Mobile Device-centric Approach for Identifying Problem Spot in Network using Deep Learning

Woonghee Lee, Joon Yeop Lee, and Hwangnam Kim

Abstract: **These days, mobile devices usually have multiple network interfaces and there are many usable access networks around the devices. To utilize a wide range of network options properly and make decisions more intelligently, the mobile devices should be able to understand networks' situations autonomously. The current mobile devices have powerful computing power and are able to collect various network information, and people nowadays almost always carry their mobile devices. Thus, the mobile devices can be utilized to figure out practical quality of service/experience and infer the network situation/context. However, networks have become not only larger but also more complex and dynamic than in the past, so it is hard to devise models, algorithms, or system platforms for mobile devices to understand such complex and diverse networks. To overcome this limitation, we leverage deep learning to devise a mobile device-centric approach to identifying problem spot having the most likely cause of network quality degradation, *MoNPI*. By using *MoNPI*, mobile devices are able to identify the network problem spot, which is like a black box to end nodes heretofore. Mobile devices with *MoNPI* are able to understand networks' situations and thus take a more proper action.**

Index Terms: **Deep learning, mobile, network, problem spot identification, transmission control protocol**

I. INTRODUCTION

SUPPORTED by the advancements of communication and hardware technologies, today's mobile devices usually have multiple network interfaces, and there are various usable networks around the devices [1]. In addition, there are an enormous number of Internet applications, and many Internet service providers (ISPs) provide their own network infrastructures to users [2]. In other words, the mobile users have a wide range of network options than they have ever had [3]. To utilize a wide range of network options properly, the mobile devices should be able to understand networks' situations autonomously, so that the devices can make decisions more intelligently depending on the situation. As shown in Fig. 1, in the transmission between the mobile device and the server, there are various factors that degrade communication performance. In addition, there is a possibility that problems may occur in any router, base station, and server that are involved in data transmission. However, the mobile device cannot identify what and where the transmission problem occurs.

Manuscript received December 19, 2020; revised April 22, 2020; approved for publication by Namyoon Lee, guest editor, May 1, 2020.

The authors gratefully acknowledge the support from Nano UAV Intelligence Systems Research Laboratory at Kwangwoon University, originally funded by Defense Acquisition Program Administration (DAPA) and Agency for Defense Development (ADD).

The authors are with the School of Electrical Engineering, Korea University, email: {tgorevenge, charon7, hnkim}@korea.ac.kr.

H. Kim is the corresponding author.

W. Lee and J. Y. Lee equally contributed to this work.

Digital Object Identifier: 10.1109/JCN.2020.000008

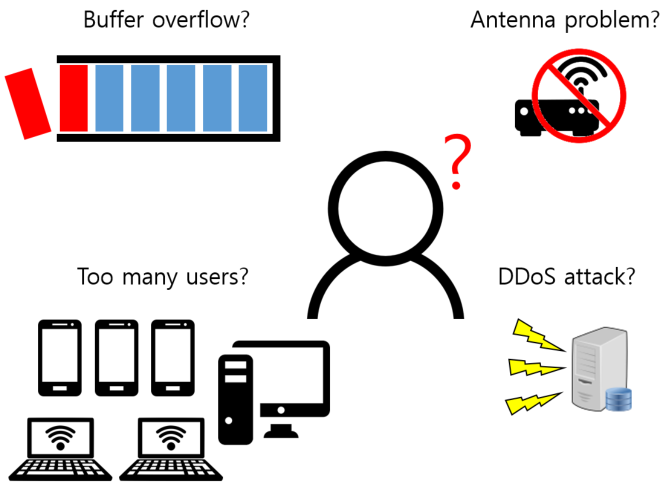

Fig. 1. Various causes of transmission degradation.

In the conventional method, since the the mobile device does not know where the problem occurs in the network, it should repeat data retransmissions or meaningless reconnection attempts [4]. By exactly knowing where the problem occurs, mobile device can appropriately resolve network transmission problem. For example, if there is a problem with the mobile device, the device can notify the problem to the user. Also, the mobile device is able to change the data transmission path by recognizing that there is a problem on the router, or request a service provider to identify the problem if there is a problem with the server. Nowadays, the mobile devices have powerful computing power and are able to collect various network information [5]. In addition, people nowadays almost always carry their mobile devices [6]. Thus, the mobile devices can be utilized to figure out practical quality of service/experience (QoS/QoE) and infer the network situation/context around the people. However, it is not trivial to devise models, algorithms, or system platforms for mobile devices to understand complex and diverse networks.

To overcome these limitations, we propose a mobile device-centric approach to identify network problem through machine learning, *MoNPI*. As far as we know, this work is the first attempt to devise a mobile device-centric scheme which identifies the problem spot having the most likely cause of network quality degradation by leveraging deep learning. *MoNPI*'s goal is to provide information of problem spot that enables mobile devices to flexibly cope with various network problems. The contributions of this paper are summarized as follows.

1229-2370/19/$10.00 © 2020 KICS

- We show that, by using *MoNPI*, mobile devices are autonomously able to understand the network's situation, which is like a black box to end nodes heretofore.
- *MoNPI* can detect network problems via ordinary data transmissions without requiring additional protocols or equipment installation.
- We leverage deep learning to devise a mobile device-centric scheme, whereas most of the related work utilized deep learning to devise solutions for huge systems or infrastructures.
- We conducted various analysis and experiments, and the results demonstrate that a mobile device using *MoNPI* is able to identify the problem spot having the most likely cause of network quality degradation.

The remainder of this paper is organized as follows. Firstly, we introduce related work and describe *MoNPI*'s novelties and advantages compared to the related work in Section II. We explain challenges and problem situations considered in this paper in Section III. After that, in Section IV, we explain the design of *MoNPI*. In Section V, we give an explanation about data collection for learning in *MoNPI*. In Section VI, we explain analysis and experiments, and evaluate the performance of *MoNPI*. Section VII describes the discussion, and Section VIII finally concludes this paper.

II. RELATED WORK

In this section, we introduce various researches relevant to our research. After that, we describe novelties and advantages of *MoNPI* compared to the related work.

Deep learning has been utilized mostly in image recognition, computer vision, speech recognition, etc. However, recently, various libraries and platforms for deep learning have been proposed, and many researchers have tried to use deep learning for improving networking and communication. To predict the amount of data traffic is important for ISPs to operate network infrastructures efficiently and conduct resource allocation properly. Poupart *et al.* described how to use several learning techniques to estimate flow sizes based on information extracted from the first few packets of each flow [7]. In addition to the amount of data traffic, the characteristics of data traffic should also be considered to control and manage network infrastructures properly. Arzani *et al.* conducted a research on performance problems in data centers [8]. They presented a diagnostic tool to identify causes of performance problems in data centers. Z. Wang utilized neural network and deep learning to propose a method for protocol classification and identification [9]. According to the results, the proposed method works well on the applications. As explained above, deep learning has been utilized to predict the amount of data traffic and recognize the data characteristics. In addition, since deep learning method can identify the unique situation of the network, it is also utilized to identify security issues in networks [10]–[14]. On the other hand, there are studies that use deep learning to find the spot where the power transmission has a problem [15], or to determine the appearance of drones [16]. As these studies show, deep learning is being used to identify various problems.

Including the aforementioned researches, most of the work using deep learning were proposed mostly for large-scaled systems, such as data centers, which are controlled centrally. However, recently, there are some researches which tried to utilize deep learning to identify network problems in user's devices. Nguyen *et al.* proposed a framework that leverages a deep learning approach to detect cyber attacks in mobile cloud environment [17]. Similarly, Yuan *et al.* proposed a machine learning based method that utilizes features extracted from analysis of Android application for malware detection [18]. Este *et al.* proposed traffic classification method based on support vector machine (SVM) [19]. Authors showed that traffic can be classified using only the transmission control protocol (TCP) information from the user. Tan *et al.* proposed a method to identify unauthorized and anomalous network services based on the single connection characteristic [20]. The authors analyzed the TCP traffic information using a modified neural network. As a result, this method identified security issues with high probability.

As shown in these studies, it is possible to identify that a problem has occurred in the network with only information obtained from the user's devices. However, most of the studies only identify whether a problem occurred. Also, there was a lack of research to identify the problem spot. For example, when a congestion problem occurs due to high traffic, it is necessary to determine where this problem occurred among access point (AP), server, or core networks to solve the problem. In order to solve the network problem, it is necessary to be able to identify both what and where problem occurred. Also, mobile devices are able to collect various information around users, so the mobile devices can be effectively utilized to understand the users' communication/network situation or context.

Motivated by these, we leverage deep learning to devise a mobile device-centric network problem spot identification. Compared to the related work, our research has novelties and advantages in several perspectives. Firstly, most existing researches, focusing on recognizing network problems, devised infrastructure-centric methods. However, *MoNPI* is the mobile device-centric method, and this paper shows that it is possible to detect the problem spot in network from the standpoint of end node. Secondly, by leveraging deep learning, *MoNPI* is able to remotely recognize the network situation, so that the device using *MoNPI* can make decisions more intelligently. Thirdly, we evaluate the performance of *MoNPI* using not only simulation but also empirical data, which means that *MoNPI* is verified practically.

III. PROBLEM STATEMENT

In this section, we explain why it is hard for mobile devices to understand network situations and identify the problem spot having the most likely cause of network quality degradation. After that, we explain the problem situations considered in this paper.

A. Challenges

As we explained in Section II, many researches utilized deep learning for networks, but most of them focused on security issues in networks [10]–[14], [21]–[23]. Few researches tried to identify network problems by using deep learning. Among

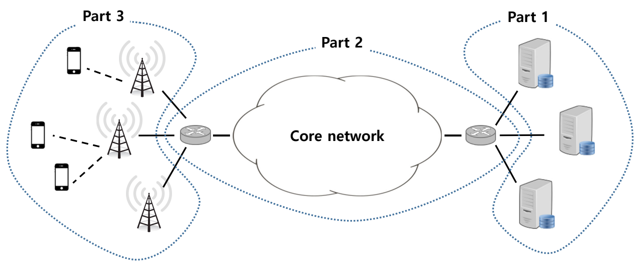

Fig. 2. The typical topology of today's network in a mobile environment.

them, some researches [7], [8] tried to utilize deep learning to identify performance problems, but the research's target system is a data center which uses a centralized control. In fact, without using deep learning, most of components in data centers can be monitored continuously and controlled in real time using advanced network technologies, such as software defined network (SDN) and network function virtualization (NFV). However, unlike data centers, it is challenging for end nodes, such as mobile devices, to understand network situations and identify the problem spot having the most likely cause of network quality degradation. The major causes of this are as follows.

1. **Limited information of network**: Without cooperation of ISPs or infrastructures, the mobile device should gather information measured only by the end node itself. Thus, using such limited information, it is hard for the device to guess the network situation correctly.
2. **Different characteristics of measured parameters**: In the end node, different parameters are collected depending on the layer. For instance, in physical and data link layer, information mostly about communications can be collected, whereas parameters collected in network and transport layer have more to do with networks. These parameters have very different characteristics, so they should be preprocessed to obtain meaningful information.
3. **Many factors influencing the network**: In data communications, data are transmitted through many intermediate nodes between the device and server. Thus, there are a lot of factors which can influence the network, as well as the state of the device and server.
4. **Huge size and complexity of network**: Modeling is one of the widely used methods to represent a certain system or phenomenon. However, today's networks used by mobile devices have complex and huge topology, and there are a lot of factors to be considered. Thus, it is practically impossible to model the entire network with a well-defined mathematical/analytical/systematic method.

Deep learning is able to deal with complicated problems, which is a significant advantage. Motivated by this, we leverage deep learning to overcome the above challenges and devise a mobile device-centric network problem spot identification.

B. Problem Situations

In this paper, we consider two representative wireless/mobile networks, wireless local area network (WLAN) and cellular network, which are the most widely used networks these days, as target networks of *MoNPI*. Fig. 2 shows the typical topology of networks in a mobile environment. When a mobile device downloads data from a server, the data transmitted by the server is delivered to the mobile device through the core network. This procedure can be divided into three parts as shown in the figure.

- The first part is that the data is transmitted from the server.
- The second part is that the transmitted data is delivered to the cell tower or the AP through the core network.
- The last part is that the data is finally delivered to the device from the cell tower or the AP.

In this situation, the poor performance due to the abnormal state of the server indicates the problem in the part 1. Similarly, the part 2's problem means that there is a problem in the core network provided by the ISP. In addition, a problem can occur in the part 3 when the cell tower is overloaded due to a large crowd around the tower. If the mobile device is able to identify the problem spot in network, it is possible for the device to take a more proper action, and we discuss the possible actions in Section VII.

IV. DESIGN OF *MONPI*

In this section, we firstly describe the overall design of *MoNPI*. Then, we explain how to measure parameters used in *MoNPI* and process them. Lastly, we give detailed explanations about the learning model of *MoNPI*.

A. Overall Design

Fig. 3 shows the overall design of *MoNPI*. As shown in the figure, *MoNPI* in a mobile device measures various parameters. After that, *MoNPI* processed collected data in one time window to extract information and make an input vector, I, composed of n values. W_{input} is the length of time window, and the step length of time window, δ, means the time between consecutive windows. *MoNPI* serves the vector as the input into the deep learning model, which is already trained. Using the input vector, the model returns the result which indicates where the problem is expected to occur. The above processes are performed repeatedly, and *MoNPI* infers the problem spot by considering one or more results together.

B. Parameters Used for MoNPI

In mobile devices, various parameters can be measured in different layers, such as physical, data link, network, transport, and application layer. To consider the network situation, *MoNPI* utilizes many parameters except parameters collected in application layer because collecting the parameters in application layer requires the modification of application program. Table 1 lists the parameters used for *MoNPI*.

1. **Parameters collected in physical and data link layer**: In physical and data link layer, *MoNPI* is able to obtain information mostly about communications, such as throughput, receive (RX) drop count, and frame receive delay of long term evolution (LTE). From these values, *MoNPI* is able to guess the communication situation around the device and the communication quality of the last one-hop, the link between the device and the cell tower or the AP.
2. **Parameters collected in network and transport layer**: *MoNPI* utilizes parameters in network and transport layer,

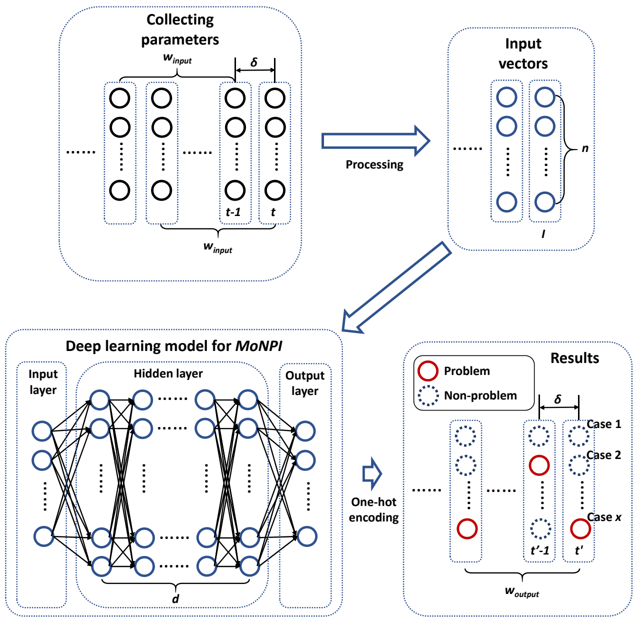

Fig. 3. The overall design of *MoNPI*.

Table 1. Communication and network parameters used for *MoNPI*.

Layer	Parameters
Physical and data link	Throughput RX drop count Frame receive delay of LTE
Network and transport	Congestion window size Received window size Packet inter-arrival time RTT

such as congestion window size (CWND), received window size (RWND), packet inter-arrival time, and round trip time (RTT). These parameters represent characteristics of end-to-end connection between the mobile device and server. Moreover, the parameters are affected by the state of core network, so *MoNPI* utilizes these values to understand the state of end-to-end network.

C. Preprocessing

MoNPI utilizes diverse parameters which have different characteristics, so the preprocessing is essential to obtain meaningful information from such various parameters. Among the parameters' characteristics, scales or ranges are most significantly different. For instance, in general, the average value of congestion window size is thousands of times larger than that of RX drop count. If values of one specific parameter are much larger than those of other parameters, the learning model is excessively influenced by the specific parameter, so the model is trained improperly. To avoid this problem, *MoNPI* conducts the normalization for each parameter. If there are some collected values of a certain parameter, x, the ith value, x_i, is normalized as follows:

$$x'_i = (x_i - x_{\min})/(x_{\max} - x_{\min}), \tag{1}$$

where $x_{\min}$ and $x_{\max}$ are the minimum and maximum value among values of x, respectively. x'_i indicates the normalized value of x_i.

After the normalization, *MoNPI* calculates each parameter's mean, average, standard deviation, and maximum values to extract various characteristics from the collected parameters. While calculating the above values, the time window length, W_{input}, is important because the calculated values are changed depending on the time window length. To reflect the characteristics of end-to-end connection, W_{input} should be larger than one RTT. Thus, we set W_{input} to 500 ms since RTT on mobile usually ranges between 100 and 1000 ms according to [24].

D. Deep Learning Model for MoNPI

To properly configure learning model is important to accurately identify problem spot having the most likely cause of poor network quality. We use various techniques of deep learning to configure the learning model of *MoNPI*.

1. **Initializer**: Each node in hidden layer has a weight, and this value is initialized before starting learning. Many researchers conducted researches on initialization and proved that determining the initial weight value properly is important to improve the learning performance. Xavier initializer [25] automatically determines the scale of initialization based on the number of input and output nodes, and it is one of the most widely used initializers in deep learning. The algorithm used in Xavier initializer is rather simple compared to other initializers, such as LSUV [26] and MSRA [27], so *MoNPI* uses Xavier initializer.
2. **Optimizer**: Similar to the initializer, to choose a proper optimizer is important to achieve better performance. The Adam optimizer [28] was recently announced, and it is an extension to stochastic gradient descent. This optimizer is a popular algorithm in deep learning because it achieves good results fast. Empirical results demonstrate that Adam optimizer works well in practice and compares favorably to other stochastic optimization methods. Thus, we utilize the Adam optimizer for *MoNPI*.
3. **Activator**: Each node in neural networks needs an activation function which defines the output of the node. Among various activators, rectified linear unit (ReLU) has been widely used because ReLU usually proves faster to train than standard sigmoid units, which is the most popular activation function in the past [29]. Thus, ReLU is used for nodes in learning model of *MoNPI*.
4. **Techniques for preventing overfitting**: In *MoNPI*, the dropout technique [30] is used to prevent the overfitting problem, which is the production of an analysis that corresponds too exactly to a particular set of data. When using dropout, nodes are randomly dropped along with their connections from the neural network during training, which prevents nodes from co-adapting too much. The probability of retaining a node, p, controls the intensity of dropout, and the lower value of p means more dropout. Typical values of p for learning are in the range 0.5 to 0.8, and we set p of *MoNPI* to 0.7 during the training. The detailed explanation about determining this value will be given in Section VI.A.2.

5. **Softmax and One-hot encoding**: *MoNPI* analyzes the collected information and returns the result which indicates where the problem is expected to occur. The softmax function is for a generalization of the logistic function, and the output of softmax function can be used to represent a categorical distribution. In addition, when there are some values, one-hot encoding picks the largest value, and then sets the value to 1. At the same time, the remaining values are changed to 0. Thus, *MoNPI* utilizes softmax function with one-hot encoding to find the case which is most likely.

E. Identification Method Using Consecutive Results

Using one input vector, the model generates one result, and *MoNPI* is able to infer the problem spot according to the result. However, parameters related to communication or network can fluctuate in a split second, which may lead to the misjudgment of *MoNPI*. Thus, by default, *MoNPI* uses multiple results together to infer the problem spot more accurately. w_{output} means the number of results used to infer the problem spot as shown in Fig. 3. *MoNPI* accumulates w_{output} results, and then picks the problem spot that the learning model pointed out most frequently in a short period of time. With this method, MoNPI can obtain information about the tendency of the network from multiple packets. Therefore, MoNPI can more accurately grasp the meanings of network parameters and reduce the influence of exceptional parameters obtained by temporary network fluctuation. We will explain an evaluation on the effectiveness of this method and show the result of evaluation in Section VI.E.

V. DATA COLLECTION

This section explains the data collection to make data sets for *MoNPI*. We explain the simulation data collection firstly, and then give an explanation about the collection of empirical data. It should be noted that we collected only data which can be obtained from mobile devices since *MoNPI* is a device-centric approach. In addition, we collected data by selecting frequently used scenarios and fault cases that actually occur in real world.

A. Simulation Data Collection

Using a simulator, we are able to control various factors relevant to server and network equipment, so various situations can be constructed in the simulation. We used a network simulator, ns-3 [31], and considered LTE and Wi-Fi networks which are most widely used by mobile users these days.

A.1 LTE Network Scenario

In the first simulation, we collected data while performing data communication using the LTE network. We constructed the LTE network similar to the topology shown in Fig. 2. A user equipment (UE) had a wireless connection with evolved node B (eNB) which provided LTE communication, and the UE communicated with the server at a data rate of 25 Mbits/s. We used Verizon wireless network characteristics from [32]. The eNB forwarded data to packet data network (PDN) gateway by wire, and then the PDN gateway transmitted data to the server through the core network. This is a general topology of today's LTE networks. With this topology, we considered three cases as follows. Each fault case was selected on the assumption of a general situation that would occur in a practical situation.

- Normal case: As its name says, this case represents the normal situation without any problem of data communication. Data transmission was performed with error rate of 0.01% or less.
- Fault case 1: The server suffers from packet drops caused by receiving problem of medium access contention (MAC) layer.
- Fault case 2: There is congestion caused by traffic from multiple UEs to the server. In this case, we experimented with 6 UEs.

A.2 Wi-Fi Network Scenario

In the second simulation, we tried to construct a general topology of Wi-Fi network where data was transmitted through an AP in Wi-Fi network environment. In this simulation, mobile users were connected to the AP, which was connected to the server through the Internet by wire. To emulate practical situations, users were configured to walk randomly within the communication range of the AP. Using this topology, three cases that can actually occur in reality were considered as follows.

- Normal case: This case represents the normal situation without any problem of data communication with an error rate of 0.01% or less.
- Fault case 1: Unlike LTE, many APs are not controlled centrally in general, so there is a high probability that communication or network problems occur in the APs. In this case, we emulated the situation where the throughput of data communication deteriorates due to the problem of AP.
- Fault case 2: In this case, the throughput of data communication is degraded because of a problem in the server receiving data.

The UE using TCP communicated with the server at a rate of 25 Mbits/s for 60 s in all the simulations. While running simulations, we measured the parameters explained in Section IV.B.

B. Empirical Data Collection

Empirical data is necessary to verify the performance of *MoNPI* in practical situations. We made a scenario where an Android smartphone (Galaxy Nexus) was connected to an AP through Wi-Fi, and the smartphone transmitted data to the server (linux PC) with Iperf application [33]. The AP was connected to the server by wire, and the network topology was similar to the simulation in Section V.A.2.

In addition to the upload scenario explained above, we made an additional scenario where the smartphone downloaded files through the Internet. In these two scenarios (upload/download), four cases were considered as follows.

- Normal case: This case represents the normal situation without any problem of data communication.
- Fault case 1: DDoS is an attack that an attacker maliciously generates heavy traffic such as UDP flooding to cause heavy congestion in router or server [34]. To emulate the DDoS attack, a device that imitated an attacker transmitted a huge amount of UDP packets to make the AP congested.
- Fault case 2: 5 smartphones using TCP simultaneously transmitted data to the server.

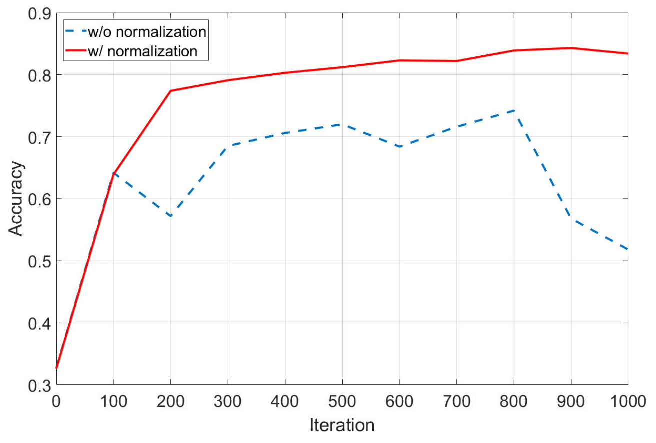

Fig. 4. The analysis on the effect of normalization.

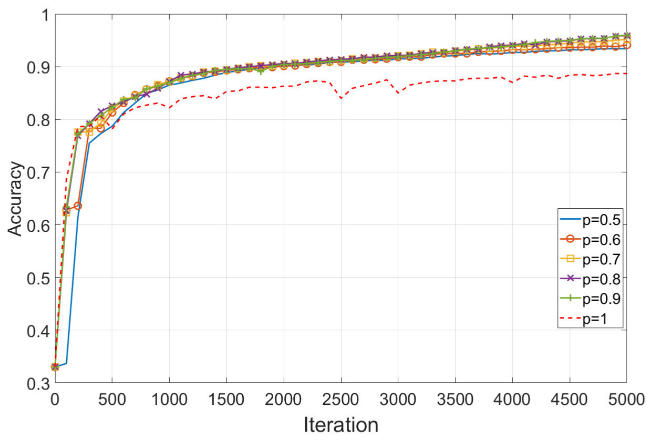

Fig. 5. The accuracy results depending on the probability of retaining a node and the number of iterations.

- Fault case 3: The quality of communication was degraded due to a problem of AP in this case.

All data transmissions were conducted for 60 s. In each case, we measured the parameters explained in Section IV.B using *Wireshark* application [35].

VI. PERFORMANCE EVALUATION

In this section, we explain various experiments and analyze the results of experiments. Firstly, we analyze techniques applied to *MoNPI*. After that, we conduct an analysis on the iteration times for learning and configurations of learning model. Then, we evaluate the performance of *MoNPI* and show each parameter's influence on *MoNPI*'s performance. In addition, we show the effectiveness of identification method using consecutive results.

To verify performance, we utilized TensorFlow library [36] to implement the learning model of *MoNPI*. Also, we created a data set that prevents overfitting by randomly combining the data source from the Section V and used it for learning. We implemented *MoNPI* on Windows 10 using the desktop equipped with Intel(R) Xeon(R) CPU E5-2630 v3 and 32GB RAM. We trained the learning model by utilizing Nvidia's compute unified device architecture (CUDA) on NVIDIA GeForce GTX 980 Ti graphic card for faster learning.

A. Analysis on Learning Techniques Applied to MoNPI

In this subsection, we analyze the effect of techniques used for *MoNPI*. We conduct experiments to show the effectiveness of normalization and dropout, and then analyze the accuracy and processing time depending on the number of iterations.

A.1 Normalization

As explained in Section IV.C, the ranges and scales of measured values are very different from one another. Thus, the preprocessing is essential to make the learning model of *MoNPI* learn the characteristics of parameters properly, and we used the normalization method. In this analysis, we prepared two data sets, the original value set and normalized value set, based on the collected data in LTE network scenario explained in Section V.A.1. Using the sets, we conducted the learning and compared the results.

Fig. 4 shows the result of comparison. The accuracy means the number of right estimations over the number of total estimations. As shown in the figure, in the beginning of the learning, the accuracy values in both cases rise as the iteration increases. However, after the number of iteration becomes larger than 100, the accuracy in the case without normalization does not increase any more and fluctuates after that. This result proves that the normalization can improve the accuracy for analysis on networks.

A.2 Dropout

We used dropout to avoid the overfitting problem as explained in Section IV.D. If the probability of retaining a node, p, is 0.7, nodes are dropped with a probability of 0.3 every iteration. To analyze the effect of dropout, we conducted learning using the data set used in the previous experiment and measured the accuracy by varying p. According to [30], typical values of p are in the range 0.5 to 0.8, so we changed p from 0.5 to 1.

As shown in Fig. 5, the accuracy in every case rises as the iteration increases similar to Fig. 4. However, when p is 1, in other words, when the dropout is not used, the increase rate of accuracy is significantly decreased after the iteration becomes larger than about 500. Moreover, there are some fluctuations unlike other cases' graph lines, which means that the model was not trained properly without using dropout.

Using the test set, which was a tenth of the total data set and not used for training, we evaluated the estimation performance of the models trained using different p values. Fig. 6 shows the results, and it is easy to see that the result without using dropout is poor compared to the results of other cases. Through the results, we decided to use the dropout with p of 0.7 for *MoNPI* to avoid the overfitting problem and obtain better performance.

B. Analysis on Iteration Times

As shown in the results of the previous experiments, the accuracy increases as the number of iterations increases. However, after the learning operation is performed sufficiently, the increase rate of accuracy significantly decreases, and the performance improvement finally reaches the limit. To analyze the

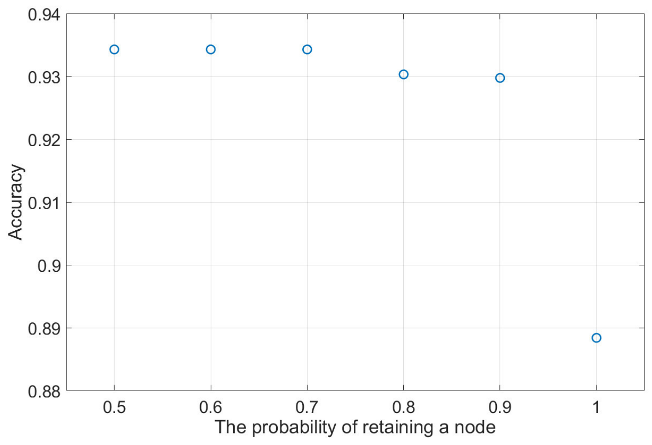

Fig. 6. The accuracy results depending on the probability of retaining a node.

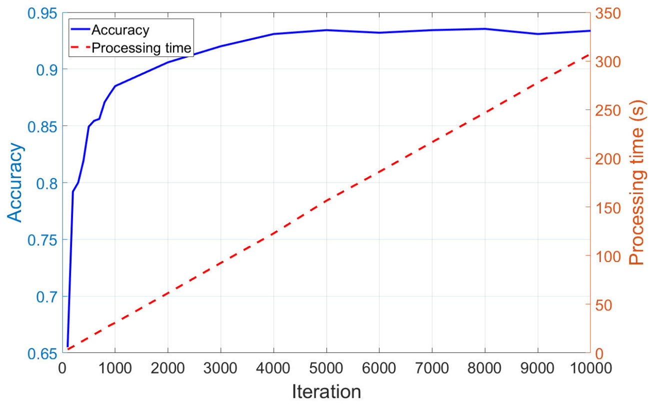

Fig. 7. The accuracy and processing time depending on the number of iterations.

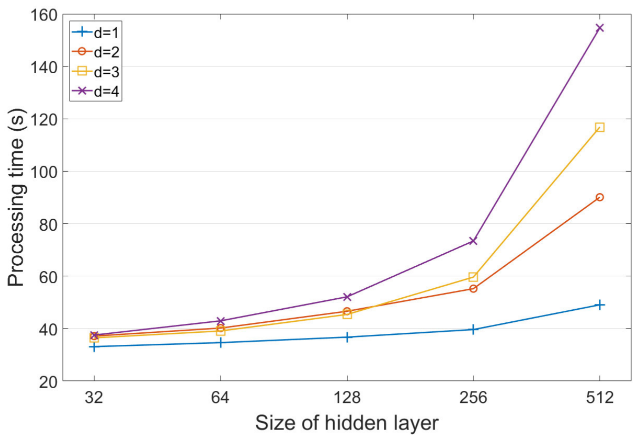

Fig. 8. The processing time depending on the depth and size of hidden layer.

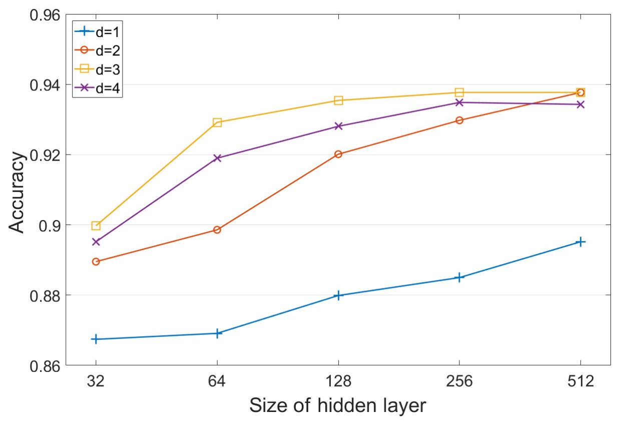

Fig. 9. The accuracy results depending on the depth and size of hidden layer.

relation between the performance improvement and iteration in learning process, we measured the learning time and accuracy according to the number of iteration. We measured them while conducting the learning similar to the previous experiments.

Fig. 7 shows the results of this analysis. In the figure, the accuracy rises according to the increase of iteration. However, after the learning is performed sufficiently, the accuracy does not increase anymore. On the contrary, the processing time for learning naturally keeps increasing as the number of iterations increases. Thus, for efficient learning and satisfactory performance, the appropriate number of iterations should be determined. In this analysis, after the number of iterations is more than about 5000, the accuracy scarcely increases. According to this result, we decided to set the number of iterations for learning process of *MoNPI* to 5000. In this case, as shown in Fig. 7, the training time of the neural network takes 150 s.

C. Analysis on Configurations of the Learning Model

To configure learning model properly is important to provide the acceptable performance. Thus, in this subsection, we analyze the learning model's performance by changing the configuration of model, and determine the configuration values suitable for *MoNPI*.

C.1 Performance Depending on the Number and Size of Hidden Layers

Firstly, we analyze the performance depending on the number and size of hidden layers. We measured the processing time by varying the number and size of hidden layers, and Fig. 8 shows the result. In the figure, the size of hidden layer means the number of nodes composing each layer, and d in the legend means the depth of neural network. In other words, d is the number of hidden layers in Fig. 3. As shown in Fig. 8, the processing time naturally rises as d or the size increases. Especially, the processing time rises significantly when the size is increased from 256 to 512.

In addition, we measured the accuracy by varying the size and d. As shown in Fig. 9, the accuracy rises as the size of hidden layer increases in all cases. However, when d is small (1 or 2), in other words, when the hidden layer is too shallow, it is hard for the model to learn characteristics of networks profoundly. Similar to the depth, the size also needs to be large enough to provide acceptable accuracy performance. In conclusion, considering the above results, we decided to set the size and depth to 256 and 3, respectively.

C.2 Performance Depending on Learning Rate

The learning rate affects the accuracy and time required to obtain acceptable result. Thus, we analyze the accuracy depending

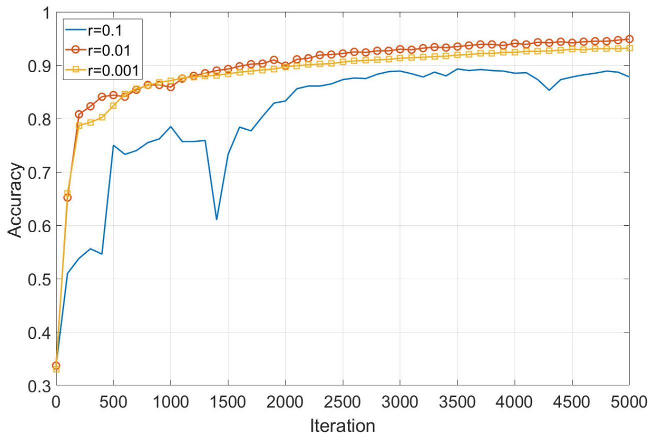

Fig. 10. The accuracy results depending on the learning rate and number of iterations.

Table 2. Accuracy results of *MoNPI*.

	Simulation		Empirical experiment	
	LTE	***Wi-Fi***	***Wi-Fi (download)***	***Wi-Fi (upload)***
Total	90.56%	90.26%	81.83%	99.36%
Normal case	92.57%	95.41%	68.91%	100.00%
Fault case 1	98.78%	83.03%	76.27%	100.00%
Fault case 2	81.65%	100%	100.00%	97.50%
Fault case 3	N/A	N/A	81.65%	99.13%

on the learning rate, and Fig. 10 shows the result. When the learning rate is too large, a local minima problem can occur, and it may be hard for the model to learn properly, so that the model cannot provide the acceptable performance. On the contrary, when the learning rate is too small, it takes a longer time to obtain the acceptable accuracy. The figure shows such characteristic of learning rate. Thus, we set the learning rate in *MoNPI* to 0.01 based on the result.

D. Evaluation on Problem Spot Identification of MoNPI

Using analysis results explained in the previous subsections, we configured *MoNPI*. Then, we evaluated *MoNPI*'s performance of problem spot identification using the data sets explained in Section V. Table 2 shows the each case's accuracy result as well as the overall accuracy result of each scenario. As shown in the table, *MoNPI* identified the problem spot in network accurately. This result shows that mobile devices using *MoNPI* are able to understand networks' situations autonomously. In these results, the method explained in IV.E was not applied yet, in other words, the number of results used to infer the problem spot, W_{output}, is 1.

In addition, we analyze each parameter's influence on *MoNPI*'s performance. A certain parameter's influence means the accuracy result when *MoNPI* conducts problem spot identification using only the parameter. As shown in Table 3 from the simulation experiment and Table 4 from the empirical experiment, various parameters were utilized together to distinguish problem spots. This result shows that different parameters should be considered to understand complex and diverse network situations.

Table 3. Analysis on influence of each parameter in the simulation experiment.

Parameter	***LTE***	***Wi-Fi***
Throughput (mean)	50.44%	56.34%
Throughput (max)	57.52%	24.18%
Throughput (median)	55.75%	56.34%
Throughput (std)	N/A	56.04%
CWND (mean)	44.54%	56.34%
CWND (max)	47.78%	56.34%
CWND (median)	48.96%	56.34%
CWND (std)	56.04%	56.34%
RWND (mean)	55.13%	74.45%
RWND (max)	54.3%	68.16%
RWND (median)	48.22%	73.65%
RTT (std)	33.92%	N/A
Inter-arrival time (mean)	42.77%	56.34%
Inter-arrival time (max)	41.00%	56.34%
Inter-arrival time (median)	32.44%	N/A
Inter-arrival time (std)	40.70%	56.34%
RX drop count	24.48%	50.14%
Frame receive delay of LTE	51.91%	N/A

Table 4. Analysis on influence of each parameter in the empirical experiment.

Parameter	***Wi-Fi (down)***	***Wi-Fi (up)***
Throughput (mean)	67.73%	16.98%
Throughput (max)	50.42%	16.98%
Throughput (median)	23.29%	16.98%
Throughput (std)	65.81%	16.98%
RWND (mean)	23.29%	91.93%
RWND (max)	23.29%	16.98%
RWND (median)	23.29%	76.43%
RWND (std)	51.28%	16.98%
RTT (mean)	51.49%	91.29%
RTT (max)	51.28%	93.20%
RTT (median)	25.42%	90.23%
RTT (std)	51.49%	16.98%
Inter-arrival time (mean)	23.29%	16.98%
Inter-arrival time (max)	59.61%	16.98%
Inter-arrival time (median)	23.29%	16.98%
Inter-arrival time (std)	56.19%	16.98%
RX drop count	23.29%	35.66%

E. Effectiveness of Identification Method using Consecutive Results

As explained in Section IV.E, *MoNPI* uses multiple results together to infer the problem spot more accurately. To verify the effectiveness of this method, we applied the method to the results shown in Section VI.D. Fig. 11 shows accuracy results depending on the number of results used to infer the problem spot, W_{output}. As shown in the figure, the accuracy values of all scenarios rise as W_{output} increases, and they are always 1 when W_{output} is larger than 60. In Wi-Fi download scenario, without using the method, in other words, when W_{output} is 1, *MoNPI* identifies the problem spot with a accuracy of 0.81. However, *MoNPI* using the method always identifies the problem spot ac-

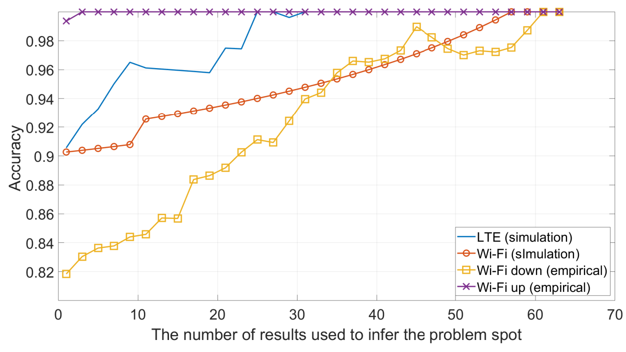

Fig. 11. The accuracy results depending on the number of results used to infer the problem spot.

curately by considering results for just 3 s[1]. This result shows that, by using the method, *MoNPI* is able to infer the problem spot accurately regardless of momentary changes in networks or communications.

VII. DISCUSSION

In this paper, we chose some major problem which can occur frequently in practical situations. If a device using *MoNPI* identifies what the problem is, the device is able to take a proper action. For instance, if the device knows that the AP used by the device is congested due to many other devices using the AP, the device can try to make a new connection with another usable APs around the device. In addition, to use cellular networks instead of WLANs can be a good choice for the device in this situation. On the other hand, if the problem is because of the poor state of server, the device should consider utilizing another service which can be an alternative to the service being used by the device. As explained above, by using *MoNPI*, mobile devices are able to conduct data communication more intelligently. The aforementioned examples are some of many countermeasures. Thus, to take other proper actions or utilize techniques handling network problems with *MoNPI* brings more performance improvement.

VIII. CONCLUSION

Nowadays, mobile devices have powerful computing power and are able to collect various network information, so they can be utilized to infer the network situation and context around the people. However, it is not trivial for mobile devices to understand the state of networks because today's networks are not only large but also complex and dynamic. Moreover, it is hard to devise models or algorithms capable of presenting the networks accurately or analytically. To overcome such limitation, we leverage deep learning to devise a mobile device-centric network problem spot identification, *MoNPI*. We designed and implemented *MoNPI*, and conducted various evaluations. Through the results of evaluations, we demonstrated that a mobile device with *MoNPI* is able to identify the spot having the most likely cause of poor network quality. In conclusion, by using *MoNPI*, mobile devices are able to understand networks' situations without cooperation of other devices or systems and thus make decisions more intelligently.

We have several directions for future work. This paper shows that *MoNPI* is able to identify some major network problems which can occur frequently in practical situations. Furthermore, we plan to improve *MoNPI* to recognize more complex and various network problems. We will analyze various parameters to improve *MoNPI* to more accurately understand situations in which communication quality changes frequently occur, such as downlink or mobility.

[1] δ was 50 ms in this experiment.

REFERENCES

[1] S. Park, H. T. Kim, and H. Kim. "Energy-efficient topology control for UAV networks," *Energies*, vol. 12, no. 23, pp. 1–19, Nov. 2019.

[2] M. C. Luizelli, L. R. Bays, L. S. Buriol, M. P. Barcellos, and L. P. Gaspary, "How physical network topologies affect virtual network embedding quality: A characterization study based on ISP and datacenter networks," *J. Network Comput. Applicat.*, vol. 70, pp. 1–16, July 2016.

[3] J. Jung, S. Yoo, W. G. La, D. R. Lee, and H. Kim, "AVSS: Airborne video surveillance system," *Sensors*, vol. 18 no. 6, June 2018.

[4] H. Kim and H. Kim, "Designing interference-aware network selection protocol for WLAN mobile devices," *Trans. Emerg. Telecommun. Technol.*, vol. 28, no. 1, Jan. 2017.

[5] P. Kulkarni and Y. Ozturk, "mPHASiS: Mobile patient healthcare and sensor information system," *J. Network Comput. Applicat.*, vol. 34, no. 1, pp. 402–417, Jan. 2011.

[6] M. Bae *et al.*, "Devising mobile sensing and actuation infrastructure with drones," *Sensors*, vol 18, no. 2, Feb. 2018.

[7] P. Poupart *et al.*, "Online flow size prediction for improved network routing," in *Proc. IEEE ICNP*, 2016.

[8] B. Arzani, S. Ciraci, B. T. Loo, A. Schuster, and G. Outhred, "Taking the blame game out of data centers operations with netpoirot," in *Proc. ACM SIGCOMM*, 2016.

[9] Z. Wang, "The applications of deep learning on traffic identification," *BlackHat USA*, 2015.

[10] V. L. Thing, "IEEE 802.11 network anomaly detection and attack classification: A deep learning approach," in *Proc. IEEE WCNC*, 2017.

[11] S. Yadav and S. Subramanian, "Detection of application layer ddos attack by feature learning using stacked autoencoder," in *Proc. IEEE ICCTICT*, 2016.

[12] I. Sohn, "Robustness enhancement of complex networks via No-Regret learning," *ICT Express*, vol 5, no. 3, pp. 163–166, Sept. 2019.

[13] A. Javaid, Q. Niyaz, W. Sun, and M. Alam, "A deep learning approach for network intrusion detection system," in *Proc. BIONETICS*, 2016.

[14] B. Dong and X. Wang, "Comparison deep learning method to traditional methods using for network intrusion detection," in *Proc. IEEE ICCSN*, 2016.

[15] R. Kou and Y. Wang, "Transmission line fault identification based on BP neural network," in *Proc. IEEE ISGT Asia*, 2019.

[16] DR. Lee, WG. La, and H. Kim. "Drone detection and identification system using artificial intelligence," in *Proc. IEEE/KICS ICTC*, 2018.

[17] K. K. Nguyen *et al.*, "Cyberattack detection in mobile cloud computing: A deep learning approach," in *Proc. IEEE WCNC*, 2018.

[18] Z. Yuan, Y. Lu, Z. Wang, and Y. Xue, "Droid-sec: Deep learning in android malware detection," *ACM SIGCOMM Comput. Commun. Review*, vol. 44, pp. 371–372, 2014.

[19] A. Este, F. Gringoli, and L. Salgarelli, "Support vector machines for TCP traffic classification," in *Comput. Networks*, vol. 53, no. 14, pp. 2476–2490, Sept. 2009.

[20] KMC. Tan, and B. S. Collie, "Detection and classification of TCP/IP network services," in *Proc. IEEE ACSAC*, 1997.

[21] S. Potluri and C. Diedrich, "Accelerated deep neural networks for enhanced intrusion detection system," in *Proc. IEEE ETFA*, 2016.

[22] T. A. Tang, L. Mhamdi, D. McLernon, S. A. R. Zaidi, and M. Ghogho, "Deep learning approach for network intrusion detection in software defined networking," in *Proc. IEEE WINCOM*, 2016.

[23] C. Yin, Y. Zhu, J. Fei, and X. He, "A deep learning approach for intrusion detection using recurrent neural networks," *IEEE Access*, vol. 5, pp. 21954–21961, Oct. 2017.

[24] I. Grigorik, "Latency: The new web performance bottleneck," [Online] Available: https://www.igvita.com/2012/07/19/latency-the-new-web-performance-bottleneck/

[25] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Proc. PMLR*, 2010.
[26] D. Mishkin and J. Matas, “All you need is a good init,” *arXiv preprint arXiv:1511.06422*, 2015.
[27] K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *Proc. IEEE ICCV*, 2015.
[28] D. P. Kingma and J. Ba, “Adam: A method for stochastic optimization,” *arXiv preprint arXiv:1412.6980*, 2014.
[29] G. E. Dahl, T. N. Sainath, and G. E. Hinton, “Improving deep neural networks for lvcsr using rectified linear units and dropout,” in *Proc. IEEE ICASSP*, 2013.
[30] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learning Research*, vol. 15, no. 1, pp. 1929–1958, June 2014.
[31] G. F. Riley and T. R. Henderson, “The ns-3 network simulator,” *Modeling Tools Network Simulation*, pp. 15–34, 2010.
[32] T. guide, “The fastest wireless networks of 2018,” [Online] Available: https://www.tomsguide.com/us/best-mobile-network,review-2942.html
[33] “iperf - the ultimate speed test tool for tcp, udp and sctp,” [Online] Available: https://iperf.fr/en/
[34] S. Acharya and N. Tiwari, “Survey of DDoS attacks based on TCP/IP protocol vulnerabilities,” *IOSR J. Comput. Eng.*, vol. 18, no. 3, pp. 68–76, 2016.
[35] U. Lamping, R. Sharpe, and E. Warnicke, “Wireshark user’s guide for wireshark 2.1,” 2014.
[36] M. Abadi *et al.*, “Tensorflow: A system for large-scale machine learning,” in *Proc. OSDI*, 2016.

Woonghee Lee received the B.S.E. degree in Electrical Engineering at Korea University, Seoul, Korea, in 2013, and the Ph.D. degree in Department of Electrical and Computer Engineering at Korea University in 2019. His research interests are in TCP/IP, IEEE 802.11, Bluetooth, network analysis/modeling, and machine/deep learning for improving networking and communication.

Joon Yeop Lee is currently a Ph.D. student in the School of Electrical Engineering at Korea University, Seoul, Korea. He received his B.S. degree in Electrical Engineering at Korea University, Seoul, Korea, in 2014. His research interests are in the area of community wireless networks, and network simulations.

Hwangnam Kim received the B.S.E. degree in Computer Engineering from Pusan National University, Busan, Korea, in 1992, the M.S.E. degree in Computer Engineering from Seoul National University, Seoul, Korea, in 1994, and the Ph.D. degree in Computer Science from the University of Illinois at Urbana-Champaign in 2004. He is currently a Professor with the School of Electrical Engineering, Korea University, Seoul, Korea. His research interests are in the areas of wireless networks, unmanned aerial system (UAS), UAS traffic management (UTM), counter UAS system, Internet of Things, and cyber physical systems.