Communication Simulator with Network Behavior Logging Function for Supporting Network Construction Exercise for Beginners

Yuichiro Tateiwa and Naohisa Takahashi

Abstract Interconnecting virtual machines realizes computer networks on ordinary personal computers. Such a technique enables each student instead of a group to construct networks in network exercises for beginners. In the exercises, students may ask teachers to judge the correctness/incorrectness of their networks and to support the debugging for their networks. The waiting time of students can be long because the number of teachers is less than the number of students. An effective solution to this problem is to develop a system that can judge whether students' networks are correct and visualize the behavior of students' networks as hints. Detail logs of network behavior are necessary for realizing such a system. Here, we propose a communication simulator to record network behavior in detail during request/response communications, which are the transmissions of request data (e.g., icmp echo request) and the corresponding response data (e.g., icmp echo reply).

Keywords Communication simulator · Network construction exercise · Network behavior log

1 Introduction

It is important to increase the number of network engineers who administer computer networks as an infrastructure to a ubiquitous society and provide new services to the society. The experience of basic network construction is useful for not only network administrators but also network application programmers and network system designers.

Y. Tateiwa (✉) · N. Takahashi
Nagoya Institute of Technology, Gokiso-cho, Showa-ku, Nagoya, Aichi, Japan
e-mail: tateiwa@nitech.ac.jp
URL: http://tk-www.elcom.nitech.ac.jp/

N. Takahashi
e-mail: naohisa@nitech.ac.jp

G. De Pietro et al. (eds.), *Intelligent Interactive Multimedia Systems and Services 2016*, Smart Innovation, Systems and Technologies 55,
DOI 10.1007/978-3-319-39345-2_22

Network engineers define network specifications based on the network usage specified by their clients, and then, these engineers construct networks that satisfy these specifications. Therefore, there are network exercises where problems are based on frequent usage examples and students must construct networks for the available examples. We call these examples **communication examples**; they include "Client A can browse web sites on server B" and "Communications from client C to server A are intercepted by firewall D."

In traditional exercises for beginners, students are first divided into several groups; then, they construct networks with physical network devices. In recent years, however, it has become possible to realize networks by an interconnection of virtual machines running on an ordinary personal computer because of the performance improvement of personal computers and the evolution of virtual machine technology. We call such networks of virtual machines **VMN**. VMNs realize an e-learning environment where each student can construct his/her own network without considering time and placement (e.g., [1–5]).

In traditional exercises, each group attempts to construct networks based on exercise problems. When they want to confirm the correctness of their networks or cannot solve certain problems related to these networks, they approach their teachers for help. In e-learning, however, their waiting time is increased because students construct networks individually and the number of teachers is less than the number of students. An effective solution to this problem is to develop a system that can judge whether students' networks (we call them **answers**) are correct and visualize the behavior of these networks as hints.

Detail logs of network behavior are necessary for realizing such a system. For example, the logs needed for the judgment system are passing points of communication data (e.g., network interface Y on node X) and a terminated operation (e.g., discarding communication data because of a mismatch of the destination MAC address in the communication data to the MAC address assigned to the network interface that received the communication data). By comparing such data between the answers and the correct answers, the system judges whether the answers are correct. The logs needed for the hint generation system are operations for communication data in each node, one of which is packet forwarding caused by a mismatch of the destination IP address in the communication data to the IP addresses in its received node. It is not possible to gather such data by using packet capturing of tcpdump [6] or by using the trace data of network simulator ns-3 [7].

Therefore, here, we propose a communication simulator to record network behavior in detail during **request/response communications**, which are transmissions of request data (e.g., icmp echo request) and the corresponding response data (e.g., icmp echo reply). Fourteen proposed operations realize request/response communication in the simulator. The simulator receives network configurations and communication attributes (e.g., source node and destination IP address) as its input and then, computes the transmission of request data and response data. During such computation, the simulator writes the executed operations into a file (called **log file**) one by one. The simulator has the following features:

- The target of simulation is limited to the transmissions of requests and responses because it is used for judging whether the communication examples are successful in the networks.
- Each operation uses a few configurations as its input in order to obtain detailed relations between configurations and the success/failure of the communication examples.

2 Preparation

This section describes the operators to operate data structures.

- operator []: It returns a sequence whose elements are its arguments. The arguments are of variable length and are separated by commas. For example, "[1, 2, 3]" returns a sequence whose first element is "1," whose second element is "2," and whose last element is "3."
- operator <>: It returns a structure whose elements are its arguments. The arguments are of variable length and are separated by commas.
- operator { }: It returns a set whose elements are its arguments. The arguments are of variable length and are separated by commas.
- operator +: It returns a merged sequence between its arguments. For example, "[1] + [2] + [3]" returns a sequence whose first element is 1, whose second element is 2, and whose last element is 3.
- operator .: It accesses the value of a member valuable in a structure. For example, when a structure s has a member variable e and there is a variable y and whose data type is s, "y.e" accesses the value of valuable e.

3 Network Construction Exercise

3.1 Exercise Purpose

The target students have studied TCP/IP but have never constructed networks. The exercise target involves the students getting accustomed to designing networks that satisfy the following requirements and construct networks based on these designs.

1. IP address and subnet mask
2. IP network in a segment
3. Default route
4. Static routing
5. Server services (web page publishing on WWW)

Table 1 Data structure used by network devices

Name	Meaning	Data structure
nd	Node identifier	String
work	State of working	Boolean (if it is working, the value is true; otherwise, the value is false)
proc	A set of running processes	A set consisting of structures (process name **name**, listening protocol name **prot**, listening port number **port**, listening IP address **ip**)
ni	A set of network interface configurations	A set consisting of structures (network interface name **name**, assigned IP address **ip**, assigned subnet mask **mask**, assigned MAC address **mac**)
rtable	Routing table	A set consisting of structures (destination IP address *ip*, destination network address *nwaddr*, next hop IP address *nh*, subnet mask of *nwaddr* **mask**, sender network interface name *ni_name*)
atable	ARP table	A set consisting of structure (IP address **ip**, MAC address **mac**)
ni_num	Number of network interfaces	Integer

Table 2 Data structure of network

Name	Data structure
Network	A structure (a host set **hs**, a router set **rt**, a switching hub set **sw**, a repeater hub set **rp**, a cable set **cb**)
Host	A structure (nd, work, proc, ni, rtable, atable)
Router	A structure (nd, work, ni, rtable, atable)
Switching hub	A structure (nd, work, ni_num)
Repeater hub	A structure (nd, work, ni_num)
Cable	A structure (node identifier **nd1**, network interface name of nd1 **ni1_name**, node identifier **nd2**, network interface name of nd2 **ni2_name**)

Table 3 Actions of network devices

Name	Actions
Host	It receives a destination IP address and a destination port number as its input, and gets web pages with HTTP. It receives a destination IP address as its input, and checks a network continuity by using icmp echo messages
Router	It relays IP packets. It replies to ICMP echo requests
Switching hub	It relays Ethernet frames
Repeater hub	It relays Ethernet frames
Cable	It transmits data only between a network interface (host or router) and another network interface (switching hub or repeater hub)

3.2 Network

Tables 1, 2, and 3 list the specifications of networks in the exercise.

3.3 Structure of Exercise Problem

Exercise problems consist of **communication examples** and **configuration requirements**. The former is an example of communication available in students' networks. The latter consists of nodes that must be installed into networks, and setting values that must be set in nodes, and cables that must connect the assigned nodes.

Communication examples consist of items in Table 4. Configuration requirements have the same data structure as the network, as given in Table 2. Node identifiers (e.g., the node identifier of host hs_0 is $hs_0.nd$) and network interface names (e.g., the network interface name of router rt_0 is $ni_0.name(ni_0 \in rt_0.ni)$) must be concrete values, and the other items can be either '$*$' (denoting arbitrary values) or concrete values.

3.4 Example of Exercise Problem

Communication examples are expressed using natural language in actual exercise problems. For example, when a communication example consists of the following values, it is expressed by sentences given at the top of Fig. 1.

Table 4 Data of communication example

Name	Data structure
kind	A string where "web-get" represents a communication getting web pages from WWW servers and "icmp-echo" denotes the communication checking network continuity
source	A string that denotes a node identifier
destination	A structure (destination node identifier **nd**, process name running on node nd **proc**, listening IP address of proc **ip**, listening port number of proc **port**, listening protocol name of process proc **prot**)
communication route	A labeled rooted tree whose vertices are nodes that send or receive data, labels are identifiers of the nodes, edges are data transmissions between nodes, and root is the source node. Note that multiple children exist when the data are broadcast by repeater hubs and switching hubs
process reception information	A set consisting of structures (**proc**, **t**), where proc denotes a process reading requests or responses, t represents a communication timepoint, i.e., a time point when the communication data remain at a node of communication). Values of communication timepoints are vertices in communication routes

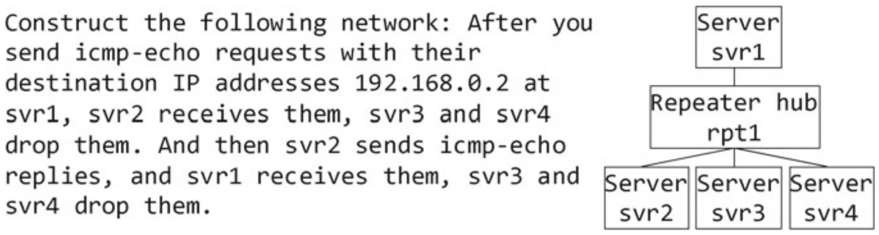
Construct the following network: After you send icmp-echo requests with their destination IP addresses 192.168.0.2 at svr1, svr2 receives them, svr3 and svr4 drop them. And then svr2 sends icmp-echo replies, and svr1 receives them, svr3 and svr4 drop them.

Fig. 1 Exercise problem

- kind $k =$"*icmp* − *echo*"
- source $s =$"*svr*1"
- destination $d = <$"", "", "192.168.0.2", "*icmp*", ""$>$
- communication route $R =< v_0, V, E, L, F >, V = [v_0, v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}], E = \{\{v_0, v_5\}, \{v_5, v_2\}, \{v_5, v_3\}, \{v_5, v_4\}, \{v_6, v_{10}\}, \{v_{10}, v_7\}, \{v_{10}, v_8\}, \{v_{10}, v_9\}, \{v_1, v_7\}\}, L =$ ["*svr*1", "*svr*2", "*svr*3", "*svr*4", "*rpt*1", "*svr*1", "*svr*2", "*svr*3", "*svr*4", "*rpt*1"]$, F(V_i) returnsL_i$
- process reception information $P = \{<v_1,$"*kernel*"$>, <v_6,$ "*client*"$>\}$

Configuration requirements are expressed using natural language and figures in actual exercise problems. For example, when configuration requirements $cr =< hs, rt, sw, rp, cb >$ consist of the following values, they are expressed by a figure given at the bottom of Fig. 1, where letters at the top of the squares denote node type and letters at the bottom represent the node identifier.

- $hs = \{n1, n2, n3, n4\}, rt = \{\}, sw = \{\}, rp = \{n5\}, cb = \{l1, l2, l3, l4\}$
- $n1 = <$"*svr*1"$, *, *, *, *, *>, n2 = <$"*svr*2"$, *, *, *, *, *>, n3 = <$"*svr*3"$, *, *, *, *, *>, n4 = <$"*svr*4"$, *, *, *, *, *>, n5 = <$"*rpt*1"$, *, 5>$
- $l1 = <$"*svr*1"$, *,$"*rpt*1"$, * >, l2 = <$"*rpt*1"$, *,$"*svr*2"$, * >, l3 = <$"*rpt*1"$, *,$"*svr*3"$, * >, l4 = <$"*rpt*1"$, *,$"*svr*4"$, * >$

3.5 Exercise Process

Each student constructs networks with physical devices and virtual devices on the basis of exercise problems. Then, they write reports on the executed steps and configurations in the constructions. After solving all problems, they submit their reports to their teachers. In the case they cannot comprehend certain sections of the exercises, they approach their teachers or teaching assistants for help.

4 Communication Simulator

4.1 Function Definition

The deta structure of communication data **pkt** used in the following functions is a structure (protocol name **prot**, destination IP address **dip**, source IP address **sip**, destination port number **dp**, source port number **sp**, payload **pl**, destination MAC address **dmac**, source MAC address **smac**).

4.1.1 Functions to Get Data from Configurations

- RT(nd): It returns a routing table from the node whose identifier is nd.
- AT(nd): It returns an ARP table from the node whose identifier is nd.
- MAC(nd): It returns a set consisting of structures (**ni**, **mac**), where ni is a network interface name in the node whose identifier is nd, and mac is its MAC address.
- IP(nd): It returns a set consisting of structures (**ni**, **ip**), where ni is a network interface name in the node whose identifier is nd, and ip is its IP address.
- NI(nd): It returns a set consisting of all network interfaces in the node whose identifier is nd.
- PROC(nd): It returns a set consisting of listening processes in the node whose identifier is nd.
- WORK(nd): It returns the working state of the node whose identifier is nd.
- ARP(nd, ni, ip): If the node whose identifier is nd gets a MAC address corresponding to an IP address ip by an ARP communication at the network interface whose name is ni, it returns the MAC address; otherwise, it returns "".
- KIND(nd): If the node of nd is a host, it returns "hs"; else, if the node of nd is a router, it returns "rt". Further, if the node of nd is a switching hub, it returns "sw"; else, if the node of nd is a repeater hub, it returns "rp".
- PairNI(nd, ni_name): Here, nd denotes a node identifier, and ni_name representes a network interface name. It returns a tupple $(nd_{pair}, ni\_name_{pair})$ whose ni_name_{pair} is the name of the network interface connected to a network interface whose name is ni_name in the node whose identifier is nd, and whose nd_{pair} is an identifier of the node that has the network interface ni_name_{pair}.

4.1.2 Functions Consisting Operations in Node

Every function writes its name, its inputs, and its outputs at the end of a **log file** during its execution.

- ReqPkt(prot, pl, dip, dp): Here, prot denotes the protocol name; pl, the payload; dip, the destination IP address; and dp, the destination port number. If $prot =$"tcp" is true, it inputs at random an ephemeral port number into source port sp; otherwise, it sets $sp =$"". Then, it returns <$prot, dip$, "", dp, sp, pl, "", ""> as the communication data.

- RepPkt(proc, pkt): Here, proc denotes a process name and pkt represents the communication data. If $proc =$"*apache*" $\wedge pkt.pl =$ "*HTTP GET REQUEST*" is true, it sets $pl =$ "*HTTP GET RESPONSE*"; else, if $proc =$"*kernel*"$\wedge pkt.pl =$ "ICMP *ECHO REQUEST*" is true, it sets $pl =$ "*ICMP ECHO REPLAY*". Then it returns $<pkt.prot, pkt.sip, pkt.dip, pkt.sp, pkt.dp, pl,$"", ""$>$ as the communication data.
- ChkDIP(dip, nd): Here, dip denotes the destination IP address and nd represents the node identifier. If there is i whose $i.ip = dip(i \in IP(nd))$ is true, it returns true; otherwise, it returns false.
- ChkDMac(dmac, nd, ni): Here, dmac denotes the destination MAC address; nd, the node identifier; and ni, the network interface name. If there is m whose $m.ni = ni \wedge m.mac = dmac(m \in MAC(nd))$ is true, it returns true; otherwise, it returns false.
- Proc(prot, dp, dip, nd): Here, proc denotes the process name; dp, the destination port number; dip, the destination IP address; and nd, the node identifier. If there is proc whose $proc.prot = prot \wedge proc.port = dp \wedge proc.ip = dip$ in $proc \in PROC(nd)$ is true, it returns $proc.name$; otherwise, it returns "".
- L1Rtng(nd, ni): Here, nd denotes the node identifier and ni represents the network interface name. It returns a queue whose elements are ni and whose $ni \neq ni_s \wedge WORK(nd_{adj}) = true \wedge nd_{adj} \neq$ "" is true in $(nd_{adj}, ni_{adj}) = PairNI(nd, ni_s)(ni_s \in NI(nd))$. The elements are sorted in an ascending order by nd_{adj} as the first key and ni_{adj} as the second key.
- L2Rtng(dmac, ni, nd): Here, dmac denotes the destination MAC address; ni, the network interface name; and nd, the node identifier. If there is ni_s whose $m.ni = ni_{adj} \wedge m.mac = dmac \wedge ni \neq ni_s \wedge WORK(nd_{adj}) = true \wedge nd_{adj} \neq$ "" in $m = MAC(nd_{adj}), (nd_{adj}, ni_{adj}) = PairNI(nd, ni_s), ni_s \in NI(nd)$ is true, it returns a queue whose elements are ni_s sorted in the ascending order by nd_{adj} as the first key and ni_{adj} as the second key; otherwise, it returns $L1Rtng(nd, ni)$.
- L3Rtng(dip, nd): Here, dip denotes the destination IP address, and nd represents the node identifier. If it finds $rtable \in RT(nd)$ whose $rtable.ip$ matches dip by the longest prefix match or whose $rtable.nw$ and $rtable.mask$ match dip by the longest prefix match, it returns a tupple $(rtable.nh, rtable.ni)$; otherwise, it returns a tupple ("", "").
- Rcv(proc, pkt): Here, proc denotes a process name and pkt represents communication data. If ($proc =$"*apache*" $\wedge pkt.pl =$"*HTTP GET REQUEST*") $\vee$ ($proc =$ "*kernel*"$\wedge pkt.pl =$"*ICMP ECHO REQUEST*") is true, it returns *false*; otherwise, it returns *true*.
- SIP(nd, ni): Here, nd denotes the node identifier and ni represents the network interface name. If there is i whose $i.ni = ni(i \in IP(nd))$ is true, it returns i.ip; otherwise, it returns "".
- DMac(ni, dip, nd): Here, ni denotes the network interface name; dip, the destination IP address, and nd, the node identifier. If $record.mac \neq$ "" $(record \in AT(nd))$ is true, it returns $record.mac$; otherwise, it returns $ARP(nd, ni, dip)$.
- SMac(nd, ni): Here, nd denotes the node identifier and ni represents the network interface name. If there is m whose $m.ni = ni(m \in MAC(nd))$ is true, it returns m.mac; otherwise, it returns "".

- Transmit(nd, ni): Here, nd denotes the node identifier and ni represents the network interface name. It returns $PairNI(nd, ni)$.
- Dsc(pkt): Here, pkt denotes the communication data. It discards pkt. It returns nothing.

4.2 Pesudo Code

Procedures 1–4 show the pesudo code of the simulator. When you execute simulation by this simulator on the basis of communication example E, you execute function Communicate(d.prot, pl, dip, d.port, s), where d denotes the destination of E; pl is set as "HTTP GET REQUEST" (type of E is "web-get"); pl is set as "ICMP ECHO REQUEST" (type of E is "icmp-echo"); dip is set as "d.ip" (d.nd=""); dip is set as $ni.ip(ni \in IP(d.nd))$ ($d.nd \neq$ ""); and s is source of E.

Procedure 1 Communicate

```
Input: protocol name prot, payload pl, destination IP address dip, destination port number dp,
    source node identifier nd
1: if $WORK(nd)$ then
2:     $pkt = ReqPkt(prot, pl, dip, dp)$
3:     $HsRtOut(pkt, nd)$
4: else
5:     $Dsc(pkt)$
6: end if
```

5 Prototype System

5.1 Communication Simulator

Let us consider an incorrect network for the exercise problem described in Sect. 3.4. Its incorrect configurations are the IP addresses of svr4 (192.168.0.2) and svr2 (192.168.0.254). Figure 2 shows a part of the log file generated by our simulator by using the network. Each line consists of a function name, input values, and output values. The symbol "\\" on the right side denotes hyphenation.

```
ReqPkt,icmp,icmp-echo,192.168.0.2,,<icmp,192.168.0.2,,,,icmp-echo,,>
L3Rtng,192.168.0.2,svr1,true,{<kernel,icmp,,>},{<dummy0,,,AE:10:94:\\
E3:51:E7>,<eth0,192.168.0.1,255.255.255.0,FE:FD:98:82:21:3A>,<lo,12\\
7.0.0.1,255.0.0.0,>,<tunl0,,,>},{<192.168.0.0,0.0.0.0,255.255.255.0\\
,U,0,0,eth0>},{},<0.0.0.0,eth0>
```

Fig. 2 Log file

Procedure 2 HsRtOut

```
Input: sent communication data pkt, identifier of sender node nd
 7: $(ip, ni) = L3Rtng(pkt.dip, nd)$
 8: if $ni \neq$ “” then
 9:     if $pkt.sip =$ “” then
10:         $pkt.sip = SIP(nd, ni)$
11:     end if
12:     $pkt.smac = SMac(nd, ni)$
13:     $pkt.dmac = DMac(ni, ip, nd)$
14:     if $pkt.dmac \neq$ “” then
15:         $(nd_r, ni_r) = Transmit(nd, ni)$
16:         if $KIND(nd) =$“$rp$”$\vee KIND(nd) =$ “$sw$” then
17:             $SwRp(pkt, nd_r, ni_r)$
18:         else
19:             $Dsc(pkt)$
20:         end if
21:     else
22:         $Dsc(pkt)$
23:     end if
24: else
25:     $Dsc(pkt)$
26: end if
```

Procedure 3 SwRp

```
Input: received data pkt, identifier of receiver node nd, name of receiver network interface ni
27: if $WORK(nd)$ then
28:     if $KIND(nd) =$“$rp$” then
29:         $NI_s = L1Rtng(nd, ni)$
30:     else if $KIND(nd) =$“$sw$” then
31:         $NI_s = L2Rtng(pkt.dmac, ni, nd)$
32:     end if
33:     if $NI_s \neq \{\}$ then
34:         repeat
35:             $pickupanelementfromtheheadofNI_sandsetittoni_s$
36:             $(nd_r, ni_r) = Transmit(nd, ni_s)$
37:             if $KIND(nd_r) =$“$hs$”$\vee KIND(nd_r) =$“$rt$” then
38:                 $HsRtIn(pkt, nd_r, ni_r)$
39:             else
40:                 $Dsc(pkt)$
41:             end if
42:         until $NI_s \neq \{\}$
43:     else
44:         $Dsc(pkt)$
45:     end if
46: else
47:     $Dsc(pkt)$
48: end if
```

Procedure 4 HsRtIn

```
Input: received data $pkt$, identifier of receiver node $nd$, name of receiver network interface $ni$
49: if $WORK(nd)$ then
50:     if $ChkDMac(pkt.dmac, nd, ni) = true$ then
51:         if $ChkDIP(pkt.dip, nd) = true$ then
52:             $proc = Proc(pkt.prot, pkt.dp, pkt.dip, nd)$
53:             if $proc \neq$ “” then
54:                 if $Rcv(proc, pkt) = false$ then
55:                     $pkt_s = RepPkt(proc, pkt)$
56:                     $HsRtOut(pkt_s, nd)$
57:                 end if
58:             else
59:                 $Dsc(pkt)$
60:             end if
61:         else
62:             $HsRtOut(pkt, nd)$
63:         end if
64:     else
65:         $Dsc(pkt)$
66:     end if
67: else
68:     $Dsc(pkt)$
69: end if
```

The first log implies that the execution of function ReqPkt with prot = “icmp,” pl = “icmp-echo,” dip = “192.168.0.2,” and dp = “” as its input arguments returned communication data (prot = “icmp,” dip = “192.168.0.2,” sip = “”, dp = “”, sp = “”, pl = “icmp-echo,” smac = “”, and dmac = “”). The second log implies the execution of function L3Rtng, and in particular, the log stores the node information of its argument nd.

5.2 Application Example of Log

The visualization of communication is helpful for students to debug their networks. We visualized the executed operations during communication by getting several data from the log file. Figure 3 expresses the executed operations, their orders, and their nodes by a directed graph. The vertices consist of “an operation name,” “@,” and “identifier of its execution node.” The direction of the edges denotes the execution order of the operations on the vertices. This network is incorrect because svr4 receives communication data while the exercise problem requires that svr2 receives the data according to the graph.

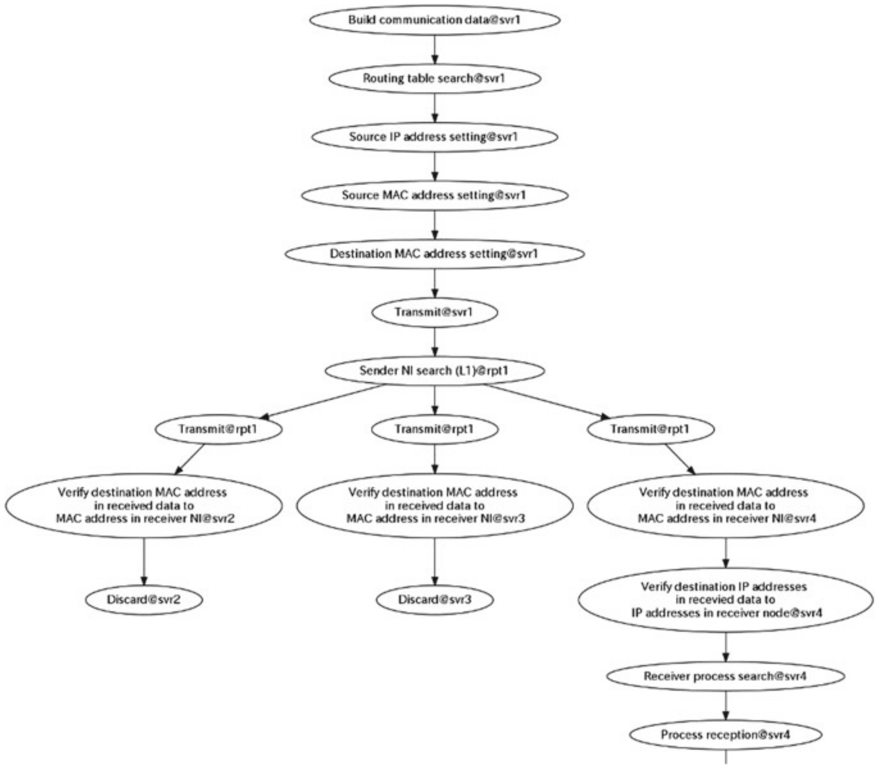
Build communication data@svr1
Routing table search@svr1
Source IP address setting@svr1
Source MAC address setting@svr1
Destination MAC address setting@svr1
Transmit@svr1
Sender NI search (L1)@rpt1
Transmit@rpt1
Transmit@rpt1
Transmit@rpt1
Verify destination MAC address in received data to MAC address in receiver NI@svr2
Verify destination MAC address in received data to MAC address in receiver NI@svr3
Verify destination MAC address in received data to MAC address in receiver NI@svr4
Discard@svr2
Discard@svr3
Verify destination IP addresses in recevied data to IP addresses in receiver node@svr4
Receiver process search@svr4
Process reception@svr4

Fig. 3 Graph of execution operations

6 Evaluation Experience

This experience aims at clarifying the basic performance of our simulator. We implemented our simulator in C++, and executed it with ping executions of several networks on a computer with an Intel core i7-3770K 3.50-GHz CPU and 4-GB main memory. Table 5 shows the result. “Transmissions” denotes all counts at which the nodes transmitted communication data. “Routing entries” representes the counts of the routing table entries in all nodes. According to the result, we conclude that our simulator is available for beginners’ exercises because the number of routers in the exercises is less than 10, and the consumption of the computer resources in the case is enough small to current computers.

Table 5 Measurement result

Network	Communication	Average CPU usage (%)	Exec. time (s)	Max. RSS (KB)	Transmissions	Routing entries	Log file size (Bytes)	Log counts
A correct network (Fig. 4)	Execute ping to svrR at svrL	64	0.006	2,980	8	4	5,594	36
Fig. 4 (N = 10)	Execute ping to svrR at svrL	81	0.02	3,488	44	112	91,061	184
Fig. 4 (N = 100)	Execute ping to svrR	96	1.228	37,076	404	10,102	6,718,846	1,624
Fig. 4 (N = 254)	Execute ping to svrR	96	7.22	213,204	1,020	64,772	43,354,830	4.088

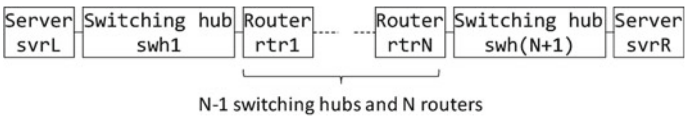

Fig. 4 Network for the experiment. It is possible to transmit data between svrL and svrR

7 Conclusion

In this paper, we proposed a simulator that stores network behavior for request/response communications. We consider that the stored data are useful for the correct/incorrect judgment and hint generation. We showed a graph to express network behavior as an example of hints. We carried out an evaluation experiment where we measured the basic features of the simulator. According to its result, we concluded that the simulator can sufficiently work on computers with ordinary performance in computing small networks that beginners construct in the exercises. Our future works include an expansion of the simulation target, e.g., firewalls and NAT.

Acknowledgments This study was partially funded by the Grants-in-Aid for Scientific Research Foundation (25750082) and public interest TATEMATSU foundation.

References

1. Tateiwa, Y. et al.: LiNeS: virtual network environment for network administrator education. In: Proceedings of Information and Control (ICICIC-2008) (2008)
2. Iguchi, N. et al.: Development of hands-on IP network practice system with automatic scoring function. In: Proceedings of 2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems, pp.704–709 (2013)
3. Le, Xu, Huang, Dijiang, Tsai, Wei-Tek: Cloud-based virtual laboratory for network security education. IEEE Trans. Educ. **57**(3), 145–150 (2014)
4. Wannous, Muhammad, Nakano, Hiroshi: NVLab, a networking virtual web-based laboratory that implements virtualization and virtual network computing technologies. IEEE Trans. Learn. Technol. **3**(2), 129–138 (2010)
5. Ruiz-Martínez, Antonio, Pereñíguez-García, Fernando, Marín-López, Rafael, Ruiz-Martínez, Pedro M., Skarmeta-Gómez, Antonio F.: Teaching advanced concepts in computer networks: VNUML-UM virtualization tool. IEEE Trans. Learn. Technol. **6**(1), 85–96 (2013)
6. TCPDUMP_LIBPCAP public repository. http://www.tcpdump.org/ (2016). Accessed 28 Jan 2016
7. ns-3. https://www.nsnam.org/ (2016). Accessed 28 Jan 2016