Development of Advanced Edge Computing Framework using Rich Client Devices

Masato Oguchi
Department of Information Sciences
Ochanomizu University
Tokyo, Japan
oguchi@is.ocha.ac.jp

Saneyasu Yamaguchi
Faculty of Informatics
Kogakuin University
Tokyo, Japan
sane@cc.kogakuin.ac.jp

***Abstract*—An idea of edge computing becomes popular as the data can be processed in the vicinity of client IoT devices. Since the performance of client devices is not high, they only collect the data and send them to the edge server to process. Due to the data transfer cost and privacy matters, it is desirable to analyze the data within the client device as much as possible. Thus, we focus on a smartphone that is a typical and very popular rich client device. Performance of some types of smartphone is surprisingly high, and it is advanced greatly year by year. In this paper, we propose a framework to analyze data and make use of them inside the Android terminal. Since we already have investigated into the development of elaborative software executed inside Android kernel, and we also have investigated into the deep learning analysis using data obtained from Android terminal, the proposed framework can be realized by utilizing the result of such research works.**

***Index Terms*—edge computing, rich client, IoT device, Android kernel, deep learning**

I. Introduction

As client IoT devices such as sensors and cameras are connected to the cloud, the collected data are transferred to the cloud, and interesting information can be obtained by analyzing the data using machine learning techniques. Even if the devices are connected with a high-speed network, the data transfer cost is not negligible. In addition, passing raw data to the server might cause privacy problems.

Therefore, the idea of edge computing has become popular because the data can be processed in the vicinity of client devices, which is shown in Figure 1. Since the performance of client IoT devices such as wireless sensors is not high (which is called thin client devices), they only collect data and send those data to the edge server to analyze. Thus, the framework still presents data transfer costs and privacy problems. Management of the edge servers also matters because they are distributed, making management difficult compared to that of a centralized cloud server. For these reasons, a framework of rich client devices is desired, in which advanced data analysis can be executed before sending to edge/cloud servers.

We focus on smartphones, which are typical and very popular rich client devices. The performance of some types of smartphones is surprisingly high and is improved greatly year by year. Above all, these devices are currently carried around by users regularly. Thus, an advanced edge computing environment can be considered with smartphones as rich client devices, which is shown in Figure 2. Smartphones have a wide variety of sensors and collect data with them. If the data can be analyzed inside the smartphone and only the obtained information is sent to servers or utilized within the smartphone, the problem of data transfer cost and privacy matters can be solved.

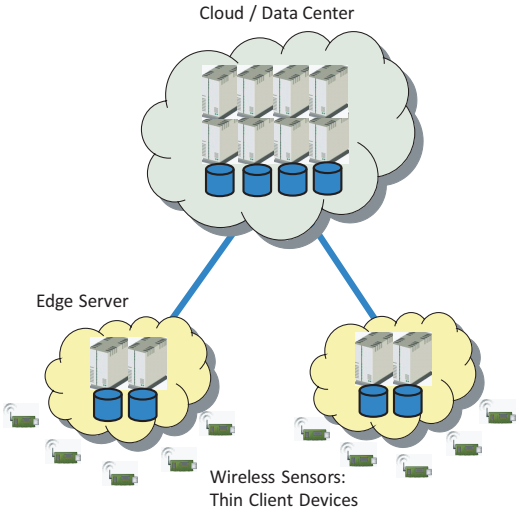

Fig. 1. Edge Computing Environment

In this regard, however, the smartphone is not an experimental terminal but a practical one, and therefore, their structure is more complicated than that of simple sensors. A strong security mechanism is also established. As a result, it requires a certain technique and know-how if we try to develop an advanced computing environment inside the smartphone, which is much harder than the development of ordinary user applications.

Fortunately, we have experience developing computing mechanisms inside the Android kernel. In addition, machine learning mechanisms that can be executed on a smartphone such as TensorFlow Lite have been released recently.

978-1-7281-5320-9/20/$31.00©2020 IEEE

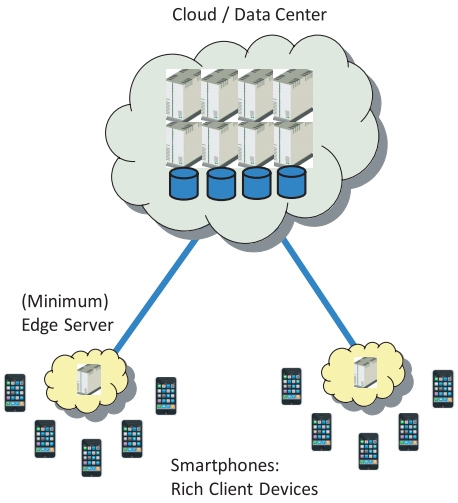

Fig. 2. Advanced Edge Computing Environment with Smartphones

Thus, we are developing a framework of advanced computing mechanism on an Android smartphone, in which data are captured and processed inside the device. We can retrieve data even from inside the kernel, process the data using a machine learning model, and control the system based on the information obtained by this process. We show the design of such a scenario.

The remainder of this paper is organized as follows. Section II explains the Android hardware and software environment. A recent machine learning framework for mobile devices is introduced in Section III. The research outcomes of Kernel Monitor and middleware are summarized in Section IV. Section V shows the parameter control mechanism based on machine learning. Some security issues are discussed in Section VI. Section VII introduces some related work, and Section VIII presents a conclusion and future direction of this research.

II. Android terminal

A. Android hardware environment

Previously, the smartphone was a poor-performing device compared with ordinary computers, even with portable laptops, not to mention servers. Currently, the performance of a low-end model smartphone is still poor. However, high-end models are different; they have amazingly high performance. The specifications of very recent models are almost comparable with those of ordinary laptop PCs.

Table I shows the performance of Google Pixel 3 released in October 2018 and Google Pixel 3a released in May 2019, respectively. A Qualcomm Snapdragon SoC is used with an Adreno GPU.

The performance of these devices is surprisingly high: The score of Snapdragon 845 on Google Pixel 3, measured by Geekbench 5, is 469 as a single-core device and 1815 as a multi-core device, while the baseline score using the Intel Core i3-8100 is 1000 [1].

TABLE I
Specification of Google Pixel 3 and 3a

	Pixel 3	**Pixel 3a**
CPU	Snapdragon 845 SDM845 (4 x 2.5 GHz + 4 x 1.6 GHz)	Snapdragon 670 SDM670 (4 x 2.0 GHz + 4 x 1.7 GHz)
GPU	Adreno 630	Adreno 615
Storage	64 GB	64 GB/128 GB
RAM	4 GB LPDDR4X	4 GB LPDDR4X
Battery	2915 mAh	3000 mAh
Monitor	5.5 inch (2160x1080 px)	5.6 inch (2220x1080 px)
Wi-Fi	IEEE 802.11 a/b/g/n/ac	IEEE 802.11 a/b/g/n/ac

B. Android software environment

Android OS is a platform for mobile terminals whose development is led by Google, and it is distributed as a package that includes an operating system, middleware and a set of applications. It is the largest installed base of any operating system, with over two billion monthly active users.

The source code of Android is available via the Android Open Source Project (AOSP) by the Open Handset Alliance. The standard code of Android is developed by AOSP [2]. This code is customized for particular hardware by adding unique code suitable for particular chipsets and devices. We have downloaded and compiled the Android kernel, which is developed based on Linux, as well as the Android userland. The built image data are copied to the Android terminal, which can be booted with the original image data.

III. Machine Learning on Smartphones

A. Deep Learning Framework

Deep learning is a neural network technique that is widely used for the analysis of images or videos. Deep learning refers to a machine learning scheme that relies on a neural network with many intermediate layers. A neural network is an information system that imitates the structure of the human cerebral cortex. Deep learning makes it possible to automatically extract features from data. Several deep learning frameworks have been developed so far, such as Caffe [3], TensorFlow [4], and Chainer [5].

B. TensorFlow Lite

TensorFlow Lite is a toolkit based on TensorFlow, a machine learning framework developed by Google, that is suitable for a mobile environment [6]. It converts a trained model from TensorFlow for a mobile environment, which requires decreasing the size of the model. The architecture of TensorFlow Lite is shown in Figure 3.

TensorFlow Lite consists of two components: the TensorFlow Lite converter and the TensorFlow Lite interpreter. An original TensorFlow model is converted into an efficient form that is suitable for lightweight devices such as a smartphone, and the converted model is executed on a TensorFlow Lite interpreter, which runs on the smartphone.

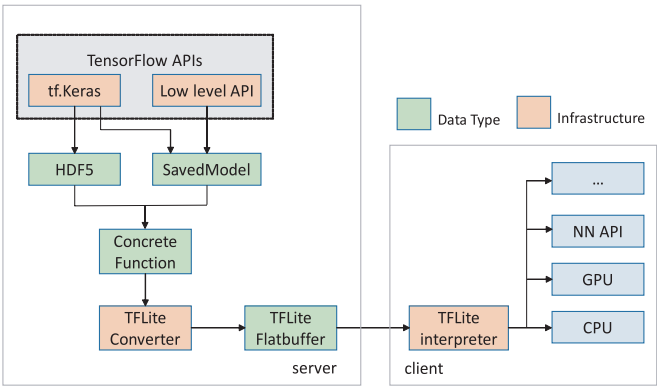

Fig. 3. Architecture of TensorFlow Lite [6]

IV. Kernel Monitor and Middleware

A. Kernel Monitor

We have developed a method to observe the behavior of parameters inside the Linux kernel code. The Kernel Monitor is a system tool that can observe the behavior of parameters inside the Linux kernel code, which is generally never observed from outside of the kernel. Our previous work successfully embedded it in the Android platform in order to analyze the connection status of a mobile host in real time [7].

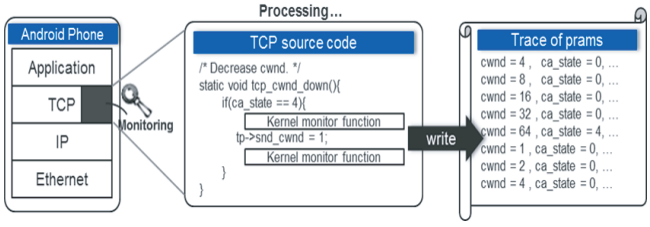

Fig. 4. Kernel Monitor observing TCP parameters

As shown in Figure 4, Kernel Monitor allows users to monitor parameters in the kernel processing at the mobile host, which include the congestion window (CWND), RTT, and timing of errors. The parameters are defined in the TCP implementation of the Linux kernel code, and applications in the user space can generally never observe or even recognize them. By means of Kernel Monitor, it is possible to access the current values of these parameters in the kernel memory space.

B. Kernel Parameter Controlling Middleware

In our previous work, kernel parameters controlling middleware were also developed [8] [9] [10]. The middleware controls CWND if the terminal originates TCP traffic and is connected to the server via a WLAN to address congestion among the AP and other Android terminals.

The middleware can be divided into two parts, as shown in Fig. 5. One part adjusts the congestion control, using the process interface to prevent segments from overflowing and

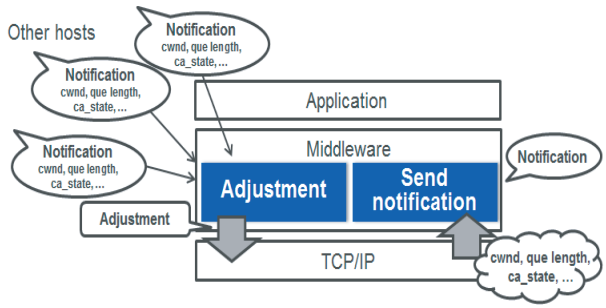

Fig. 5. Kernel Parameter Controlling Middleware

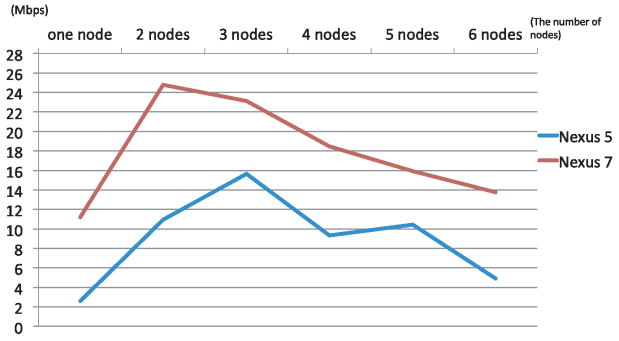

Fig. 6. Total Throughput as the Number of Terminals Changed

saturating the bandwidth. The notification is broadcast by UDP every 0.3 seconds from the other part because the kernel parameters frequently change. The adjustment is executed every 10 seconds because the number of mobile hosts changes less often, and this lower frequency is enough to collect information from all hosts, considering the notification interval and the arrival rate of notifications.

C. An Effect of System Control by Middleware

In this subsection, an effect of system control middleware is introduced. In this experiment, packets are transferred from 6 Android terminals connected to an access point to a server using iPerf [11]. An artificial delay of 512 milliseconds is inserted between the access point and the server to emulate the cloud server environment. As Android terminals, Nexus 5 and Nexus 7 smartphones were used.

The total throughput of the terminals is shown in Figure 6. As the number of terminals increases beyond 3 nodes, the total throughput decreases due to the traffic congestion at the access point.

By introducing the kernel parameter controlling middleware mentioned in the previous subsection, TCP parameters inside the kernel such as CWND can be controlled based on the observed traffic condition. The evaluation results are shown in Figures 7 and 8 when 6 terminals send packets to a server simultaneously. Not only does the total throughput increase but also the fairness of transmission among the terminals is increased by the middleware.

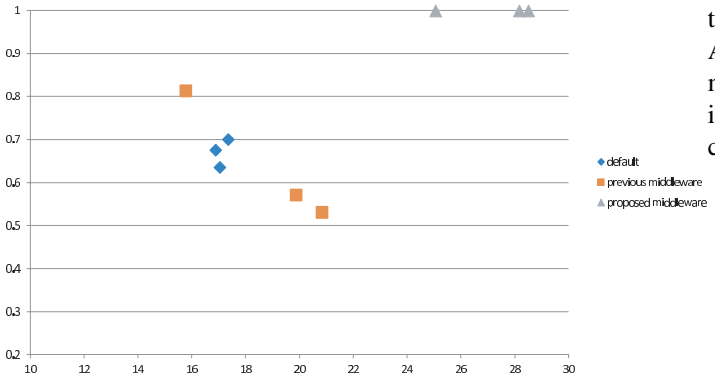

Fig. 7. Total Throughput and Fairness (Nexus 5)

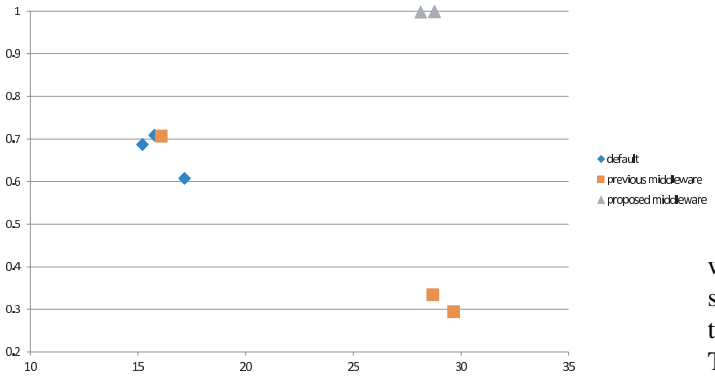

Fig. 8. Total Throughput and Fairness (Nexus 7)

D. Implementation into the Latest Smartphone

In our previous works, the Kernel Monitor and middleware are implemented in smartphones such as Google Nexus S [9] and Nexus 5 and 7 [10]. Compared with these devices, the latest smartphones have greater performance, as shown in Section II. The Android firmware versions are also different: ver. 4.1.1 for Nexus S, ver. 5.1.1 for Nexus 5, ver. 6.0.0 for Nexus 7, and currently, ver. 9.0.0 for Pixel 3/3a.

Such differences require a technique and know-how for the implementation of elaborate software. We have implemented the Kernel Monitor and the middleware into Pixel 3a successfully.

V. Parameter Control Mechanism based on Machine Learning

A. An Overview of the Proposed Framework

As shown in Section IV, we have established a mechanism that can acquire data from Android terminal, including parameters inside the kernel. The obtained data can be injected into TensorFlow Lite for analysis, and the behavior of the system can be predicted as a result. It should be used to control the system through the middleware introduced in Section IV.

Thus, we can establish a system that acquires parameters when the Android terminal is used, executes a machine learning process based on them, and sends back the result to control the system. For example, TCP parameters are obtained from Android kernel in real time and analyzed using a deep learning model on TensorFlow Lite, and the behavior of network traffic is predicted in order to control the packet transfer to avoid congestion. The proposed framework is described in Figure 9.

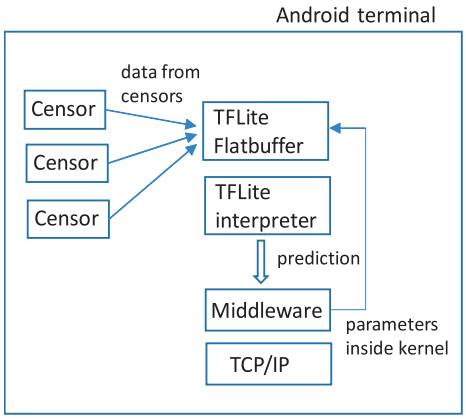

Fig. 9. An Overview of the Proposed Framework

To complete the implementation of the proposed framework, we need to convert a model of deep learning analysis to TensorFlow Lite Flatbuffer. For that purpose, we have investigated the deep learning analysis using data of the Android terminal. This is explained in the next subsection.

B. Deep Learning based on Data Collected on the Android Terminal

To realize the scenario described in the previous subsection, we require a model that predicts the behavior of network traffic using parameters of packets. We have already researched the deep learning model based on LSTM, a type of recurrent neural network (RNN) model, using parameters of captured packets of Wireless LAN to predict the congestion of the network [12].

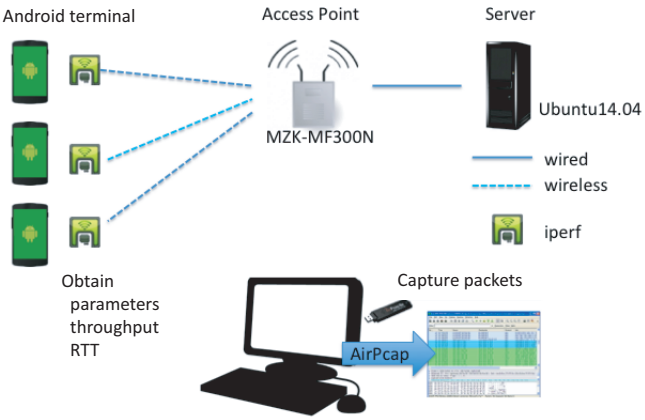

Fig. 10. Experiment Environment to Analyze Android Data

The experimental environment to analyze Android data is shown in Figure 10. Packets are transferred from multiple

Android terminals connected to an access point to a server using iPerf. On the Android terminals, parameters inside the kernel, throughput, and RTT values are obtained. In addition, AirPcap [13] (Riverbed Company) is used to capture the transferred packets. AirPcap is a USB-based adapter that captures 802.11 wireless traffic for analysis. These data are analyzed by LSTM to build a model of deep learning.

In this experiment, 6 Android terminals of Nexus S and Nexus 7 are used. First, 70 seconds of a dataset is created for the learning phase, in which packets are sent for only 30 seconds between 20 and 50 seconds by all Android terminals. Next, 50 seconds of a dataset is created for the validation phase, in which packets are sent for only 20 seconds between 10 and 30 seconds by all Android terminals. For the validation, the throughput of t+1 seconds is predicted using the dataset obtained until t seconds.

The learning dataset is analyzed by LSTM to build a model. Figure 11 shows a learning dataset and prediction result in which the learning data are put again into the learning model made by the data. According to this figure, the learning model is well trained for this type of network traffic.

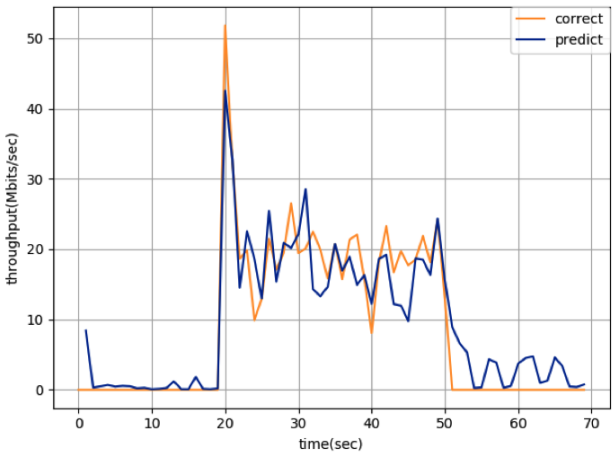

Fig. 11. Prediction Result using the Learning Dataset

Next, the validation dataset is analyzed to evaluate the accuracy of the prediction. The result is shown in Figure 12.

According to this figure, although accurate throughput values cannot be predicted, the timing at which the value increases and the timing at which it decreases can be predicted. Since the timing of sending packets and the time of the data sent are different from those of the learning data, it is considered that this type of network traffic can be generalized.

Currently, the prediction ability is limited to some types of network traffic since the learning dataset is not large enough. We are developing an experiment by utilizing a much larger volume of data to build a more generalized learning model. The model should be converted to TensorFlow Lite Flatbuffer, which is implemented in the Android terminal, and should perform deep learning analysis, as described in the proposed framework shown in the previous subsection.

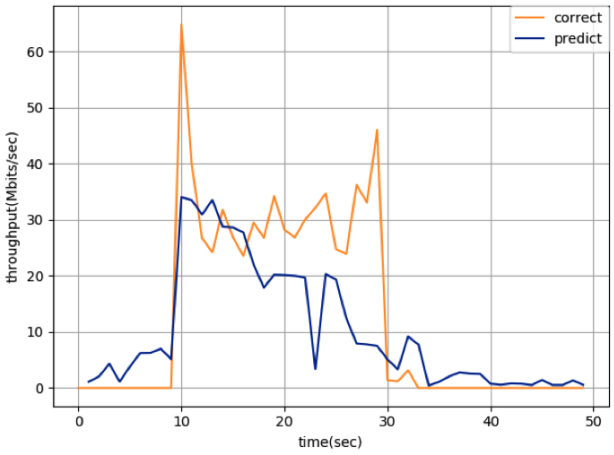

Fig. 12. Prediction Result using the Validation Dataset

VI. Security Issues

If the collected data is processed and used only within a smartphone, no security issue by eavesdropping is concerned. However, when such data and/or the retrieved information are transmitted to servers or other terminals, we must care about its security.

Among various encryption methods, we are focusing on Fully Homomorphic Encryption (FHE) which allows us to process data while they are encrypted [14] [15]. With the feature, it is possible to share data in safety between a terminal and a server or among terminals. The defect of FHE is that it requires huge volume of calculation for encryption and decryption. Therefore, only processing on a high-performance server machine is discussed so far.

However, as mentioned in Section II, the latest smartphone has greater performance. Thus, using FHE on smartphone may not be necessarily impossible anymore. In our experiment using HElib [16], one of FHE libraries, the key generation of FHE using Google Pixel 3 is about 1.7 times slower compared with the case using laptop PC that has 1.3GHz Intel Core CPU, perhaps because the laptop PC has much volume of memory than the Pixel 3. However, the encryption time is almost comparable in both cases. As the performance of smartphones increases, using FHE should become reasonable to share data in safety even for smartphones.

VII. Related Work

Edge computing has recently attracted attention from various viewpoints. Various distributed processing methods using edge computing are proposed to analyze data obtained from IoT devices. For example, a data analysis method for IoT devices using deep learning is described in [17]. A deep learning application for the IoT devices is built using edge computing since the processing capacity of an IoT device is limited, and an offload strategy for performance optimization is designed and evaluated. A hierarchical distributed fog computing architecture is proposed, which supports massive

infrastructure integration for future smart cities [18]. The models and architectures of typical fog computing systems are investigated in [19]. The design space for the four dimensions of system, data, human, and optimization are analyzed.

In those works, IoT devices are assumed to be thin clients with poor computing resources. Data obtained from client devices are transferred to edge/fog servers in the vicinity of the devices and analyzed by deep learning methods. In our research, client devices are assumed to be rich clients that have relatively powerful computing resources. It is possible to analyze the data inside the client devices, and only the result of deep learning might be transferred to edge servers and/or cloud servers.

VIII. Conclusions

In this paper, we have discussed the advancement of edge computing and focused on a smartphone, which is a typical and very popular rich client device. The performance of some types of smartphones is surprisingly high and is improved greatly year by year.

We have proposed a framework to analyze data and make use of it inside the Android terminal. Since we have already investigated the development of elaborate software executed inside the Android kernel and have also investigated the deep learning analysis using data obtained from the Android terminal, the proposed framework should be realized by utilizing the result of this work.

Acknowledgment

This work was partly supported by JST CREST Grant Number JPMJCR1503, Japan.

References

[1] Google Pixel 3 Benchmarks. Retrieved September 6, 2019 from https://browser.geekbench.com/android_devices/835

[2] Android open source project. Retrieved September 6, 2019 from http://source.android.com

[3] Y. Jia et al., “Caffe: Convolutional architecture for fast feature embedding,” arXiv preprint arXiv:1408.5093, 2014.

[4] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vigas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, “TensorFlow: Large-scale machine learning on heterogeneous systems,” 2015. http://download.tensorflow.org/ paper/whitepaper2015.pdf. pp. 1-19.

[5] S. Tokui, K. Oono, S. Hido, and J. Clayton, “Chainer: a nextgeneration open source framework for deep learning,” In Proceedings of Workshop on Machine Learning Systems (LearningSys) in The Twenty-ninth Annual Conference on Neural Information Processing Systems (NIPS), 2015. 6 pages.

[6] TensorFlow Lite, Retrieved September 6, 2019 from https://www.tensorflow.org/lite

[7] Kaori Miki, Saneyasu Yamaguchi, and Masato Oguchi, ”Kernel Monitor of Transport Layer Developed for Android Working on Mobile Phone Terminals,” In Proc. the Tenth International Conference on Networks (ICN2011), pp.297-302, January 2011.

[8] Hiromi Hirai, Saneyasu Yamaguchi, and Masato Oguchi, ”A Proposal on Cooperative Transmission Control Middleware on a Smartphone in a WLAN Environment,” In Proc. the 9th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob2013), pp. 710-717, October 2013.

[9] Ai Hayakawa, Saneyasu Yamaguchi, Masato Oguchi, ”Reducing the TCP ACK Packet Backlog at the WLAN Access Point,” In Proc. the 9th ACM International Conference on Ubiquitous Information Management and Communication (IMCOM2015), 5-4, January 2015.

[10] Ayumi Shimada, Saneyasu Yamaguchi, and Masato Oguchi, ”Performance Improvement of TCP Communication based on Cooperative Congestion Control in Android Terminals,” In Proc. the 12th ACM International Conference on Ubiquitous Information Management and Communication (IMCOM2018), 3-2, January 2018.

[11] Iperf For Android Project in Distributed Systems, Retrieved September 6, 2019 from http://www.cs.technion.ac.il/ sakogan/DSL/2011/projects/ iperf/index.html

[12] Aoi Yamamoto, Haruka Osanai, Akihiro Nakao, Shu Yamamoto, Saneyasu Yamaguchi, Takeshi Kamiyama, and Masato Oguchi, ”Prediction of Traffic Congestion on Wired And Wireless Networks Using RNN,” In Proc. the 13th International Conference on Ubiquitous Information Management and Communication (IMCOM2019), pp.3-2, January 2019.

[13] Riverbed AirPcap, Retrieved September 6, 2019 from https://support.riverbed.com/content/support/software/steelcentral-npm/airpcap.html

[14] Yuri Yamamoto and Masato Oguchi, ”A Decentralized System of Genome Secret Search Implemented with Fully Homomorphic Encryption,” In Proc. the 1st IEEE International Workshop on Big Data and IoT Security in Smart Computing (BITS2017) in conjunction with the 3rd IEEE International Conference on Smart Computing (SMARTCOMP2017), May 2017.

[15] Masato Oguchi, Kurt Rohloff and Yuki Yamada, ”Homomorphic Encryption for Privacy-Preserving Genome Sequences Search,” In Proc. the 3rd IEEE International Workshop on Big Data and IoT Security in Smart Computing (BITS2019) in conjunction with the 5th IEEE International Conference on Smart Computing (SMARTCOMP2019), June 2019.

[16] HElib, Retrieved September 6, 2019 from https://github.com/homenc/HElib

[17] He Li, Kaoru Ota, M. Dong, ”Learning IoT in Edge: Deep Learning for the Internet of Things with Edge Computing,” IEEE Network, Vol.32, No.1, pp.96-101, Janyary 2018.

[18] B. Tang, Z. Chen, G. Hefferman, T. Wei, H. He, Q. Yang, ”A hierarchical distributed fog computing architecture for big data analysis in smart cities,” October 2015. DOI:10.1145/2818869.2818898, .

[19] S. Yang, ”IoT Stream Processing and Analytics in the Fog,” IEEE Communications Magazine, Vol.55, No.8, pp.21-27, August 2017. DOI:10.1109/MCOM.2017.1600840