Practical Implementation of Web-Based Hands-on System for Network Security Classes

1st Yuichiro TATEIWA
Nagoya Institute of Technology
Aichi-ken, Japan
tateiwa@nitech.ac.jp

***Abstract*—This abstract represents our full paper categorized under the 'Innovative Practice' category.**

In traditional network experiments, students gathered in a classroom and built networks using specialized equipment. However, there was a problem of high financial costs associated with providing classrooms and equipment. On the other hand, with the advancement of virtual machine technology and the improvement of computer performance, it has become possible to realize networks on computers using virtual machines as network devices. E-learning systems have been developed to conduct network experiments by operating virtual machines from networked computers.

The author proposed LiNeS Cloud, a web-based exercise system aimed at intuitive and seamless operation with quick responsiveness for network security exercises using the virtual machine User-mode Linux. The prototype of LiNeS Cloud was evaluated in a laboratory and received favorable evaluations regarding its operability, and no issues were pointed out regarding its responsiveness.

In this paper, the author proposes methods to reduce construction errors, minimize operational overhead, and make it easier to find errors during troubleshooting in the construction and operation of the system for practical use. The author then describes the measurement results of computer load and the evaluation of responsiveness from students when operating the system implemented based on this method in an actual class.

***Index Terms*—network security, e-learning, virtual machine**

I. Introduction

With the advent of the ubiquitous society, everything is connected to the Internet. In such an environment, it is socially important to train engineers who can develop and operate network infrastructure and user service systems. Therefore, it is crucial for university students studying information engineering to learn and experience technology through network experiments (building networks and executing security tools).

In traditional network experiments, students gathered in a classroom and built networks using specialized equipment. However, there was a problem of high financial costs associated with providing classrooms and equipment. On the other hand, with the advancement of virtual machine technology and the improvement of computer performance, it has become possible to realize networks on computers using virtual machines as network devices. This has led to the development of e-learning systems that allow students to conduct network experiments by operating virtual machines from networked computers [1] [2] [3] [4] [5].

The author proposed LiNeS Cloud [6], a web-based exercise system aimed at intuitive and seamless operation with quick responsiveness for network security exercises using the virtual machine User-mode Linux (UML) [7]. This system has the following features:

- It builds virtual networks on remote servers and provides dynamic web pages for operating the networks. In addition, both text transfer and screen transfer are differently employed for communication between the user interfaces (UIs) and the servers. Consequently, the students can use the exercise environment comfortably and easily through browsers.
- It provides dynamic web pages where the students can edit the topologies of the virtual networks by mouse operations. Furthermore, UML terminals are mapped to the networks drawn on the web pages. They enable the students to intuitively form virtual networks.
- It builds overlay networks implemented by tunnels that can forward Ethernet frames between virtual devices running on various remote servers. This enables the implementation of virtual networks that contain various versions of UMLs.

Figure 1 shows examples of clients (i.e., student's UI) in Google Chrome for Mac, namely a topology page, a terminal page, and an X page sorted from the top. After selecting a device from the available icons (1), clicking on the area (2) will draw the selected device. The window (3) displays the property of the device *cli* and the terminal area at the bottom draws a UML terminal running on a remote server and accepts keyboard input. Clicking the button (4) displays the window (5) that shows the X window manager of the device *cli*. In window (5), the browser (X client) started in the device *cli* is displayed. Furthermore, the icons representing the devices in the area (2) can be moved by mouse dragging.

The prototype of LiNeS Cloud was evaluated in a laboratory and received favorable evaluations regarding its operability, and no issues were pointed out regarding its responsiveness. In this class, students design network and communication experiments to solve the tasks outlined in the instruction manual. They then construct networks based on their designs, execute communications, and observe the status of the equipment during these processes. Teachers and Teaching Assistants (TAs) are available to answer students' questions and assist

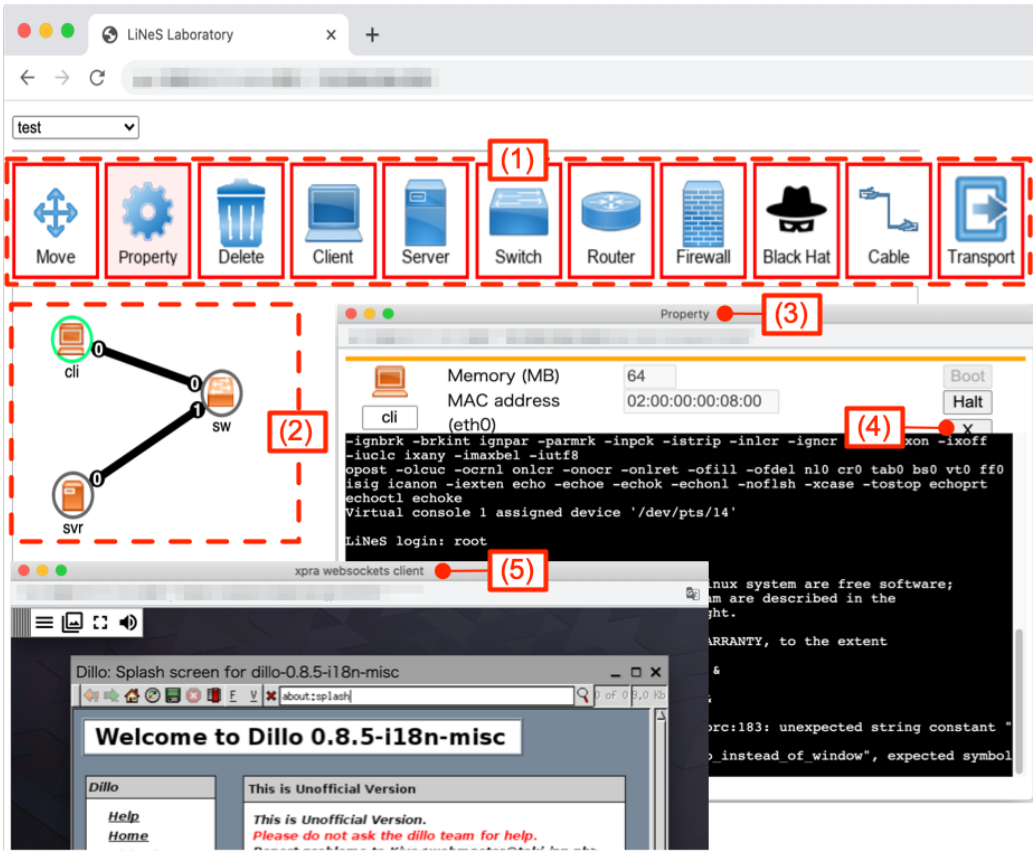


Fig. 1: Example of web pages for managing networks.

with network creation.

In this paper, the author proposes implementation methods to reduce construction errors, minimize operational overhead, and make it easier to find errors during troubleshooting in the construction and operation of the system for practical use. The author then describes the measurement results of computer load and the evaluation of responsiveness from students when operating the system, which was built based on these methods, in an actual class.

II. System design of LiNeS Cloud

The proposed method implements the system for practical use, following the system design proposed in the previous study [6]. Figure 2 shows the system architecture of LiNeS Cloud. The simulation servers accept remote operations from the configuration server and implement virtual networks by employing virtual devices. The virtual devices—servers, clients, routers, and firewalls—are realized using UML processes, while the virtual devices, switching hubs, are realized using bridge instances. The configuration server provides web pages for editing the connections among virtual devices (hereinafter referred to as topology pages), web pages for input/output with UML terminals (hereinafter referred to as terminal pages), and web pages for input/output with UML X clients (hereinafter referred to as X pages). The relay server enables the exchange of Ethernet frames between the tunnel interfaces of two simulation servers by connecting the tunnel interfaces of the Ethernet tunnels connecting themselves to the simulation servers using Bridges. The terminal server relays the input and output between the terminal pages and the UML terminals. The X forwarding server accepts drawing requests from UML X clients and draws the output on the X pages. The student can manage virtual networks with Windows, Mac, Android, and IOS web browsers.

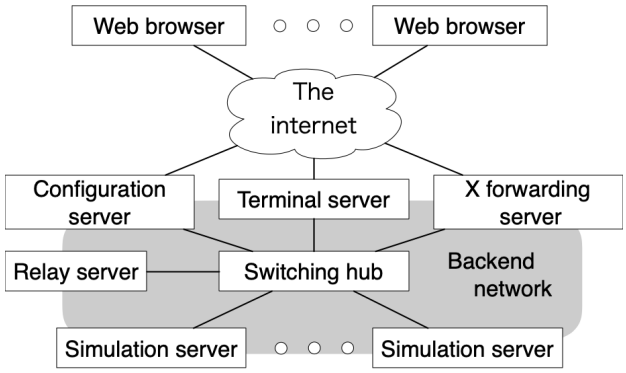


Fig. 2: System architecture.

III. Implementation method of the LiNeS Cloud prototype

This chapter describes the implementation method of the prototype system proposed in [6].

A. System architecture

The configuration server was implemented using Node.js [8], the topology pages using JavaScript, jQuery [9], Ajax, and SVG.js [10], the terminal pages using Xterm.js [11], the terminal server using Socket.IO [12], the remote command acceptance of the simulation servers using OpenSSH [13], the Ethernet tunnels using vtun [14], and the X forwarding server and X pages using Xvfb [15] and xpra [16]. Additionally, the network address of the backend network was set to 10.0.0.0/8.

B. Devices

The author implemented virtual devices (servers, clients, routers, and firewalls) using UML processes on the mock server. In this implementation, LiNeS Cloud obtains the UML process IDs using start-stop-daemon [17] to check the execution status of the UML processes and stop them. When creating a process for a command cmd and recording its process ID in a file named $umlPid$, the start-stop-daemon command is executed with the '-p' option set to $umlPid$ and the '–exec' option set to cmd.

Virtual devices implemented using UML have a specified number of network interfaces (e.g., eth0). The author implemented these interfaces using tap instances on the simulation server. To create a tap instance named $tapName$, the tunctl command is executed with the '-t' option set to $tapName$. To create a UML process that adopts a tap instance named $tapName$ as the network interface eth$N$ (N is a non-negative integer) and has an internal UML MAC address of $umlMac$, the UML command is executed with the 'eth$N$' option set to 'tuntap,$tapName$,$umlMac$'.

The terminal page provides access to the primary console (tty1) of the virtual device implemented using UML. The author implemented this functionality using the following mechanism: First, the system creates a screen [18] session with the UML's primary console connected to a screen virtual terminal. Then, upon generation of the terminal page, it attaches to that screen session, and upon deletion of the terminal page, it detaches from that screen session. This allows the generation/deletion of the terminal page to be separated from the generation/deletion of the UML process. As a result, for example, if a student closes the terminal page to take a break from the exercise, the operating state of the virtual device remains unchanged. To create a screen session named $userScreen$, the screen command is executed with the '-S' option set to $userScreen$. Furthermore, to generate a UML process with the UML's primary console connected to a screen virtual terminal, the screen command is executed with the UML command specified as the final argument.

The system sets the IP addresses and environment variables inside the UML. The author implemented this functionality using a screen session with a virtual terminal of screen connected to the UML's secondary console (e.g., tty2). The system can execute Linux commands within UML through this console (hereafter referred to as the management console). This session is created using the following steps:

1) Connect a pseudo-terminal to the UML's secondary console. To achieve this, execute the UML command with the 'con1' option set to 'pts'.
2) Extract the device file of the pseudo-terminal connected to the secondary console from the boot log output to the primary console.
3) Open that device file with screen and create a screen session named $mngrScreen$. To accomplish this, execute the screen command with the '-S' option set to $mngrScreen$ and the device file specified as the final argument.

To connect the X client of UML (client of the virtual device) with the X transfer server's xpra (i.e., X server), the following four items are required:

- Set the environment variable DISPLAY in UML to the IP address (on the backend network side of the X forwarding server in the figure) and display number $xpraDsp$ of the X forwarding server. To achieve this, use the management console after starting the UML process.
- Set the IP address $umlIp$ to the network interface (for forwarding X windows) inside UML. To accomplish this, use the management console after starting the UML process.
- Set the IP address $hostIp$ and MAC address $hostMac$ to the network interface (i.e., a tap instance) outside UML. To do this, create a tap instance named $tapName$, and set $hostIp$ and $hostMac$ using the ifconfig command. Then, execute the UML command with the 'eth$N$' option set to 'tuntap,$tapName$,$umlMac$,$hostIp$'.
- Make the X window managed by the display number $xpraDsp$ on the X forwarding server accessible from the X page via TCP port number $xpraPort$. To achieve this, execute the xpra command on the X transfer server with start as the first argument, $xpraDsp$ as the second argument, and $xpraPort$ specified for the '–bind-tcp' option.

The author implemented virtual devices (switches) using bridge instances on the simulation server. To create a bridge instance named $bridgeName$, the brctl command is executed with the 'addbr' option set to $bridgeName$.

C. Cable

The author implemented cables connecting virtual devices running on different simulation servers using vtun processes, tap instances, and bridge instances through the following steps:

1) With the two ends of the cable being a and b, create a tunnel from the simulation server on the a-side to the relay server, and a tunnel from the simulation server on the b-side to the relay server. To do this, execute the vtun command on the simulation servers with the second-to-last argument being the session name ($vtunSessionA$

for the a-side, $vtunSessionB$ for the b-side) and the last argument being the address of the relay server.
2) After tap instances $tapRelayA$ (a-side) and $tapRelayB$ (b-side) are created on the relay server, bridge them using the bridge instance $bridgeRelay$.
3) After tap instances ($tapSimA$ for the a-side, $tapSimB$ for the b-side) are created on the simulation servers, bridge them with the network device of the virtual device (a tap instance for UML, a bridge for a switch).

IV. Proposed method

A. Requirements

During the practical exercises, if troubles occur due to system errors or bugs, the system operator (that is, the author) must respond to them promptly. Therefore, it is especially important for the system operator to quickly identify the cause of the trouble based on the symptoms encountered by the students. Consequently, the system operator wants to easily establish a correspondence between the objects (i.e., devices and cables) that the student operates and the instances (e.g., UML processes, TAPs) and parameters (e.g., IP addresses) that implement them.

- The system operator wants to easily establish a correspondence between the objects (devices and cables) on the web page and the instances (UML process IDs, TAP names, Bridge names) on the simulation servers and relay server.
- The system operator wants to easily establish a correspondence between the client on the web page and the parameters (xpra listen port, Xvfb listen port, UML internal IP address, UML external IP address) used for forwarding X windows of that client.

B. Implementation method

In the prototype implementation of the section III, when a student places a device on the topology page, a record managing that device is added to the database (sqlite3 [19]). When Ethernet ports of devices are connected, a record managing that connection is added to the database. In this case, the ID of an sqlite3 record is of INTEGER type. For IDs between 0 and 29767, a 5-digit string with 0 padded at the head is called a device ID or cable ID. These IDs are also stored on the topology page and can be read from its HTML source code.

Table I shows the rules for embedding device ID DEV and cable ID CBL into the target (instances or parameters). For a variable v storing a string, the string concatenating v and the string 'str' is denoted as '${v}str', and the string consisting of the i-th to j-th characters of v is denoted as '${v[i:j]}'. The variable EPN stores a string of length 2 with 0-padded Ethernet port number. The tunnel ID TNL is '${CBL}a' when the tunnel is from the object connected to the a-side of the cable to the relay server, and '${CBL}b' when it is from the b-side. Finally, the values determined based on the rules are considered to be excellent in coding ease and not problematic for the scale of the system's use.

In the X forwarding server, Xvfb and xpra listen for external tcp access. The listen port of Xvfb is the display number plus 6000. The maximum value for a tcp listen port is 65535. Therefore, by using the allocation rules shown in Table I, the system can utilize 29768 virtual displays. Since the maximum length of a screen session ID is 32 characters, the allocation rules shown in the table satisfy this constraint. The maximum length of Bridge and TAP names is 15 characters, so the allocation rules shown in the table meet this constraint. In the MAC address allocation rule, for example, if DEV is '012345' and EPN is '01', then $umlMac$ becomes '02:00:01:23:45:01'. The maximum length of a vtun session name is 254 characters, so the allocation rules in the table satisfy this constraint. In the IP address allocation rule, for example, if DEV is '012345', then $hostIp$ becomes '10.48.105.1'. The X forwarding server and the UML's X clients can communicate on the 10.0.0.0/8 network. Additionally, the configuration server communicates with the relay server, X forwarding server, and simulation servers. Assuming a maximum of about 100 students, with each person using up to 2 types of simulation servers, a maximum of 200 simulation servers would be required. The IP addresses 10.255.255.1 to 10.255.255.254 are assigned to these servers. Therefore, it should be noted that the 10.0.0.0/8 network cannot be used for the students' networks.

V. Measuring computational load

A. Exercise content

The exercise consists of seven 90-minute classes. All students are divided into four groups, and each group attends the exercise separately. During this time, each student performs the following tasks according to the instruction manual:

- Build networks with servers, clients, switching hubs, and firewalls
- Attack networks by executing tools (DoS/DDoS, password cracking, packet sniffing, backdoors, session hijacking)
- Observe the state and logs of the victim machines and networks
- Filter packets with iptables

The instruction manual consists of 1) an explanation of network construction, attacks, and defense, 2) how to use tools to realize them and practice problems, and 3) exercise assignments to design attacks and defenses and consider their network behavior. Figure 3 shows an example of an experimental task, where Step 1 requires the construction of a network, Step 2 demands the blocking of ICMP echo communication, and Step 3 requires proof of the ICMP echo communication blocking through iptables logs.

During the class, instructors and TAs circulate among the students, answering questions about the experimental tasks and helping to resolve any network-related issues that students encounter.

Note that this exercise is a subset of the exercise described in the paper [6].

TABLE I: Embedding rules for device ID DEV and cable ID CBL into instances and parameters

Target	Rule
umlPid	${DEV}.umlpid
tapName	lines${DEV}-${EP}
bridgeName	linesbr${DEV}
umlMac	First octet=02
	Second octet=00
	Third octet=${DEV[0:1]}
	Fourth octet=${DEV[2:3]}
	Fifth octet=${DEV[4:5]}
	Sixth octet=${EP}
xpraDsp	${DEV}
xpraPort	35768 plus DEV arithmetically
hostMac	First octet=06
	Second octet=00
	Third octet=${DEV[0:1]}
	Fourth octet=${DEV[2:3]}
	Fifth octet=${DEV[4:5]}
	Sixth octet=09
umlIp	First octet=10
	Second octet=Quotient of DEV divided by 255
	Third octet=Remainder of DEV divided by 255
	Fourth octet=2
hostIp	First octet=10
	Second octet=Quotient of DEV divided by 255
	Third octet=Remainder of DEV divided by 255
	Fourth octet=1
userScreen	lines${DEV}
mngrScreen	linesM${DEV}
vtunSessionA	linestn${CBL}a
vtunSessionB	linestn${CBL}b
bridgeRelay	linesbr${CBL}
tapRelayA	linestn${CBL}a
tapRelayB	linestn${CBL}b
tapSimA	linestn${CBL}a
tapSimB	linestn${CBL}b
Servers' IP addresses on their backend network interfaces	10.255.255.1-10.255.255.254

Step 1: Construct the network shown in the figure below. The IP settings for the client and server can be arbitrary.
Step 2: Add rules to iptables to filter ping packets.
Step 3: Show evidence in the iptables logs that the added rules have been applied to ping packets.

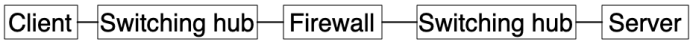


Fig. 3: Example of an experimental task

B. Exercise environment

Figure 4 shows the configuration of LiNeS Cloud required for the exercise content in the previous section. First, as the anticipated maximum number of students taking this class was considered to be 60, I prepared 15 client machines. Also, the X forwarding server and relay server were omitted from Figure 2. The reason for omitting the X forwarding server is that the exercises using X windows were omitted. The reason for omitting the relay server is that the exercise content can be covered with one type of simulation server, and by assigning a dedicated simulation server to each student, the impact on other students can be minimized in case of trouble with the exercise system, and troubleshooting can be made

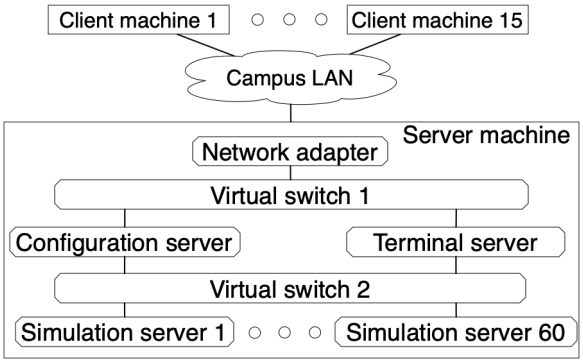


Fig. 4: Exercise environment

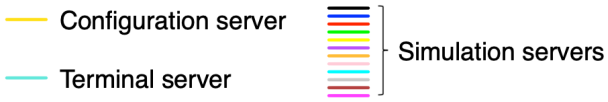


Fig. 5: Legends for Figures 6 - 14.

easier. Therefore, the number of simulation servers is 60, the same as the expected number of students taking this class.

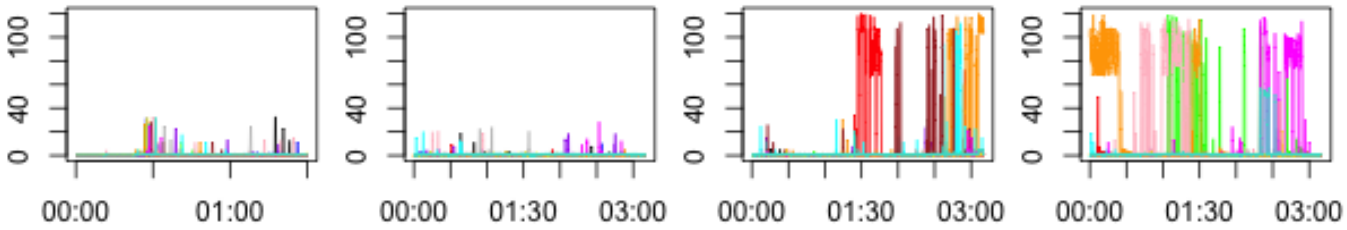
The following is the hardware specification of Figure 4.

- Client machine
  - Model: Dell OptiPlex 5090 SFF
- Campus LAN
  - Specification: Gigabit Ethernet
- Server machine
  - Motherboard: MSI B550-A PRO
  - CPU: Ryzen 9 3900X
  - Memory: 4 x 32 GB TEAM DDR4 3200Mhz PC4-25600
  - Hypervisor: VMware ESXi-7.0
  - Strage: WD Red SN700 NVMe WDS400T1R0C
- Configuration server, Terminal server
  - CPU: 1 vCPU
  - Memory: 4 GB
- Simulation server
  - CPU: 1 vCPU
  - Memory: 1.5 GB

C. Measurement results

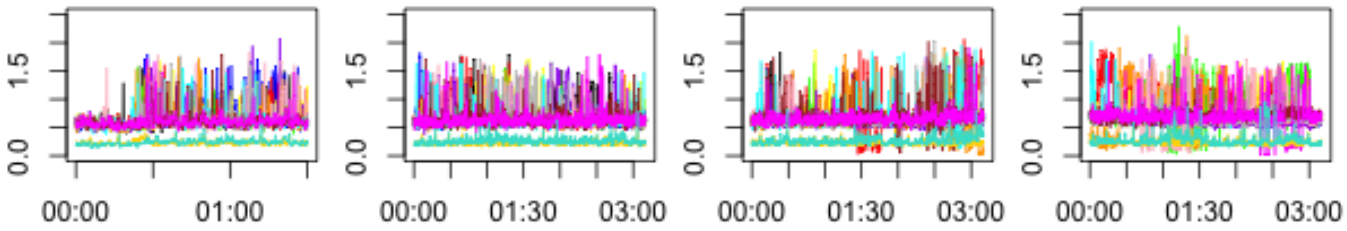
There were 44 students, divided into four groups of 12, 12, 12, and 8 people. During the class hours of all groups, esxtop [20] was executed in batch mode, and the usage of computational resources was measured every 10 seconds. Figures 5-14 show the usage of computational resources when one of the groups (12 people) attended the classes, and the other three groups showed roughly the same trend.

From Figure 6, the CPU used percentage exceeded 100% in some intervals. Figure 7 shows that the CPU ready time percentage during this period was below 2.5%. Also, the CPU co-stop time percentage (%CoStop) and CPU max limited percentage (%Max Limited) were 0 during the measurement



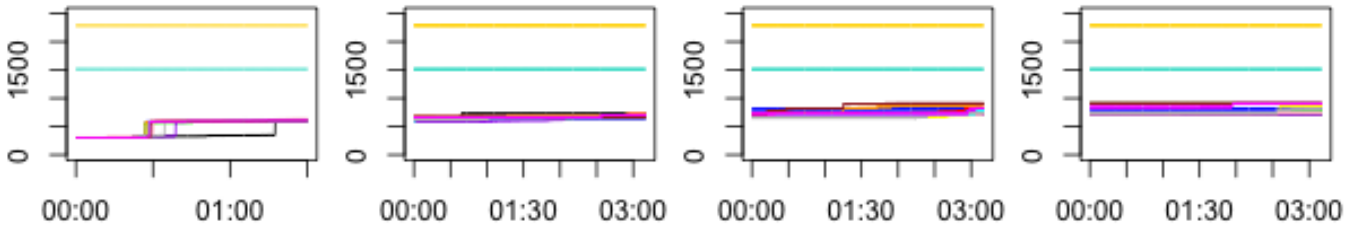
(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 6: %CPU used for the 14 servers, as measured by esxtop



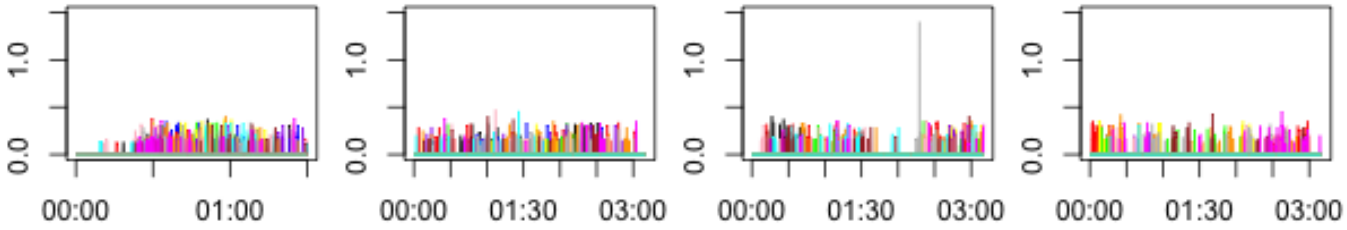
(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 7: %CPU ready for the 14 servers, as measured by esxtop



(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 8: Granted memory size in MBytes for the 14 servers, as measured by esxtop



(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

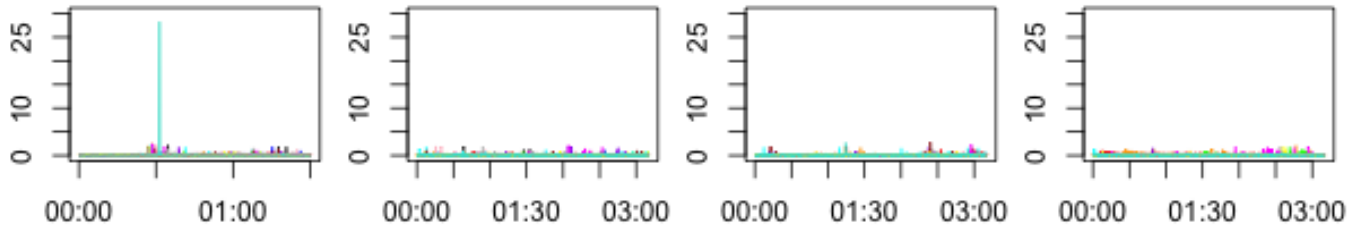
Fig. 9: Average milliseconds per read for disk operations on the 14 servers, as measured by esxtop

period. From the above, it can be concluded that the CPU resources were sufficient for the exercise.

From Figure 8, the configuration server actually used about 2300MB of memory, the terminal server used about 1500MB of memory, and each simulation server used about 900MB of memory. These are less than the amount of memory actually allocated. The reason for this is thought to be that VMWare's memory management technology was applied. Also, at this time, no memory swapping occurred. Therefore, it is unlikely that a small amount of memory was allocated due to a lack of memory. Thus, it can be concluded that the memory resources were sufficient for the exercise.

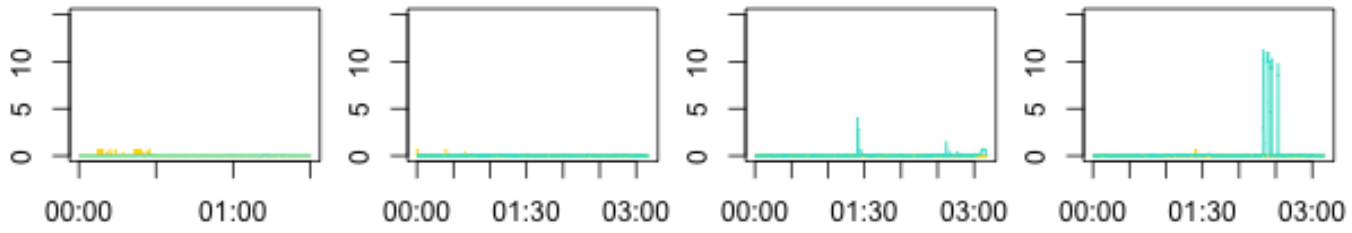
From Figures 9 and 10, spikes occurred in some intervals. Since they occurred for a very short time and only a few times, it can be inferred that the impact on the operating speed of the system was minor.

From Figures 11-14, spikes occurred in some intervals. However, no packet drops were observed between these measurement machines. Therefore, it can be concluded that the network resources were sufficient for the exercise.



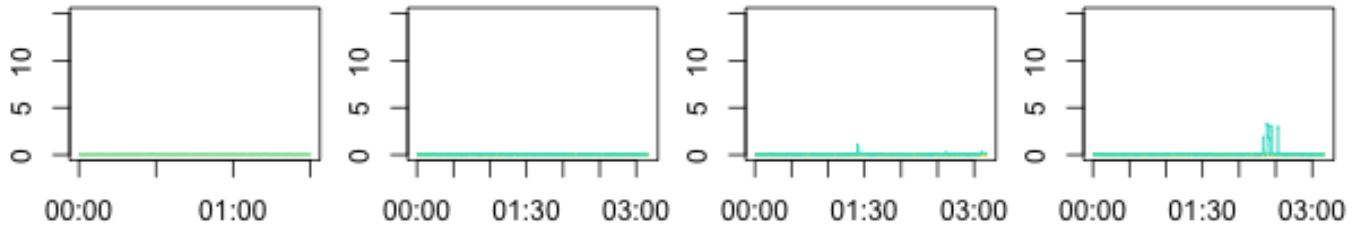
(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 10: Average milliseconds per write for disk operations on the 14 servers, as measured by esxtop



(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 11: Network throughput on the campus LAN: MBit's transmitted per second for the configuration and terminal servers, as measured by esxtop



(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 12: Network throughput on the campus LAN: MBit's received per second for the configuration and terminal servers, as measured by esxtop

D. Subject questionnaire

The purpose of this questionnaire is to determine whether the system's response speed hindered students' learning (understanding, memory, motivation, etc.), not their ability to complete tasks. The questionnaire form consisted of the question: "Did the system's response speed hinder your learning (understanding, memory, motivation, etc.)? If it did, please describe which responses, what aspects of learning (e.g., understanding), and to what extent it hindered your learning." The students received the questionnaire form before the first class, made notes on relevant incidents during the classes, and submitted their questionnaire responses along with their answers to the experimental tasks after the seventh class. Prior to receiving the questionnaire, students were informed that their responses would not affect their class grades and that the information would be used only for academic purposes with privacy protection measures in place.

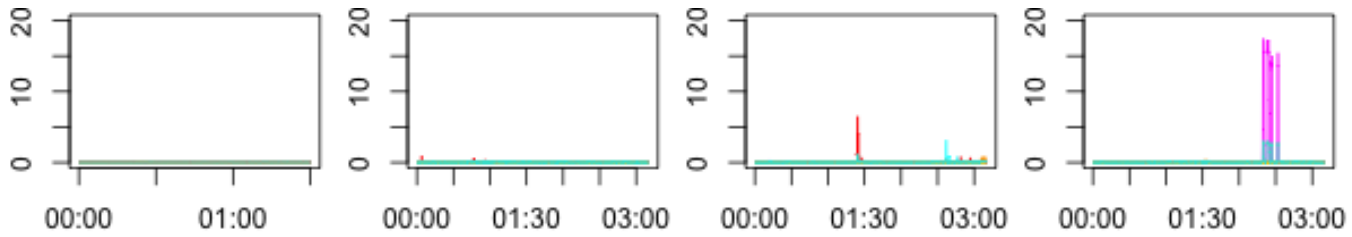
From the submission times, it appears that most students did not complete the exercise assignments within the class hours. The number of collected questionnaire responses was 44, and no events were reported that hindered learning.

E. Discussion

Since some tasks were performed outside of class hours, the load on the system was distributed. Therefore, it cannot be definitively stated that the system's response speed would not hinder learning even if a maximum of 12 students performed those tasks at roughly the same timing. However, from the measurement results described in Section V-C, it can be inferred that there is still ample room in the computational resources. Thus, it seems that the system's response speed would not hinder learning even if all students' tasks were performed within class hours.

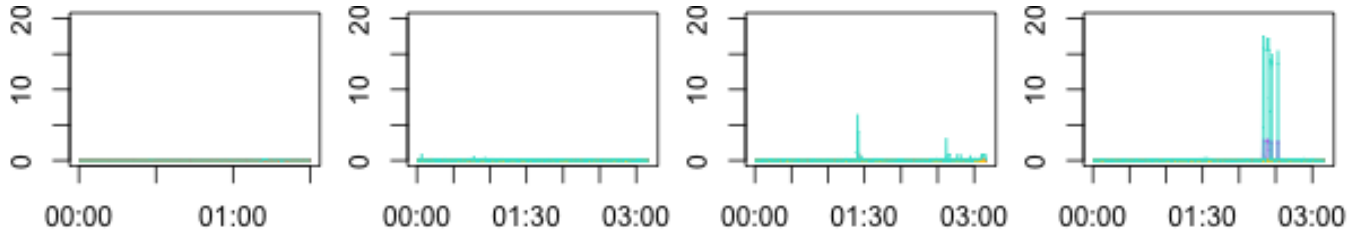
VI. Conclusion

In this paper, the author proposed instance naming and parameter allocation derived from device IDs and cable IDs to facilitate the correspondence between objects (devices and



(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 13: Network throughput on the backend network: MBit's transmitted per second for the 14 servers, as measured by esxtop



(a) First class (b) Second and third classes (c) Fourth and Fifth classes (d) Sixth and seventh classes

Fig. 14: Network throughput on the backend network: MBit's received per second for the 14 servers, as measured by esxtop

cables) manipulated by students and the instances and parameters that implement them. The device IDs and cable IDs used in this case are record IDs from a database (sqlite3), making this method easy to introduce into system implementations.

The author operated a practical system that implemented the main functions of LiNeS Cloud in the exercise. In seven 90-minute classes, up to 12 students used the system simultaneously. The computational load measured during this time was not significant enough to negatively impact responsiveness. Additionally, the subject questionnaire revealed that the responsiveness during this time did not hinder learning.

Future tasks include measuring the computational load on the guest OS of the virtual machines. In the current system implementation, the computational resources allocated to the virtual machines are generous for their intended use. By identifying the minimum necessary computational resource allocation, the system can be built at a lower cost, or more students can be accommodated in the exercise. Furthermore, we would like to increase the number of students using the system simultaneously and measure the impact on responsiveness in exercises that utilize all the functions of LiNeS Cloud.

Acknowledgments

This work was supported by JSPS KAKENHI Grant Numbers 20K12108 and 24K06257.

References

[1] N. Iguchi, "Development of a self-study and testing function for NetPowerLab, an IP networking practice system," *Int. J. Space-Based Situat. Comput.*, vol. 4, no. 3/4, pp. 175–183, 2014, doi: 10.1504/IJSSC.2014.066034.

[2] L. Xu, D. Huang, and W.-T. Tsa, "Cloud-Based Virtual Laboratory for Network Security Education," *IEEE Trans. Educ.*, vol. 57, no. 3, pp. 145–150, 2014, doi: 10.1109/TE.2013.2282285.

[3] CloudLab. Accessed: May 10, 2024. [Online]. Available: https://www.cloudlab.us/.

[4] Y. Deng, D. Huang, and C.-J. Chung, "ThoThLab: A personalized learning framework for CS hands-on projects," in *Proc. 2017 ACM SIGCSE Tech. Symp. Comput. Sci. Educ.*, Seattle, WA, USA, 2017, p. 706, doi: 10.1145/3017680.3022442.

[5] R. Beuran, D. Tang, C. Pham, K. Chinen, Y. Tan, and Y. Shinoda, "Integrated Framework for Hands-on Cybersecurity Training: CyTrONE," *Comput. Secur.*, vol. 78, pp. 43–59, 2018, doi: 10.1016/j.cose.2018.06.001.

[6] Y. Tateiwa, "LiNeS Cloud: A Web-Based Hands-On System for Network Security Classes with Intuitive and Seamless Operability and Light-Weight Responsiveness," *IEICE Trans. Inf. Syst.*, vol. E105.D, no. 9, pp. 1557–1567, 2022, doi: 10.1587/transinf.2021EDK0006.

[7] J. Dike, *User Mode Linux*. Upper Saddle River, NJ, USA: Prentice Hall, 2006.

[8] Node.js, Accessed: May 10, 2024. [Online]. Available: https://nodejs.org/en/.

[9] jQuery, Accessed: May 10, 2024. [Online]. Available: https://jquery.com/.

[10] SVG.js v3.0 — Home, Accessed: May 10, 2024. [Online]. Available: https://svgjs.dev/docs/3.0/.

[11] Xterm.js, Accessed: May 10, 2024. [Online]. Available: https://xtermjs.org/.

[12] Socket.IO, Accessed: May 10, 2024. [Online]. Available: https://socket.io/.

[13] OpenSSH, Accessed: May 10, 2024. [Online]. Available: https://www.openssh.com.

[14] VTun - Virtual Tunnels over TCP/IP networks, Accessed: May 10, 2024. [Online]. Available: http://vtun.sourceforge.net.

[15] XVFB, Accessed: May 10, 2024. [Online]. Available: https://www.x.org/releases/X11R7.6/doc/man/man1/Xvfb.1.xhtml.

[16] Xpra home page, Accessed: May 10, 2024. [Online]. Available: https://xpra.org.

[17] start-stop-daemon(8) — Linux manual page, Accessed: May 10, 2024. [Online]. Available: https://man7.org/linux/man-pages/man8/start-stop-daemon.8.html.

[18] screen(1) — Linux manual page, Accessed: May 10, 2024. [Online]. Available: https://man7.org/linux/man-pages/man1/screen.1.html.

[19] SQLite Home Page, Accessed: May 10, 2024. [Online]. Available: https://sqlite.org.

[20] Performance Monitoring Utilities: resxtop and esxtop, Accessed: May 10, 2024. [Online]. Available: https://docs.vmware.com/en/VMware-vSphere/7.0/com.vmware.vsphere.monitoring.doc/GUID-A31249BF-B5DC-455B-AFC7-7D0BBD6E37B6.html.