iperf -sD`	
	2	`Ansible1`	`iperf -c 192.168.1.102`	`bandwidth` $\geq$ `500[Mbps]`

nodes in the system design and constraints for determining the evaluation results. Each evaluation template is mapped to each evaluation unit and consists of an abstract evaluation command, parameters of the command, and search methods for acquiring parameters in the G. Each search method consists of the starting node and search steps listed in execution order. Each search step is defined as a tuple of three arguments: search direction of edge $e_{i,j} \in E$ in the G (reference: forward, service: backward), type of traversable edges, and type of destination node.

The parameter acquisition algorithm of the proposed method is shown in Algorithm 1, and Fig. 5 shows an example of search procedure when Algorithm 1 is applied to the G. The inputs of Algorithm 1 are the G, starting node v_{start}, and an array of search steps $steps$. Each search step $step_i \in steps$ is a tuple of search direction $relation_i$, type of traversable edges $wireType_i$, and type of destination node $nodeType_i$. Parameter search is executed as a recursive call of function `Search`, and edge traversal is conducted for every edge the type or the parent edge's type of which is equal to $wireType_i$ and which is connected to the current node $v_{current}$ in $relation_i$. Search step $step_i$ is repeated until the destination node, the type of which is $nodeType_i$, is found. If the destination node is found at the final search step $step_{n-1}$ and has the parameter $param$ in the `properties` field, Algorithm 1 returns the $param$ value as the output value. Otherwise, it returns $None$, which means that the search failed. Function `FindEdge` is a subprocess function used in function `Search` and determines the existence of an edge $e_{i,j} = \{v_i, v_j\}$ whose source node v_i, destination node v_j, type of edge $e_{i,j}$ or its parent edge is equal to src, $dest$, and $wireType$, respectively.

Fig. 5 shows the procedure of searching for and acquiring an IP address required to access the Web application in a system design G. First, the search process starts at App1. In the first search step $\{relation_0$= `reference`, $wireType_0$= `HostedOn`, $nodeType_0$= `VDU`$\}$, edges $\{$`App1, OS1`$\}$ and $\{$`OS1, VDU1`$\}$ are traversed in the forward direction. The types of these two edges(`Wire:OS` and `Wire:Machine`) are derived from abstract edge type `HostedOn`, which means a hosting relationship. In the final step $\{relation_1$ `service`, $wireType_1$= `HostedOn`, $nodeType_1$= `CP`$\}$, an edge $\{$`CP1, VDU1`$\}$, whose type `Wire:virtualBinding` is also derived from `HostedOn`, is traversed in backward direction. Finally, destination node `CP1`, whose node type is `CP`, is found and function `Search` returns the IP address of `CP1`.

Second, evaluation commands the parameters of which are acquired from Algorithm 1 are translated into concrete and executable evaluation scripts by referring to command templates. Table II shows a example definition of command templates. Each command template defines mappings from an evaluation command to ordered tuples of evaluation scripts and their agents. Each tuple of a script and agent has conditional branches based on the input parameters. When a command template is referred by an evaluation command with parameters, these tuples are extracted in execution order, according to parameter conditions. Extracted tuples becomes the output evaluation script. Each example script in Table II is an essential part of a whole evaluation script for simplicity, while the actual evaluation scripts include pre-evaluation scripts and post-evaluation scripts. For example, installation commands of packages are executed before the evaluation, and termination commands of evaluation tasks or uninstallation of temporary packages are executed after the evaluation.

Finally, an evaluation program generated from the above procedure and shown in Table III is defined as a list of

Algorithm 1 Search for parameter *param*

```
Input: $G = (V, E), v_{start} \in V, steps = [step_0, \ldots, step_{n-1}]$
    $(step_i = \{relation_i, wireType_i, nodeType_i\})$
Output: Value($param$)
 1: $i \leftarrow 0$
 2: $v_{current} \leftarrow v_{start}$
 3: function SEARCH($v_{current}, i, steps, param$)
 4:     if Type($v_{current}$) == $nodeType_i$ then
 5:         if $i == n - 1$ then
 6:             if $param$ in Properties($v_{current}$) then
 7:                 return Value($param$)
 8:             return $None$
 9:             end if
10:         else
11:             $i \leftarrow i + 1$
12:         end if
13:     end if
14:     if $relation_i$ is 'reference' then
15:         for all $v_{dest(j)}$ s.t. FindEdge($v_{current}, v_{dest(j)}, wireType_i$) is True do
16:             return Search($v_{dest(j)}, i, steps, param$)
17:         end for
18:     else if $relation_i$ is 'service' then
19:         for all $v_{src(j)}$ s.t. FindEdge($v_{src(j)}, v_{current}, wireType_i$) is True do
20:             return Search($v_{src(j)}, i, steps, param$)
21:         end for
22:     end if
23: end function
24: function FINDEDGE($src, dest, wireType$)
25:     for all $e_{i,j} = \{v_i, v_j\} \in E$ do
26:         if $v_i == src$ and $v_j == dest$ and (ParentType($e_{i,j}$) == $wireType$ or Type($e_{i,j}$) == $wireType$) then return true
27:         end if
28:     end for
29:     return false
30: end function
```

evaluation units and has ordered tuples of agents, scripts, and constraints. Constraints to determine the evaluation results are described in the tuple that contains the actual evaluation task, e.g., connection test or performance measurement.

For example, to evaluate the evaluation unit `HTTP[App->App]`, which confirms an HTTP connection between a client and server, an evaluation script is executed by agent `Ansible1` hosted on OS `Ubuntu1`. In the evaluation script, an HTTP request is sent from `Ubuntu1` to the URL of the Website served by the server application `App2`, by the `curl` command executed by `Ansible1`. The evaluation result is determined by parsing the response body from `App2`.

For the evaluation unit `Bandwidth[App->App]` $\geq$`500Mbps`, which measures the communication bandwidth between two applications, the evaluation result is determined by the input constraint (minimum bandwidth: 500 Mbps). First, the bandwidth-measurement tool `iperf` is executed in server mode by `Ansible2`, on Ubuntu node `Ubuntu2`, which runs the destination application `App2`. Second, `iperf` is executed in client mode on Ubuntu node `Ubuntu1` by `Ansible2`, with an IP address of a virtual network interface card (NIC) connected to `Ubuntu2` as a parameter. Finally, the measured bandwidth is extracted from the output of the `iperf` client mode, and the evaluation result is determined in comparison with the constraint value of 500 Mbps.

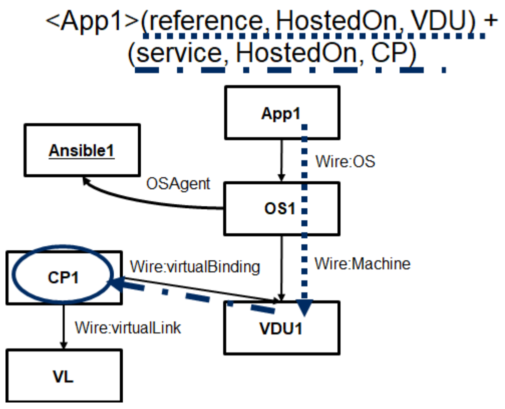

Fig. 5. Example of parameter search procedure.

IV. Evaluation

We have conducted evaluation experiments according to the model settings described in Section II. We implemented the proposed method and its algorithm, and associated them with Weaver, our provisioning planner, and deployment tool so that a major part of IBN-based SI, from designing phase to testing phase, can be automated in sequence. Weaver generates a system design from the input system requirements, and the parameters acquired from Algorithm 1 are combined with the definition of system design into an expanded TOSCA format. An evaluation program is generated as a workflow from the expanded TOSCA format by our provisioning planner and executed by our deployment tool. The result indicates the advantages of our method: flexible generation of evaluation programs and time reduction in testing phase.

A. Parameter Search

We first conducted an experiment for Algorithm 1. With Algorithm 1, parameters of evaluation programs are acquired in the system design and abstract intents can be concretized into actual evaluation scripts. In this experiment, we applied Algorithm 1 for several situations in the same system design and found that different parameters can be correctly acquired.

Figs. 6 and 7 show the structure of the target system requirements G^{req} and system design G. These graphs are expansions of the ones shown in Figs. 1–4. In addition to two applications `Client1` and `Server1`, another client application `Client2` and server application `Server2` are required to join the independent docker infrastructure. The system requirements also include two evaluation units `HTTP[App->App]` and `Bandwidth[App->App]`$\geq$`500Mbps` for each connection from a client application to server application. In the system design in Fig. 7, docker is running on both `OS1` and `OS2`. Each docker container runs Red Had Enterprise Linux (RHEL) as a base OS and hosts another client application `Client2` and server application `Server2`, respectively. Each docker container is connected to the same `VL` subnet via the virtual interface `macvlan` of each host OS.

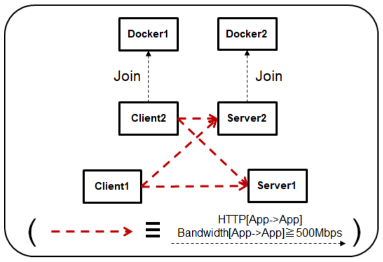

Fig. 6. System requirements G^{req} including docker containers.

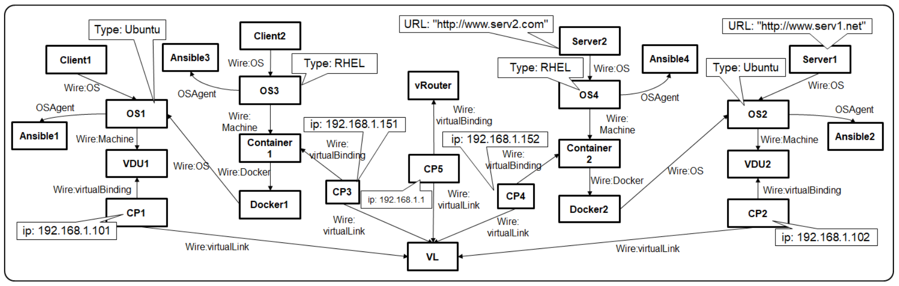

Fig. 7. System design G including docker containers.

TABLE IV
RESULTS OF PARAMETER SEARCHES

src	dest	http(Agent, OS.Type, App2.URL)			bandwidth(Agent1, OS1.Type, Agent2, OS2.Type, CP2.ip, x)				
		Agent	OS.Type	App2.URL	Agent1	OS1.Type	Agent2	OS2.Type	CP2.ip
client1	server1	Ansible1	Ubuntu	http://www.serv1.net	Ansible1	Ubuntu	Ansible2	Ubuntu	192.168.1.102
client1	server2	Ansible1	Ubuntu	http://www.serv2.com	Ansible1	Ubuntu	Ansible4	RHEL	192.168.1.152
client2	server1	Ansible3	RHEL	http://www.serv1.net	Ansible3	RHEL	Ansible2	Ubuntu	192.168.1.102
client2	server2	Ansible3	RHEL	http://www.serv2.com	Ansible3	RHEL	Ansible4	RHEL	192.168.1.152

In this network topology, we executed Algorithm 1 for every connection pattern. Table IV shows the results of parameter search for evaluation units `HTTP[App->App]` and `Bandwidth[App->App]≥500Mbps`. Algorithm 1 acquired the correct parameters of evaluation commands for every connection pattern. For example, target URL of the evaluation command `http(Agent, OS.Type, APP2.URL)` was correctly obtained from the target Web server application given in `dest`, and the target IP address of the evaluation command `bandwidth(Agent1, OS1.Type, Agent2, OS2.Type, CP2.ip, x)` was also correctly obtained from the virtual NIC connected to the VM or docker container, which runs the target Web server. As these results indicate, search methods for each parameter defined in evaluation templates enable flexible evaluation-program generation.

B. Results of Evaluation Programs

We deployed the G shown in Figs. 3 and 4 on Openstack virtual environments and executed the evaluation program generated with the proposed method on these network environments. The evaluation program determines if the G satisfies the G^{req} in Figs. 1 and 2, including the following two evaluation units:

- HTTP connection from `App1` to `App2` can be made
- Communication bandwidth between applications `App1` and `App2` is more than or equal to 500 Mbps

Fig. 8 shows the deployed network environment with the relationship of connections and agents on the evaluation program. In the evaluation environment 1, two VMs (`VDU1`, `VDU2`) are running, and each VM has a single-core single-thread CPU and 1 GBytes of virtual RAM. Each VM also runs Ubuntu OS provisioned by Ansible. The `Ubuntu1` OS runs a client application corresponding to `App1` and `ubuntu2` runs nginx to host a website, as an example of `App2`. In addition, each VM is connected to the same virtual subnet (`VL`) via its virtual NIC (`CP`), and a virtual router (`vRouter`) runs on the subnet.

Table V list the execution results of the evaluation program on evaluation environment 1. Evaluation unit `HTTP[App->App]` was determined to be `PASS` because a normal HTTP response from nginx was retuned as the program output, which means successful HTTP connection from `Ubuntu1` to the website. The evaluation unit `Bandwidth[App->App]≥500Mbps` was also determined to be `PASS` because the `iperf` client returned the measured bandwidth as 818 Mbps, which satisfies the input constraint for the minimum bandwidth of 500 Mbps.

We added two VMs on the evaluation environment 2 as shown in Fig. 9, to generate background traffic on the virtual subnet `VL`. Background traffic is continuously generated from these two VMs to the `ubuntu2` OS by large file transfer of the `scp` command. We executed the same evaluation program in this situation. Table VI lists the evaluation results. Evaluation unit `HTTP[App->App]` had the same result as in evaluation environment 1 and was determined to be `PASS`, while evaluation unit `Bandwidth [App->App]≥ 500Mbps` was determined to be `FAIL` because the `iperf` client returned the measured bandwidth as 306 Mbps, which is below the constraint for the minimum bandwidth. These results are due to the heavy traffic load of background traffic and shortage in computing resources of the `ubuntu2` node because of receiving large files.

These results indicates that the evaluation program automates the network-level testing phase in a target ICT system based on evaluation units that consists of the client's system

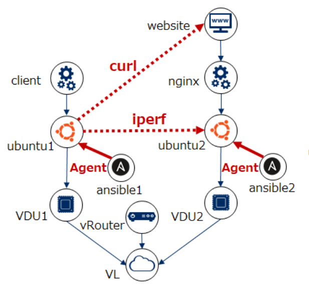

Fig. 8. Network topology of evaluation environment 1.

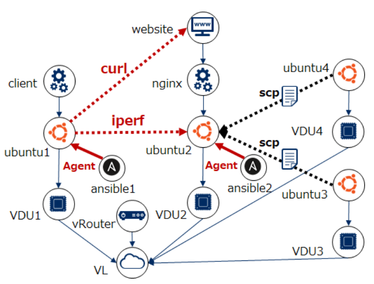

Fig. 9. Network topology of evaluation environment 2.

TABLE V
EVALUATION RESULTS ON EVALUATION ENVIRONMENT 1

Evaluation Units	Results	Program Output
`HTTP[App->App]`	**PASS**	`<!DOCTYPE HTML>` `<html><head>` `<title>Welcome to nginx!</title> ...`
`Bandwidth[App->App]` ≥ `500Mbps`	**PASS**	`[ 3] local 192.168.1.101 port 45407 connected with 192.168.1.102 port 5001` `[ ID] Interval    Transfer    Bandwidth` `[ 3] 0.0-10.0 sec 977 MBytes` **818 Mbits/sec** `...`

TABLE VI
EVALUATION RESULTS ON EVALUATION ENVIRONMENT 2

Evaluation Units	Result	Program Output
`HTTP[App->App]`	**PASS**	`<!DOCTYPE HTML>` `<html><head>` `<title>Welcome to nginx!</title> ...`
`Bandwidth[App->App]` ≥ `500Mbps`	**FAIL**	`[ 3] local 192.168.1.101 port 45812 connected with 192.168.1.102 port 5001` `[ ID] Interval    Transfer    Bandwidth` `[ 3] 0.0-10.0 sec 366 MBytes` **306 Mbits/sec** `...`

requirements. We could also easily find the system elements or connections to be improved, even though the deployed system design did not satisfy the system requirements.

C. Evaluation-Program-Generation Time

Finally, we evaluated the time required for generating an evaluation program. Our goal with the proposed method is to automate evaluation-program generation at the network level and reduce engineers' burden for the testing phase of SI. We measured the execution time of our implementation for different pattern of system requirements and designs.

In this experiment, the number of client applications varied from 1 to 5, while the number of server applications was fixed to one. Each client node had two evaluation units `HTTP[App->App]` and `Bandwidth[App->App]` ≥ `500Mbps`. We generated the system design for each system requirement pattern by using Weaver.

We measured the evaluation-program-generation time for 10 trials for each pattern. The implementation program was executed on a physical server with Intel©Xeon©E5-2420 CPU @ 1.90 GHz and 48 GBytes of RAM. The measurement results of TOSCA-format-generation time including parameter searches and entire evaluation-program-generation time are shown in Figs. 10 and 11, respectively. The horizontal axis shows the number of evaluation units, and the vertical axis shows the average time for evaluation program generation.

Fig. 10 shows the average time for generating an expanded TOSCA format from input system requirements, including parameter searches of Algorithm 1. This figure also shows that the parameter-search process finishes within a second and there is no noticeable difference in the number of clients and evaluation units. This is because each client application is in an independent relationship with the server, and an increase in client applications has no effect on the parameter search of each client applications.

Fig. 11 shows that the total program-generation time drastically increases as the evaluation units increase. This increase is due to the workflow-generation procedure in our provisioning planner. When the evaluation scripts generated from command templates are combined into the evaluation program, our provisioning planner calculates the execution order of scripts, in consideration of dependencies between tasks. Although the

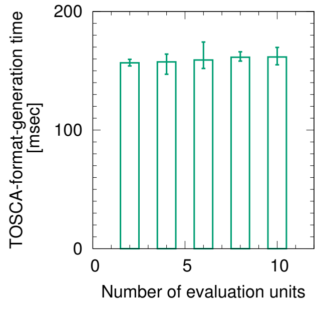

Fig. 10. Evaluation results on average time for generating expanded TOSCA format.

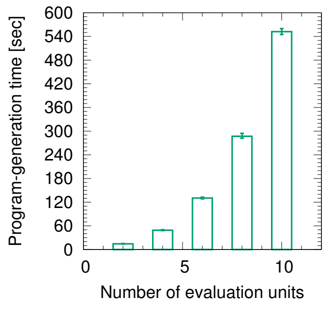

Fig. 11. Evaluation results on average program-generation time.

number of scripts for each evaluation unit is the same for every system design, the number of total scripts and their planning time increase as more client applications are connected. We will improve the workflow-generation procedure of our provisioning planner so that the execution order of scripts can be determined as referring to the command templates listed in Table II for future work. Nevertheless, Fig. 11 shows that our proposed method automatically generates an evaluation program in minutes or hours, even though the target system design becomes larger and more evaluation units need to be evaluated.

V. CONCLUSION

We proposed a method of ICT system evaluation-program generation for automating the network-level testing phase of IBN-based SI. Our method refers to the evaluation templates that define abstract commands and the command templates that translate the commands into actual evaluation scripts according to parameter conditions. By referring to these two templates, the whole process of evaluation-program generation can be sequentially automated and clients' intents can be concretized into corresponding programs. The evaluation results indicate that the method correctly acquires the required parameters of evaluation scripts and flexibly generates the evaluation program in a practical time. We also confirmed that an evaluation program can determine the satisfaction of system requirements for the deployed system design based on their evaluation units.

In this study, we assumed that the parameters of evaluation commands can be uniquely found by using search methods. To use our proposed method in more complex system designs, e.g., servers with multiple NICs and IP addresses, and network topologies that contain redundant paths, its parameter-search algorithm should be improved so that the appropriate parameter value is chosen from the multiple search results. Moreover, to indicate that our method is applicable for much larger ICT systems, we should improve our implementation program so that workflow-generation procedure is finished more rapidly.

ACKNOWLEDGMENTS

A part of this work was conducted as part of the project entitled "Research and development for innovative AI network integrated infrastructure technologies (JPMI00316)," supported by the Ministry of Internal Affairs and Communications, Japan.

References

[1] T. Kuroda et al., ”Weaver: A Novel Configuration Designer for IT/NW Services in Heterogeneous Environments,” in Proc. IEEE GLOBECOM 2019, Dec. 2019, pp. 1-6.

[2] T. Kuroda, M. Nakanoya, A. Kitano and A. Gokhale, ”The configuration-oriented planning for fully declarative IT system provisioning automation,” in Proc. IEEE/IFIP NOMS 2016, Apr. 2016, pp. 808-811.

[3] ”Jenkins,” http://jenkins.io/

[4] F. Wang and W. Du, ”A Test Automation Framework Based on WEB,” in Proc. IEEE/ACIS ICCIS 2012, June 2012, pp. 683-687.

[5] Xiao-yang Guo, Ying-hui Chen, Xue-song Qiu and Fan Tang, ”Design and implementation of performance testing model for Web Services,” in Proc. CAR 2010, Mar. 2010, pp. 353-356.

[6] ”NetTester,” http://github.com/net-tester/net-tester/

[7] ”Cucumber,” http://cucumber.io/

[8] Y. Li et al., ”A Survey on Network Verification and Testing With Formal Methods: Approaches and Challenges,” in IEEE Communications Surveys & Tutorials, vol. 21, no. 1, pp. 940-969, Firstquarter 2019.

[9] H. Mai et al., ”Debugging the Data Plane with Anteater,” in Proc. ACM SIGCOMM 2011, Aug. 2011, pp. 290-301.

[10] F. Le, G. G. Xie, and H. Zhang, ”Instability Free Routing: Beyond One Protocol Instance,” in Proc. ACM CoNEXT 2008, Dec. 2008, p. 9.

[11] G. D. Plotkin, N. Bjørner, N. P. Lopes, A. Rybalchenko, and G. Varghese, ”Scaling Network Verification Using Symmetry and Surgery,” in Proc. ACM PLDI, June 2014, pp. 69-83.

[12] ”Openstack,” http://openstack.org/

[13] ”Red Hat Ansible,” http://ansible.com/

[14] OASIS, ”TOSCA Simple Profile in YAML Version 1.3,” https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.3/os/TOSCA-Simple-Profile-YAML-v1.3-os.pdf