Journal of information and communication convergence engineering
http://jicce.org

J. Inf. Commun. Converg. Eng. 16(4): 235-241, Dec. 2018 Regular paper

Emulearner: Deep Learning Library for Utilizing Emulab

Gi-Beom Song and Man-Hee Lee*, *Member*, *KIICE*

Department of Computer Engineering, Hannam University, Daejeon 34430, Korea

Abstract

Recently, deep learning has been actively studied and applied in various fields even to novel writing and painting in ways we could not imagine before. A key feature is that high-performance computing device, especially CUDA-enabled GPU, supports this trend. Researchers who have difficulty accessing such systems fall behind in this fast-changing trend. In this study, we propose and implement a library called *Emulearner* that helps users to utilize Emulab with ease. Emulab is a research framework equipped with up to thousands of nodes developed by the University of Utah. To use Emulab nodes for deep learning requires a lot of human interactions, however. To solve this problem, *Emulearner* completely automates operations from authentication of Emulab log-in, node creation, configuration of deep learning to training. By installing *Emulearner* with a legitimate Emulab account, users can focus on their research on deep learning without hassle.

Index Terms: Deep Learning, Distributed Learning, Emulab

I. INTRODUCTION

Artificial intelligence is intelligence created by a machine whereby a computer enables human learning, thinking, and self-development so that it has a similar intelligence than a person [1]. In order to create such an artificial intelligence, artificial neural network is utilized to imitate the decision process of the human brain.

To create a sophisticated artificial neural network, the data is repeatedly computed according to the layer of the artificial neural network to improve the accuracy. For this reason, graphical processing units (GPUs) that perform simple operations with a larger number of cores show faster learning time than CPUs that perform sequential operations using a limited number of cores. However, if there are only GPUs without CUDA, one cannot but use CPU. Even in an environment that depends on the CPU, the learning time can be shortened by constructing the cluster environment and performing the learning in parallel. However, not everyone can build a parallel/distributed environment on his or her own.

To solve these problems, one can use cloud-based machine learning services like Amazon Web Service (AWS) Deep Learning AMI (Amazon Machine Image), Microsoft Azure Machine Learning, IBM Watson, and Google Cloud Machine Learning Engine [2-5]. AWS Deep Learning AMI helps users run various deep learning applications by providing many open source deep learning frameworks such as Apache MXnet, TensorFlow, Microsoft Cognitive Toolkit (CNTK), Caffe, Caffe2, Theano, Torch, Keras, etc. Azure Machine Learning uses the drag-and-drop approach, not necessary to typing commands. In IBM Watson equipped with many tools like TensorFlow, Keras, PyTorch, and Caffe, users can also compose their own neural networks via the drag-and-drop user interface. The biggest advantage of Google Cloud Machine Learning Engine is to train learning models by using other Google services such as Google Cloud Storage, Dataflow, and Datalab. In addition, by using hyperparameter tuning, the selection of important parameters that affect the

Received 29 June 2018, **Revised** 29 October 2018, **Accepted** 30 October 2018
***Corresponding Author** Man-Hee Lee (E-mail: manheelee@hnu.kr, Tel: +82-42-629-7677)
Department of Computer Engineering, Hannam University, 70, Hannam-ro, Daedeok-gu, Daejeon 34430, Korea.

Open Access https://doi.org/10.6109/jicce.2018.16.4.235 print ISSN: 2234-8255 online ISSN: 2234-8883

machine learning algorithm performance can be optimized. However, since these free functions of the services are limited, it costs a lot to solve real problems using all the functions and resources of the service.

To alleviate this burden, we propose to use Emulab, which was proposed and implemented at the University of Utah and is currently in use throughout the world as a research framework [6]. KREONet in Korea has also built and operated an Emulab, and it has been used in education and research [7]. Emulab provides on-demand allocation of the OS environment and network settings that users want using hundreds of PCs and high-performance network switches. Users who already have an account in Emulab can use the cluster by allocating resources for a short time without having to build their own physical clusters.

However, the use of Emulab in the area of deep learning needs a lot of human interactions because researchers need to configure all nodes separately for every experiment. It is almost impossible to use Emulab as is. To solve this problem, we propose and implement a user library, called *Emulearner*, completely automating operations from authentication of Emulab log-in, node creation, configuration of deep learning to training. That is, *Emulearner* removes any need to visit the Emulab web page that was the main interface between Emulab and its users. In our experiment, we asked a group of students to use Emulab as is and measured how long they spent in each step. On average, it takes about 81 seconds to configure one node through the original web interface. In addition, as the number of nodes increases, the total overhead grows proportionally, hampering the use of Emulab for deep learning area as is. However, by using *Emulearner*, the overhead becomes zero, successfully changing Emulab into a nice resource pool for deep learning.

This paper is organized as follows. Section II briefly describes TensorFlow. Section III explains in details how to use Emulab for deep learning. Section IV describes the implementation of *Emuleaner* and Section V compares the overhead of human interaction and library usage. Finally, Section VI presents our conclusions.

II. DISTRIBUTED LEARNING TECHNIQUES IN TENSORFLOW

TensorFlow is an open source machine learning library developed by the Google brain team for the purpose of studying machine learning and deep neural network [8, 9]. The TensorFlow library provides two distributed learning techniques called Data and Model parallelism. The Data parallelism divides training data and shares parameters. The Model replicas send to a parameter server the slope gained from each train. The parameter server that receives the slope data merges and updates the slope data, allowing the model replicas to update the new parameter data. The cycle in which data must be transferred between different devices is longer than the Model parallelism, and each replica can learn independently of other replicas.

In contrast, the Model parallelism method divides the parameters and shares the training data among nodes. In this model, the weight needed for computation during forward and back propagation must be communicated through the machine. Therefore, it is considered that the Model parallelism is suitable for an environment where data is exchanged between GPU devices rather than network-based data exchange. Therefore, in our study, we chose the Data parallelism because the KREONet Emulab exchanges data by using network.

III. USE OF EMULAB AS RESOURCE POOL FOR DEEP LEARNING

This section describes how to use Emulab for deep learning resource pool. Especially, we describe the procedure for using TensorFlow, which many people use for deep learning, in many nodes of Emulab. This procedure will be helpful for people who use TensorFlow in Emulab in the future, but it is also the reason why we developed the library we propose in this study.

The steps for using Emulab are as follows: obtain an Emulab account; request the necessary resources using the Emulab web GUI; connect to the reserved nodes, and run Tensorflow. For an Emulab account, a researcher needs to find a preferred Emulab system and sends a request to its administrator. After the request is granted, he or she will get an ID.

The procedure for requesting the necessary resources through the Emulab web GUI is as follows. First, the researcher visits the Emulab web page and starts by logging in with the ID and password. Emulab allocates resources in experiment units, so the researcher needs to create an experiment by clicking “Begin an Experiment” as shown in Fig. 1. The experiment creation page appears (Fig. 2). Then, it is necessary to select Project and enter the experiment name and a brief description. According to our measurements, researchers spend an average of 38.88 seconds from visiting Emulab web page to inputting the experiment’s description.

The next step is to click on the NetBuild GUI to draw the nodes with their operating systems, and the network topology. On the left, node and LAN are the resources to choose from (Fig. 3). The researcher will drag and drop as many nodes and LAN as he wants into the central pane. Then, when he clicks a component and drags it to another component, a direct link is formed between the two components, so he connects the components until he configures the desired topology. Learning performance may vary depending on the

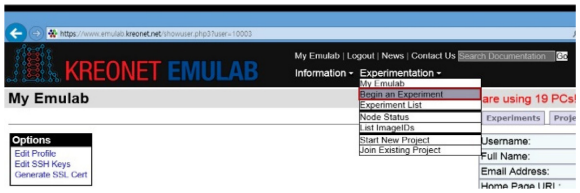

Fig. 1. Begin an Experiment page.

Fig. 2. Experiment creation page.

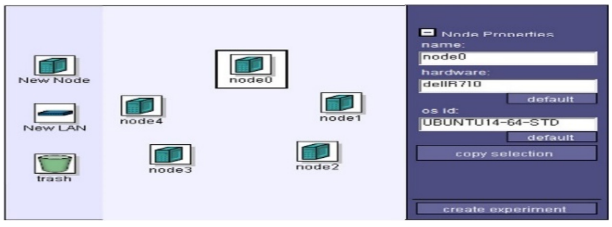

Fig. 3. Node and topology creation.

topology, but because the topic is out of the scope of this study, we only add nodes without a topology. IP is still assigned even if no topology is used, and it can be accessed internally or externally. According to our measurements, the time required to add one node is 1.19 seconds on average, and the total overhead increases proportionally as the number of nodes increases.

Then, when the researcher clicks on the bottom right "create experiment", Emulab finds available nodes, loads the operating system, and allocates new IPs. When the experiment creation is complete, the experiment becomes active and is now ready to access each node. Note that, as the number of active experiments continues to increase, all resources will be eventually assigned and new experiments will no longer be possible. To prevent this, Emulab recovers all resources that were assigned to the experiment kept idle for a certain period of time and changes the state of the experiment to inactive. Emulab calls it swap-out. Conversely, it is called swap-in to reallocate resources required for an inactive experiment by a user's request.

Reserved Nodes

Node ID	Name	Type	Default OSID	Node Status	Hours Idle[1]	Startup Status[2]	SSH URL	SSH mime	Console	Log
pc4	node11	dellR710	UBUNTU14-64-STD	up	0	none				
pc19	node8	dellR710	UBUNTU14-64-STD	up	0	none				
pc21	node9	dellR710	UBUNTU14-64-STD	up	0	none				
pc28	node12	dellR710	UBUNTU14-64-STD	up	0	none				
pc37	node10	dellR710	UBUNTU14-64-STD	up	0	none				

Fig. 4. Reserved node Information.

Node ID:	pc4
Virtual Name:	node11
Project:	DStesting12
Experiment:	learningtest
...	
Control Net IP:	203.250.172.4

Fig. 5. Node Information.

The reason why Emulab's swap-out and swap-in is related to this study is that each researcher must visit Emulab homepage using a web browser to log in and create a new experiment or reactivate an existing experiment at every deep learning experiment. This means that the human's configuration overhead occurs every time because the researcher needs a visual interaction with a keyboard or mouse for each experiment. This is a significant inconvenience to researchers when conducting many experiments simultaneously or sequentially.

The next step is to visit each node and prepare to install TensorFlow. For this, the researcher collects first newly-allocated IP information.

The researcher clicks on an active experiment's link to see a list of reserved nodes (Fig. 4). By clicking a node ID link, he can see the details of the selected node (Fig. 5). According to our measurements, it takes about 6.73 seconds to check the IP of one node and copy and paste it to an editor program, and the total overhead increases proportionally as the number of nodes increases.

Note that IP allocation cannot be predicted in advance because it depends on the IP available when generating experiments. If a swap-out experiment is swapped in, the number of nodes, the network topology, and the operating system version are the same, but the IP is newly allocated. In other words, the researcher must check the IP every time a new experiment is performed, and the IP information collection time increases in proportion to the number of nodes.

The next thing to do is to install the TensorFlow package on each node. Because Emulab loads a clean operating system image, all necessary tools must be installed by the researcher. According to the TensorFlow installation manual, it is necessary to install python-dev and python-pip [10, 11].

```
#apt-get -y install python-dev python-pip; pip install --upgrade
/proj/DStesting12/
setup_files/whl/tensorflow-1.4.0-cp27-none-linux_x86_64.whl
```

Fig. 6. TensorFlow installation command.

python-dev is a package containing the header files needed to build Python extensions, and python-pip is a package management system for managing and installing software packages written in Python. Fig. 6 shows the TensorFlow installation command.

The time required to log in to a node and enter the above setup command takes an average of 16.40 seconds. This is the time to enter the install command, not the installation time. Since this task has to be repeated in all nodes, the working time increases in proportion to the number of nodes. The important thing is that in Emulab's swap-out and swap-in process, all the information installed on each node disappears and the clean image is reloaded. The overhead of entering this script is repeated at the number of nodes at every experiment.

At this point, we found that in using Emulab for deep learning, researchers would not find it useful to reuse existing experiments that were swapped out. The reason for this is that swap-in has almost no advantages over a new creation of experiment, because all the information installed on each node disappears and the clean image is reloaded during swap-out and swap-in. Other studies that use Emulab often compare network topologies, so reconfiguring swap-in nodes may occasion less overhead than redrawing the network topology. However, for data parallelism in deep learning, topology is not important, only IP is necessary. Also, it is beneficial to visit the Emulab web page and use the available nodes as much as possible to get the training done quickly. Therefore, it is not necessary to use an experiment with a specific number of nodes. So, from now on, we assume that both researchers and libraries will create new experiments instead of using swap-out ones.

The data needed for experiment is uploaded to Emulab's file server only once and all nodes share it. Therefore, we do not consider the file upload overhead.

The next step for running TensorFlow adds the assigned IP and port number to the TensorFlow script. Port numbers can be any available port number, since these systems are clean OS, so any port greater than 1023 is not a big problem. For example, Fig. 7 is part of the TensorFlow script used in this study. Five nodes were allocated as one for parameter server and four nodes for worker nodes. Since this file is shared by all nodes, this editing is performed once for each experiment. The time to add one IP information using the editor is on average 4.45 seconds, and the editing time increases as the number of nodes increases.

Training starts by running the program edited above with

```
parameter_servers = ['203.250.172.1:2222']
workers = ['203.250.172.4:2223',
'203.250.172.6:2224','203.250.172.11:2224',
'203.250.172.16:2224']
```

Fig. 7. Tensorflow script editing.

```
[node 1] (parameter server)
#python CNN_Distributed_Learning_f84e0cad6bb6.py
--job_name="ps" --task_index=0
[node 2] (worker)
#python CNN_Distributed_Learning_f84e0cad6bb6.py
--job_name="worker" --task_index=0
[node 3] (worker)
#python CNN_Distributed_Learning_f84e0cad6bb6.py
--job_name="worker" --task_index=1
…
```

Fig. 8. Script running command.

the appropriate option as shown in Fig. 8. The time to visit each node and input the execution script is about 18.77 seconds, and the start overhead is proportional to the number of nodes since all nodes must be visited.

IV. EMULEARNER: LIBRARY UTILIZING EMULAB FOR DEEP LEARNING

In the previous section, we showed that there are various overheads to utilize Emulab for deep learning. Although detailed overhead analysis will be described in the next section, let us take a simple example. When we create an experiment with 100 nodes, it is an arduous labor to collect all the IPs of that number, connect to each node, install TensorFlow and run the script in all nodes. Therefore, the present state of Emulab is almost impossible to use for deep learning. This research proposes and implements *Emulearner*, a library that can change Emulab into a friendly deep learning resource pool. *Emulearner* acts like a normal library, but it can simply remove all the interaction overhead required to use Emulab. Table 1 shows the list of library methods.

Fig. 9 illustrates how easily a researcher utilizes Emulab by using *Emulearner*. He sets up a five-node experiment by calling *node_count(5)* and set the number of training to five by *epoch_count(5)*. Then, he configures training path, learning file path, id, and password by calling *training_path ("Q:\Exp\Data")*, *learning_file("default")* and *account("account", "password")*. Finally, calling *Learning()* starts training.

Now, we will explain the implementation of the library. The library eliminated all user interactions described in the previous section and almost everything is implemented in *Learning()*. The main part of *Learning()* source code is

Table 1. Units for magnetic properties

Method name	Description
Emulearner()	Constructs a new *Emulearner* to use Emulab
node_count(int **count**)	Set the **number** of nodes
epoch_count(int **count**)	Set the **number** of training
training_path(String **filepath**)	Set the **dataset** for learning
learning_file(String **filepath**)	Set the **learning script file**
account(String **user**, String **password**)	Set the **account** of Emulab for authentication
Learning()	Start the learning

```
def main():
…   Learning_test = Emulearner()
…   Learning_test.node_count(5)
…   Learning_test.epoch_count(5)
…   Learning_test.training_path("Q:\Exp\Data")
…   Learning_test.Learning_file("default")
…   Learning_test.account("account","password")
…   Learning_test.Learning()
```

Fig. 9. Library usage example.

```
def Learning(self):
… self.__Encryption()
… result = self.login(self.id,
… self.DecryptionAES(self.pw))
… if result is 'Failure':
…     print 'Login error!'
…     sys.exit(1)
… self.node_creating(self.nodes_for_learning)
… if self.nodes_for_learning == len(self.node_list):
…     print 'Nodes creation request successfully'
… else:
…     print 'An error occurs in node creation request'
…     sys.exit(1)
… self.checking_node_status()
… self.__file_trans_for_env()
… self.__file_validation()
… self.__compression()
… self.__files_transfer_for_learning()
… self.__setting_learning(self.
DecryptionAES(self.pw))
```

Fig. 10. *Learning()* source code.

shown in Fig. 10. *login()* automatically logs in to the Emulab web page [12]. nodes_for_learning has the value of the parameter passed to *node_count()*. In the present example, 5 is stored. *node_creating()* adds as many nodes as node_for_learning nodes to an experiment, generates an experiment creation request URL, and sends it to Emulab web server to perform node creation [12, 13]. The node_list stores the number of nodes that have completed the request. If this value is different from node_for_learning, the program is terminated.

checking_node_status() continually checks whether the creation of the requested experiment is complete. It checks every minute and if it fails to generate, it outputs the relevant log to the screen and exits the program. When the experiment is successfully created, it returns normal. Then, *__file_trans_for_env()* transfers the TensorFlow installation file. *__file_validation()* checks that the user's TensorFlow script is written in the format provided by the library. This library automatically modifies parameter server, worker, epoch number, and so on. Therefore, if the user script does not match the script format provided by the library, it cannot be modified automatically. If it is a normal format, the library compresses and transfers the script and data files.

The last function, *__setting_learning()*, modifies the Tensorflow script, install TensorFlow on each node, begin the script in all nodes, and download the final weight file when the experiment ends.

V. EXPERIMENT

This section shows how much user time our library can save. We obtained the user interaction overhead in Section III by teaching ten graduate students how to use Emulab, then repeatedly experimenting ten times and measuring the average time spent in each step. We measured the average of the library's overhead by repeating ten times. Due to the availability of KREONet Emulab, we have actually created up to ten nodes. We designed a formula to predict the time required to generate a larger number of nodes.

Table 2 compares the step-by-step overhead between web UI and *Emulearner* automation. Web UI column's shows how long it will take for a researcher to utilize Emulab through web UI while *Emulearner* automation column to use Emulab through *Emulearner*. The proportional column shows whether overhead is added per experiment or proportional to the number of nodes, and a variable name is assigned to each overhead in order to express the total overhead through formulas. Node creation and package installation are the same time spent in user interaction and the library.

Interestingly, when we created one to ten nodes, there was little difference in node creation time. We assume that Emulab is configured to load multiple nodes in parallel. So node creation time can be said to be proportionate to experiment, not the number of nodes.

The reason for entering the package installation command and TensorFlow command for each node is slightly longer than expected is that the overhead of logging in to each node using software such as putty is included [14]. Script editing

Table 2. User and library overhead

Step	Web UI (s)	Emulearner automation (s)	Proportional	Variable
Emulab log-in & experiment description	33.88	5.02	Experiment	LE
Node addition	1.18	0.30	Node	NA
Node creation	103.40	103.40	Experiment	NC
IP check	6.73	0.39	Node	IP
Package installation command	16.40	0.12	Node	PC
Package installation	369.19	369.19	Node	PI
Script editing	4.45	0.07	Node	SE
TensorFlow command	18.77	0.12	Node	TC
Total overhead	81.41	6.02	-	TO
Total preparation	554.00	478.61	-	TP

overhead is a per node value. That is, if a researcher enters ten IPs, we expect the editing will take about 4.45 seconds.

The total overhead value is the net overhead time excluding node creation and package installation time, and the total preparation is the total time taken before the start of the learning. However, this time is for configuring only one node's environment. If the number of nodes increases, the total overhead and total preparation time increase because the proportional overhead increases linearly.

By using *Emulearner*, the total overhead was reduced from 81.41 seconds to 6.02 seconds by 92.6%, and the total preparation decreased by 13.6% from 554 seconds to 478.61 seconds.

Based on the experimental results described above, we created formulas to predict the total overhead and the total preparation. We grouped steps that are proportional to the number of nodes, n, and then, we obtained the coefficient of n by summing the times required in the proportional step. All the variables in the formulas are defined in variable column of Table 2.

$$\cdot TO = LE + n*(NA + IP + PC + SE + TC) \quad (1)$$

$$\cdot TP = LE + NC + n*(NA + IP + PC + PI + SE + TC) \quad (2)$$

The new formulas with real numbers are shown below.

$$\cdot TP(user) = 137.18 + n*416.72 \quad (3)$$

$$\cdot TP(lib) = 108.42 + n*370.19 \quad (4)$$

$$\cdot TO(user) = 33.88 + n*47.53 \quad (5)$$

$$\cdot TP(lib) = 5.02 + n*1.02 \quad (6)$$

The overhead and preparation time required to create up to 100 nodes are shown in Fig. 11. It seems that TO(lib) does not increase even if the number of nodes increases. This

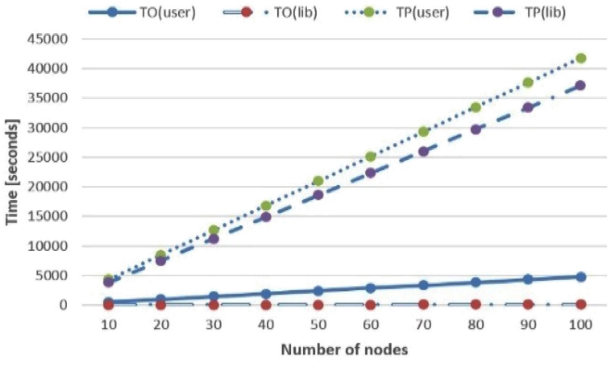

Fig. 11. Total overhead and preparation time prediction.

means that there is almost no extra overhead all the time. More importantly, TO(user) is the overhead that users need to spend, whereas TO(lib) is the overhead that the library spends for automation. That is, through *Emulearner*, human researchers are completely free from Emulab manipulation time.

VI. CONCLUSION

Emulab is a research framework actively used in various studies. We proposed an efficient method to use Emulab for deep learning. Using the existing web interface, each experiment required approximately 47 seconds of user overhead per node. This means that it is difficult to use Emulab for deep learning with many nodes.

To solve this problem, we proposed and implemented a library, called *Emulearner*, to automate account authentication, node creation, node configuration, and training. This library eliminates user intervention and successfully transforms Emulab into a convenient resource pool for deep learning. But our library has a limitation for various neural networks because current library is utilizing the only one

neural network called like convolutional neural network [15]. In addition, this library is designed to work with KRO-Net Emulab, so modest customization is required for other Emulab sites.

For the future research, we will provide various neural networks that a user can select the neural network users want. In addition, we are developing libraries that can be used in various fields such as security.

REFERENCES

[1] J. H. Park, "Understand of artificial intelligence: approach by cognitive science," *Asia-pacific Journal of Multimedia Services Convergent with Art, Humanities, and Sociology*, vol. 6, no. 10, pp. 539-547, 2016. DOI: 10.14257/AJMAHS.2016.10.44.

[2] Amazon AWS Deep Learning AMI [Internet], Available: https://aws.amazon.com/ko/machine-learning/amis/.

[3] Microsoft Azure Machine Learning Studio [Internet], Available: https://azure.microsoft.com/en-us/services/machine-learning-studio/.

[4] IBM Deep Learning [Internet], Available: https://www.ibm.com/cloud/deep-learning.

[5] Google Cloud Machine Learning Engine [Internet], Available: https://cloud.google.com/ml-engine/.

[6] M. H. Lee and W. J. Seok, "Research on the trend of utilizing Emulab as cyber security research framework," *Journal of the Korea Institute of Information Security and Cryptology*, vol. 23, no. 6, pp. 1169-1180, 2013. DOI: 10.13089/JKIISC.2013.23.6.1169.

[7] KREONet Emulab [Internet], Available: https://www.emulab.kreonet.net/.

[8] N. Gupta, "Artificial neural network," *Network and Complex Systems*, vol. 3, no. 1, pp. 24-28, 2013.

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, et al., "TensorFlow: a system for large-scale machine learning," in *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Savannah, GA, pp. 265-283, 2016.

[10] Ubuntu package, "python-dev," [Internet], Available: https://packages.ubuntu.com/search?keywords=python-dev.

[11] Ubuntu package, "python-pip," [Internet], Available: https://packages.ubuntu.com/search?keywords=python-pip.

[12] Python Software Foundation, "urllib2," [Internet], Available: https://docs.python.org/2/library/urllib2.html

[13] Python Software Foundation, "mechanize," [Internet], Available: https://pypi.org/project/mechanize/.

[14] PuTTY: a free SSH and telnet client for Windows [Internet], Available: https://www.putty.org/.

[15] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, et al., "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354-377, 2018. DOI: 10.1016/j.patcog.2017.10.013.

Gi-Beom Song

received the B.E. degree in computer engineering from Hannam University, Daejeon, Korea in 2017. Where he is now a master course student. His current research interests include information security about virtualization utilizing Emulab, detecting evasive malware about virtual environment.

Man-Hee Lee

received the B.E. and M.E. degrees in Computer Engineering Department from Kyungpook National University, Korea in 1995 and 1997, respectively, and the Ph.D. degree from the Department of Computer Science and Engineering Texas A&M University, in 2008. He is currently an associative professor at Hannam University in Korea, His interests include system/network security, AI security, secure computing architecture, secure cluster computing.