Highly Secure Mobile Devices Assisted with Trusted Cloud Computing Environments

Doohwan Oh, Ilkyu Kim, Keunsoo Kim, Sang-Min Lee, and Won Woo Ro

Mobile devices have been widespread and become very popular with connectivity to the Internet, and a lot of desktop PC applications are now aggressively ported to them. Unfortunately, mobile devices are often vulnerable to malicious attacks due to their common usage and connectivity to the Internet. Therefore, the demands on the development of mobile security systems increase in accordance with advances in mobile computing. However, it is very hard to run a security program on a mobile device all of the time due the device's limited computational power and battery life. To overcome these problems, we propose a novel mobile security scheme that migrates heavy computations on mobile devices to cloud servers. An efficient data transmission scheme for reducing data traffic between devices and servers over networks is introduced. We have evaluated the proposed scheme with a mobile device in a cloud environment, whereby it achieved a maximum speedup of 13.4 compared to a traditional algorithm.

Keywords: Mobile security, offloading computing, cloud computing, pattern matching, Wu-Manber.

Manuscript received Apr. 5, 2014; revised Nov. 7, 2014; accepted Nov. 28, 2014.

This work was supported by the ICT R&D program of MSIP/IITP, Rep. of Korea (14-000-05-001, Smart Networking Core Technology Development).

Doohwan Oh (ohdooh@yonsei.ac.kr), Keunsoo Kim (keunsoo.kim@yonsei.ac.kr), and Won Woo Ro (corresponding author, wro@yonsei.ac.kr) are with the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea.

Ilkyu Kim (ilkyu.kim@lge.com) is with the DTV SoC Development Department, LG Electronics, Seoul, Rep. of Korea.

Sang-Min Lee (sangm@etri.re.kr) is with the Broadcasting & Telecommunications Media Research Laboratory, ETRI, Daejeon, Rep. of Korea.

I. Introduction

With the emergence of mobile computing, the popularity of mobile devices, such as smartphones and tablet PCs, has exploded. With powerful hardware and software support, mobile devices have enabled many new services, such as web browsing, gaming, and email (which was only available on desktop computers in the past). In fact, some personalized services that cannot be provided on a desktop computer can now be realized on mobile devices. Such services include tracking a user's route with a GPS sensor, taking a photo (including geo-location information), and recognizing a user's gestures through the use of a gyroscope. These activities produce large amounts of personal data on mobile devices.

As a side effect of the emergence of mobile technology, mobile devices have become the target of hacking attacks [1], because the devices hold a huge amount of private data. In fact, mobile malwares; for example, Cabir, Trojan, and so on, which are developed to illegally collect such private data, are increasing in number significantly [2]–[3]. The widespread use of powerful mobile devices leads the spread of a large number of mobile malicious codes. Therefore, security systems for mobile devices are increasingly required according to the advances being made in mobile computing.

In spite of the importance of secure mobile computing, it is difficult to adapt traditional security applications to mobile devices due to hardware restrictions such as memory size, computing power, and battery life. Most security applications, including intrusion detection systems and anti-virus software, detect illegal activities through using string matching algorithms [4]–[5]. The algorithms require high computation power and large memory space; these requirements increase in proportion to the number of malicious signatures [6]–[8]. In

fact, computing power of mobile devices is not sufficient for detecting a large number of malicious signatures, despite the continual growth in hardware resources for mobile devices and advances in mobile application processors [9]. This then means that development of more efficient security schemes for mobile devices is essential.

Interest in cloud computing has increased with the popularity of mobile devices [9]–[11]. By migrating computation-intensive parts of heavy mobile applications to resourceful remote cloud servers that have higher performance and energy efficiency, mobile devices are able to offload applications to achieve low power consumption [12]–[17].

In this paper, we propose a design and implementation of a highly secure mobile system incorporating offloading computing. To be specific, we focus on the Wu-Manber (WM) algorithm [18], which is one of the most famous multiple pattern matching algorithms used in malware detection and one that is essential to implement secure mobile devices. Since the algorithm requires heavy computation, to save energy and reduce the workload of mobile devices, the heavy tasks of the algorithm are migrated to a resourceful cloud server. To achieve this, we analyze the optimal workload balancing and partition tasks into either lightweight or heavyweight tasks; the lightweight tasks are assigned to the mobile device, and the heavyweight tasks are assigned to the server. In addition, we propose a data transfer scheme that reduces both the power consumption and the delay times of data transfers. In summary, the key contributions of this paper are as follows:

- Analyzing and partitioning the workload of the security algorithm into lightweight and heavyweight tasks.
- Developing an offloading framework to provide an energy-efficient security scheme for mobile devices.
- Reducing power consumption on data transfer operations by diminishing the amount of data transmitted from a mobile device to cloud servers.

To evaluate the proposed system, we perform comprehensive experiments on a testbed consisting of an Android mobile device connected to the Amazon EC2 cloud server [19]. The proposed mobile security system saves execution time and power consumption on detecting any malicious patterns in the mobile device employing it. Compared with the traditional security algorithm running on the device without offloading computing, the best performance of the proposed approach provides a speedup factor of 13.4. In addition, the proposed approach saves up to 833 mWh of energy compared to the device's traditional security algorithm.

The remainder of this paper is organized as follows. In Section II, we present background information on malware detection algorithms and cloud computing, as well as previous studies with similar approaches. Section III describes the proposed mobile security system with cloud computing. The performance of the approach is evaluated and discussed in Section IV. Conclusions are presented in Section V.

II. Background

This section discusses related work and presents background information on previous malware detection algorithms.

1. Related Work

The emergence of mobile computing and virtualization technology has led to the commercial success of cloud computing. Major IT service providers have already launched cloud services and allow computation- or data-intensive tasks to be remotedly executed [19]. Mobile devices have adopted an offloading approach to migrate these tasks to cloud servers to improve performance and reduce energy consumption.

To implement a cloud-assisted system, the offload costs involved in the use of such a system should be considered for its efficiency. Since entire or partial amounts of data can be transmitted to cloud servers, the cost of data transmission is often considered to be the key factor in determining the suitability of an offloading computation. If the transmission cost is too high, then executing the workload on the mobile device would be the better option. Many studies have been conducted to find the optimal conditions under which offloading computing can take place in terms of the data transmission time and energy consumed [9], [20]–[21].

Previous researches have proposed many offloading approaches that migrate either an entire task or a part of a task over to other computing resources. In most cases, the detection systems utilize a centralized server that provides powerful computing resources. Oberheide and others [22] introduced a virtualized cloud system that migrates mobile antivirus functionality to a cloud resource. The cloud detects malicious intrusions by checking the files or binaries received from the mobile devices. In the approaches of Portokalidis and others [23] and Schmidt and others [24], the operating systems (OSs) of the target mobile devices are cloned in the cloud server, and the cloud detects malicious intrusions using the OS images it receives from the mobile devices. Although cloning an entire OS or sending files to be checked are simple and effective approaches, their proposed systems generally require a high degree of network activity [25]. The required high degree of network activity can be reduced by filtering data properly [25]–[27]. For instance, a mobile device can exclude files that are evidently irrelevant in detecting malwares. Cha and others [26] used bloom filters to filter binaries. Cheng and others [27] found that filtering algorithms are lightweight processes and

that the burden of the filtering is not high.

Chen and others [28] proposed multiple detection servers that can implement collaborative security platforms. The system uses routers to detect intrusions on mobile devices. Similar to other approaches, the detection can be achieved using a single router. If the workload in a router becomes heavy, then a collaborator dynamically manages other routers to help with the detection. Zonous and others [29] proposed Secloud, which is a cloud-based security system for smartphone devices. Secloud emulates a smartphone in the registered cloud server and performs data synchronization between the client and the server. Then, the cloud server inspects whether the data contains any malicious patterns, and the result is reported to the smartphone. Yang and others [30] proposed an intrusion detection system that detects intrusions on the client mobile device by collaborating with other peer devices. To minimize the required memory size to keep malicious signatures on mobile devices, only the most sensitive signatures are stored on each device.

2. WM Algorithm

The WM algorithm is a multiple pattern matching algorithm designed to find any predefined patterns in a given input text. The algorithm searches multiple patterns simultaneously by moving a search window along the incoming data stream [18]. Therefore, the algorithm is suitable in rapidly checking whether a search window contains any predefined known malicious signature strings in a data stream.

A. Preprocessing with Pattern Set

WM uses three tables; that is, the SHIFT, HASH, and PREFIX tables. These tables are constructed based on a predefined pattern set before starting the matching procedure. The SHIFT table originates from the bad-character shift scheme of the Boyer–Moore algorithm [31]. The indices of the table are made up of all possible *B*-character sequences appearing in the first LSP characters of all patterns, where LSP is the length of the shortest pattern; *B* is set as two or three in [18]. Each index has its own shift value, which defines how many characters the current search window moves. The shift values are calculated as LSP-*q*, where *q* is the location of the last character of a block in a pattern. If a block does not appear in any of the LSP-character sequences, then it's shift value is LSP–*B*+1. Figure 1 shows a SHIFT table with a set of patterns. In this example, the shift value of the "D1" index is set as four, because LSP is six with a *q*-value of two. The shift value for all blocks not appearing in any patterns is set as five, since the shift value is computed as LSP–*B*+1; these blocks are represented as "*" in the SHIFT table.

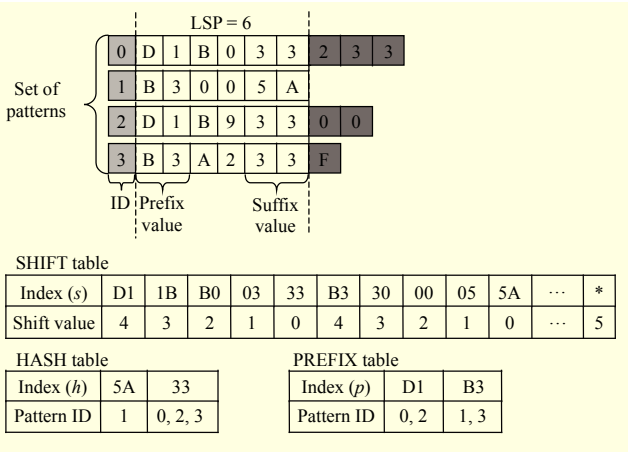

Fig. 1. Example of a set of patterns and SHIFT, HASH, and PREFIX tables of WM ($B = B' = 2$).

The HASH table groups the patterns based on their suffix value. The suffix value of each pattern indicates the *B*-character sequence located on the (LSP-*B*)th character of the pattern. For example, the "B3005A" pattern has "5A" as its suffix value, as shown in Fig. 1. Each index of the HASH table is made up of one of the suffix values and is associated with a list of patterns having that index as their suffix value.

In the PREFIX table, the patterns are grouped based on the prefix values of the patterns. The prefix value is the first *B*′-character sequence of each pattern. The patterns having an identical prefix value are grouped, and the prefix value then becomes an index of the PREFIX table. As shown in Fig. 1, the PREFIX table has two indices, and each of them has IDs of the patterns having that index as their prefix value.

B. Exact Pattern Matching

The WM algorithm starts the matching procedure with the tables constructed in the preprocessing step. In fact, the search window (*W*) is located on the first character of an incoming data stream (*text*), *T*; WM sets the size of the search window as equal to LSP. First, the suffix value of *W*, which is the last *B*-character piece of *W*, is found in the indices on the SHIFT table. Then, if the matched index has a shift value larger than zero, then the search window immediately moves to the right by as many characters as the shift value. Otherwise, the prefix value of *W*, which is the first *B*'-character piece of *W*, is checked with the PREFIX table. If there are no matched indexes, then the search window moves one character to the right and checks the suffix value of *W*, again. Otherwise, the character matching operation starts. All of the patterns listed in the index, which is equal to the suffix value of *W* in the HASH table, are matched with *W* in a brute-force manner. Figure 2 shows a working example of matching a pattern set (*P*) on an incoming data stream (*T*); the pattern set and the three tables are shown in Fig. 1.

ETRI Journal, Volume 37, Number 2, April 2015
http://dx.doi.org/10.4218/etrij.15.0114.0397

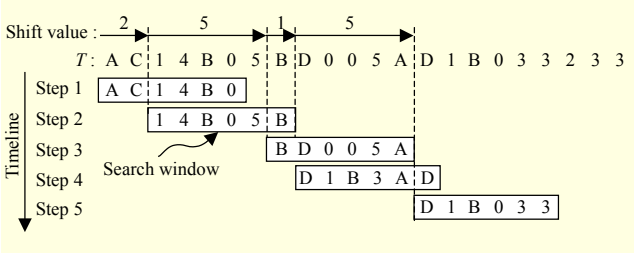

Fig. 2. Working example of WM.

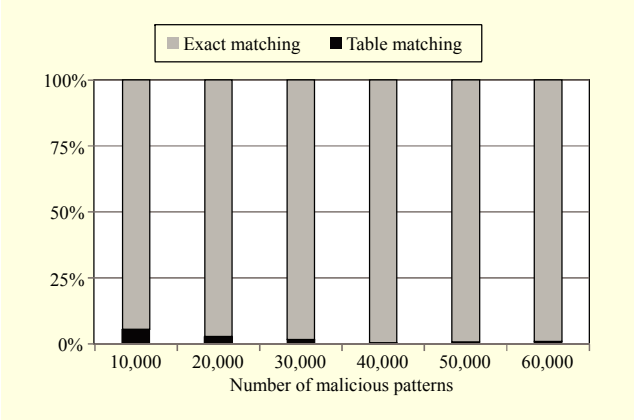

Fig. 3. Workload distribution on WM working on a mobile device.

C. Workload Analysis

To analyze the computation workload of the WM algorithm, we have evaluated the execution time of WM with various numbers of patterns. Figure 3 depicts the workload distribution of the WM process. The execution time for the exact matching operation accounts for approximately 94% of the overall time in the case of matching 10,000 patterns. This operation requires more time as the number of patterns increases. In fact, the execution time for the exact matching operation accounts for about 99% of the overall time in the case of matching 60,000 patterns. This is due to the fact that the table look-up operations consist of few data loads and comparison instructions, while the exact matching operation loads all of the patterns that have an equivalent prefix and suffix to the current input string and executes a one-by-one character comparison with all of them.

Figure 4 illustrates the process of the WM algorithm, including data and control flows. The size of a vertex represents the approximate amount of workload, and the width of a solid line indicates the amount of data transferred from one vertex to another; they are illustrated based on the experimental results shown in Fig. 3. The WM process consists of the following four key functions: the moving window function T, the shift table function S, the prefix table function P, and the exact matching function E. There is an input data stream, T_D, and output, R. Note that any of these functions can be offloaded

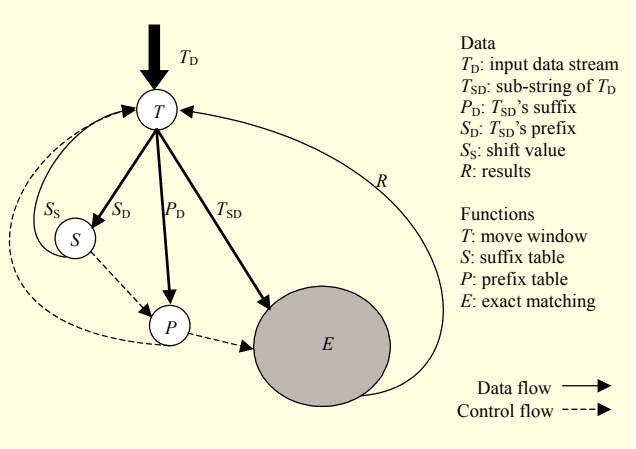

Fig. 4. WM process graph.

to cloud servers. The moving window function T reads T_D and moves a search window over T_D with a shift value, S_S. The shift table function, S, then, takes an input as a suffix on a current search window, S_D, and matches it against all indices in the shift table. If the shift value indicates zero, then P should be processed with a prefix value, P_D, which provides a prefix value on the current search window. In P, if an index on the prefix table matches with P_D, then E operates the exact matching operation with the substring on the current search window, T_{SD}, and reports the result R to T.

III. Cloud Framework

In this section, we present the CloudWM framework that performs distributed WM pattern matching by a collaborative work of a client mobile device and cloud servers. We discuss the workload partitioning policy that decides what section of the pattern matching procedure to offload. We also introduce our newly designed data-traffic reduction technique implemented in CloudWM.

1. Framework Overview

An overview of CloudWM is shown in Fig. 5. There are four major modules — Mobile Agent, Light Detector, Cloud Agent, and Heavy Detector. *Mobile Agent* manages the overall detecting processes on the mobile device; that is, reading binary files or incoming packets, managing the Light Detector module, and communicating with the cloud resources. *Light Detector* executes lightweight functions of WM. If there is the possibility of matching, then Mobile Agent is informed of this fact by Light Detector, and then sends the incoming data stream to the cloud.

On the cloud side, *Cloud Agent* manages the overall detecting execution; that is, communicating with the mobile

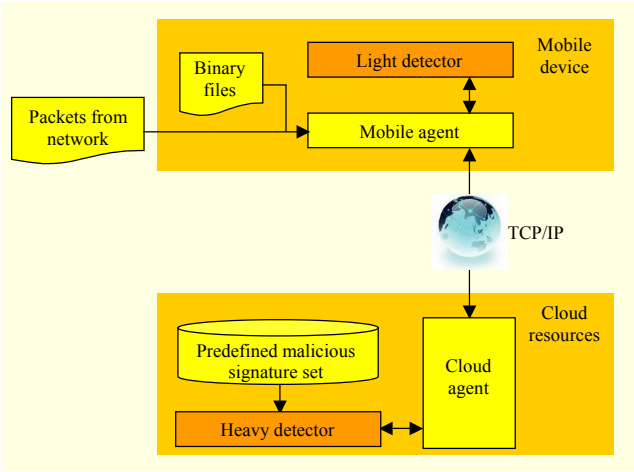

Fig. 5. Proposed mobile security system.

device and managing the Heavy Detector. When Cloud Agent receives a data stream inspection request from Mobile Agent, it initiates the detailed inspection process with *Heavy Detector* (executes heavyweight functions of WM).

2. Partitioning Workload of WM

We partition the workload of the WM algorithm statically. We first classify the functions of WM into lightweight and heavyweight ones. From the graph shown in Fig. 4, E will relatively be a heavy part of the WM process; S and P are lightweight functions. Therefore, Light Detector executes the lightweight functions, which are the SHIFT and PREFIX table look-up operations, on the mobile side, and Heavy Detector on the cloud side performs the heavyweight functions.

By offloading the heavyweight functions, both performance gain and power saving on mobile devices are achieved. However, the input line T_{SD} is wider than those of the others shown in Fig. 4. If the data flow frequently goes to E, then significant amounts of data would need to be transferred to a cloud server over the network. Therefore, an optimization scheme to reduce the amount of data transferred is required.

3. Reducing Amount of Data Transfer

In the proposed system, the data in the search window is transmitted to the cloud server for more detailed inspection when Light Detector identifies that this window may contain a potential malicious signature. This suspicious data comprises as many characters as the length of the longest pattern (LLP) from the position in the current window. Then, the suspicious data is inserted into a suspicious data queue (SDQ). In Fig. 6, an SDQ named Q9 contains a suspicious data stream. The queue is temporarily stored in the message buffer as an entry and sent to the cloud server.

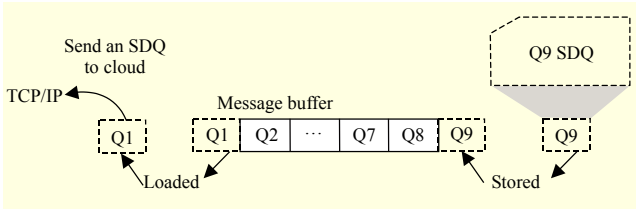

Fig. 6. Messages sent for suspicious malicious data stream.

If Light Detector frequently detects suspicious patterns or LLP is very long, then the amount of data transmitted to the cloud servers exponentially increases. In the worst case, the amount can be greater than that of the overall input data stream. In mobile environments, data transfer may consume a large amount of energy and lead to long computation times due to long network delay. In fact, the overhead on computation and power consumption for offloading computing increases as the amount of transmitting data increases.

To reduce the amount of data, an interleaved suspicious-data forwarding scheme is proposed. When a suspicious-data substring is inserted to SDQ, the window is moved to the right, immediately. If another suspicious-data substring is detected before the window moves right as many as LLP from the position detecting the last suspicious data, then there are redundant data between the last one and the current suspicious data. An interleaved forwarding scheme can prevent transmission of such redundant data. When a suspicious data stream is detected and another suspicious data stream is then detected in an adjacent position to the first, both are written on the same SDQ.

Figure 7 shows an example of the interleaved forwarding scheme. Let M_x be the position at which a suspicious data substring is detected. Let us assume that LLP is equal to 5 and that the size of SDQ is 16 bytes. The input data stream has five suspicious data streams until t_{26}. The first suspicious data stream appears at M_1. The suspicious data stream $\{t_1, t_2, t_3, t_4, t_5\}$ is inserted into an SDQ. The next suspicious data are located at M_2. The system attempts to insert data steam $\{t_3, t_4, t_5, t_6, t_7\}$ at the same SDQ. At this time, the partial data $\{t_3, t_4, t_5\}$ are already stored in the queue. Therefore, only data t_6 and t_7 are inserted. In the case of M_3, only a single datum, t_8, is stored to SDQ. Before detecting M_5, SDQ has only three empty spaces since the size of SDQ is limited to 16 bytes. This means that the data stream corresponding to M_5 cannot be fully stored in the same queue. Therefore, it will be contained in the next SDQ, and the current SDQ is stored in a sending message buffer as an entry in the buffer.

By combining the proposed interleaving scheme and lossless data compression, the total amount of transmitted data can be further reduced. The QuickLZ lossless compression is a well-known fast compression method. Before inserting SDQ into

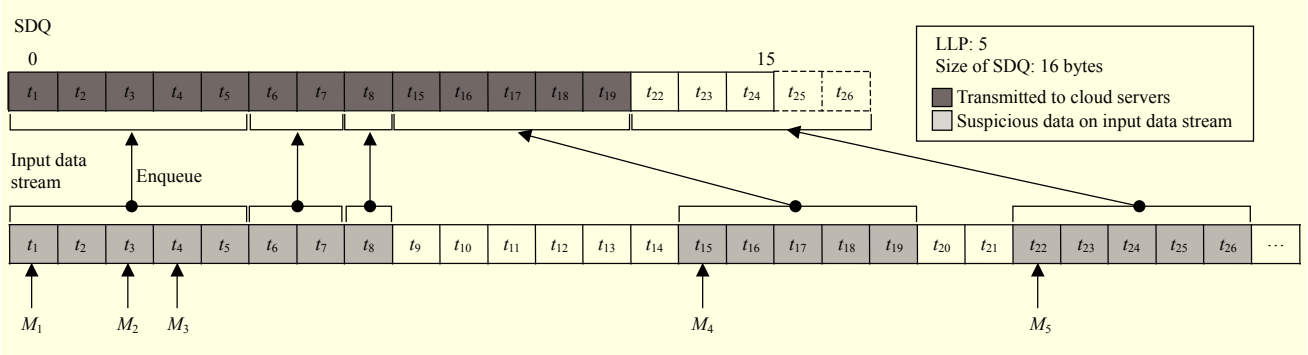

Fig. 7. Interleaved suspicious-data forwarding scheme.

the message buffer, we compress SDQ with QuickLZ. After the data are compressed by QuickLZ, the size of the data to be transmitted will be reduced to about 45% of its original size. In conclusion, the workload on transferring suspicious data is reduced to less than half compared to a naive method.

4. Detecting Modules on Mobile Side

On mobile devices, there are two main modules — Mobile Agent and Light Detector. Mobile Agent, which controls the overall detecting process on the mobile device, includes the following four modules: Manager, Window Mover, Sender, and Buffer, as shown in Fig. 8. Incoming data, such as binary files or packets from networks, is temporarily stored in Buffer. When the size of accumulated data in Buffer exceeds the length of the longest signature, Manager sends a start signal to Window Mover, which is the task corresponding to the *T* function of WM shown in Fig. 4. Window Mover moves the search window along the data in Buffer and sends the suffix value of the search window to Light Detector.

When Light Detector receives the suffix value of the input data stored in Buffer, the SHIFT table checks the shift value. If this shift value is larger than zero, then Window Mover moves the search window as many characters as this shift value. Otherwise, the PREFIX table matches the prefix value and sends the matching result to Window Mover. When there is a matched prefix, Window Mover informs this result to Manager. Otherwise, the search window moves one character and sends its suffix value to Light Detector again.

Manager signals to Sender when it decides to send suspicious data on the mobile device to the cloud server. Sender reads the suspicious data from Buffer; the location information of the data is transferred to Manager. At this time, Manager allocates an ID of the suspicious data for identifying the result from the cloud server. Next, the suspicious data are stored in SDQ and then sent to the cloud server after the interleaving and compression process described in the previous section.

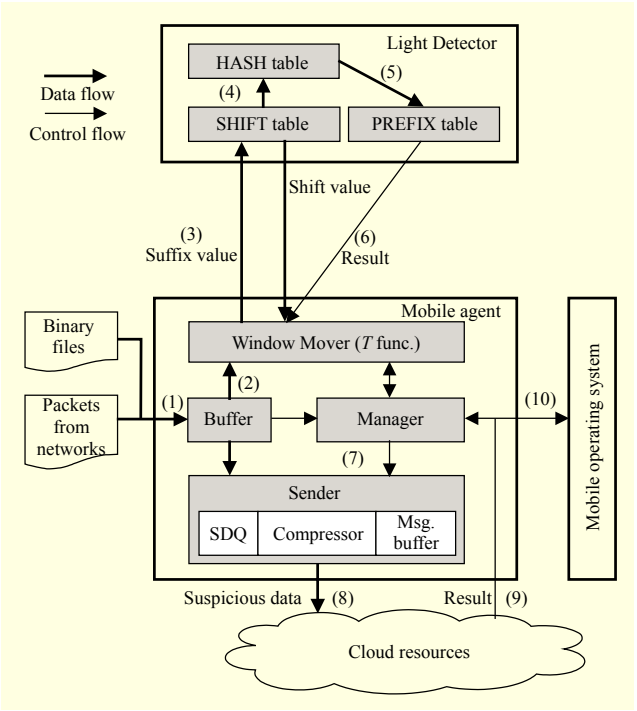

Fig. 8. Block diagram on mobile side of CloudWM.

5. Detecting Modules on Cloud Side

On the cloud side, the data received from the mobile device must be decompressed before executing the detailed data inspection through the exact character matching. Therefore, *Receiver* of Cloud Agent preprocesses the data to inspect them in Heavy Detector, as shown in Fig. 9. First, the data are decompressed to get the received SDQ and stored in DeSDQ. Then, SDQs are dequeued from DeSDQ in their arrival order, and then sent to Heavy Detector. This process is controlled by the Manager module.

When an SDQ arrives at Heavy Detector, the SHIFT table inspects it again, as on the mobile device. If the shift value is zero, then HASH table maps the suffix value to the list of the patterns having the suffix value. *Character Matcher* reads the

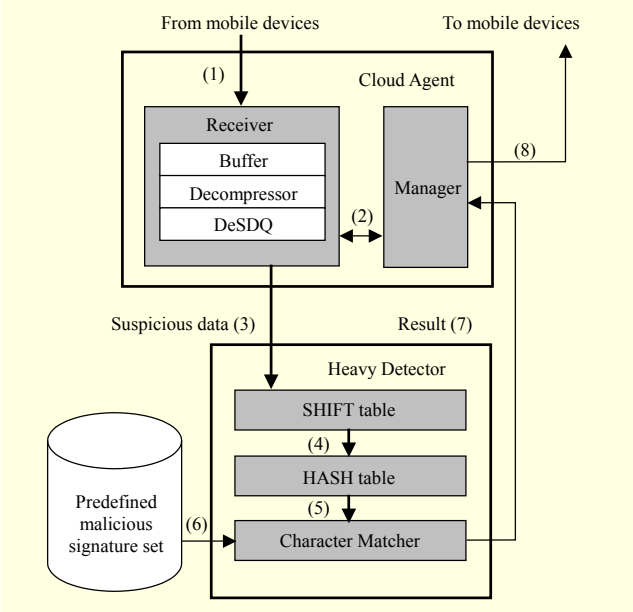

Fig. 9. Block diagram on cloud resources of CloudWM.

patterns in the list and runs exact character-by-character matching on the SDQ. When any matched pattern is found, the matched point and the SDQ's ID are returned to Manager of Cloud Agent. Finally, Manager reports the results to the mobile device.

Note that in Heavy Detector, the SHIFT and HASH tables are processed again, although these operations have already been executed in Light Detector on the mobile device. There can be several matched positions in an SDQ; therefore, information about these positions must be included in the SDQ. However, if the SHIFT table is performed on Heavy Detector, then the position information is not required on the cloud side. Indeed, the workload on SHIFT table is much less than the character matching operations, as mentioned in Section III-3. In addition, the HASH table is used by Character Matcher to ascertain those signatures that are candidates for matching on the SDQ in the signature set.

IV. Evaluation

In this section, the experimental methodology and results are presented. Then, the results are analyzed and discussed in detail.

1. Experimental Environments

For evaluation, we perform the experiments with a low-performance mobile device and high-performance servers. The application processor integrated in the mobile device has a 1 GHz CPU clock and 512 MB SDRAM. We use two types of servers — a local server and a cloud server. The local server has two physical processors that have 24 cores running with 2.3 GHz clock frequency. In addition, the system is equipped with a total of 128 GB DDR3 RAM. For the cloud server, we use Amazon EC2 instance [19]. This instance has two virtual cores configured with 2.5 EC2 Compute units and 1.7 GB memory.

The malicious patterns are extracted from the simple patterns of the ClamAV 0.97.2 signature set [32]. An input stream of around 100 MBytes is generated from some Android APK expansion files.

2. Performance Evaluation

To compare the scanning speed of CloudWM with the traditional WM algorithm, we have measured the inspecting time on matching various numbers of the malicious signatures on the target input stream. WM is operated only on the mobile device. CloudWM has three options — CWM, CWM_SDQ, and CWM_CompSDQ. CWM sends data as soon as a string subset is revealed as a candidate. CWM_SDQ additionally uses the proposed interleaved suspicious-data forwarding scheme. CWM_CompSDQ exploits the compression to CWM_SDQ.

Figure 10 compares the execution times of these schemes. The best performance is achieved on CWM_CompSDQ, although the execution times of all three methods increase as the numbers of signatures grow. CWM_CompSDQ requires 12.58 s for matching 10,000 patterns. However, the execution time is increased to 189.62 s for matching 60,000 patterns. From the results of WM, the elapsed time marks 47.3 s for matching 10,000 patterns, but grows to 2940.72 s for 60,000 patterns. As a result, CWM_CompSDQ achieves a speedup factor of 15.51 compared to WM, 12.15 to CWM, and even 2.83 to CWM_SDQ on matching 60,000 patterns.

Unfortunately, CWM shows the worst performance on

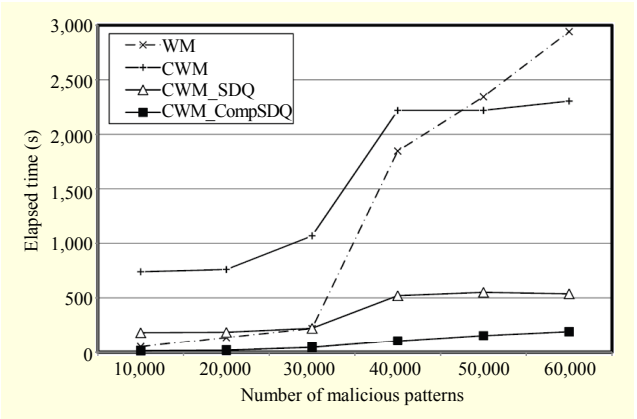

Fig. 10. ClamAV scanning time with respect to the number of malicious patterns.

ETRI Journal, Volume 37, Number 2, April 2015
http://dx.doi.org/10.4218/etrij.15.0114.0397

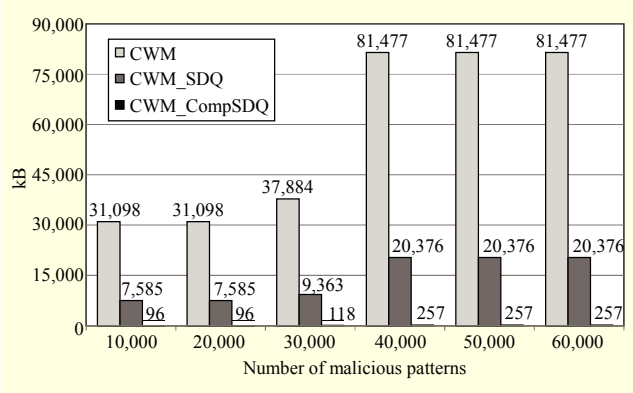

Fig. 11. Amount of transferring data to the cloud server.

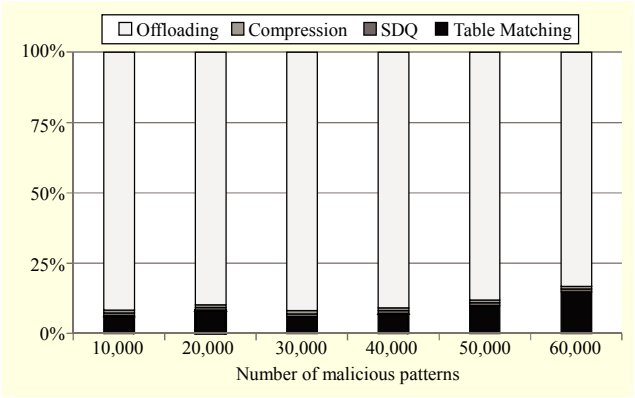

Fig. 12. Overhead for offloading heavy-weight functions of CWM_ CompSDQ.

matching patterns for less than 40,000 malicious patterns. This is due to the fact that the operations for transferring a large amount of suspicious data in CWM causes significant network delay and degrades the overall performance.

We measure the amount of data transferred to the server to observe how many data can be reduced by the proposed interleaved suspicious-data forwarding scheme and data compression technique. The results are shown in Fig. 11. On matching 60,000 patterns, CWM sends 81,477 kB of suspicious data to the remote server. This amount is almost equivalent to that of the target input data stream. In CWM_SDQ, the amount of data is significantly reduced to 20,376 kB. This means that the interleaved suspicious-data forwarding scheme successfully reduced the data size. Furthermore, in CWM_CompSDQ, which further reduces data size with lossless data compression, only 257 kB is transferred. This is only 12% of CWM_SDQ. As a result, CWM_CompSDQ achieves the best performance with significant reduction in data size.

In Fig. 12, we show a breakdown of the execution time for CWM_CompSDQ on the client side. In the figure, there is a network transmission overhead to offload heavy-weight functions, allocate SDQ, and compress the SDQ; the Table Matching operation is equivalent to that of WM. As shown in the figure, CWM_CompSDQ spends 92% of overall processing time on the offloading operation, which is mostly data transmission time. On the other hand, the SDQ and Compression amount to only a 1% portion, respectively. In fact, the offloading operation requires some time to establish connection to the offloading server and send suspicious data. As a result, it is obvious that CWM_CompSDQ spends most of its processing time on transferring suspicious data to the offloading server.

To observe energy consumption for matching a malicious signature-set on the mobile device, we measure average power consumption by using the PowerTutor tool [33]. The power

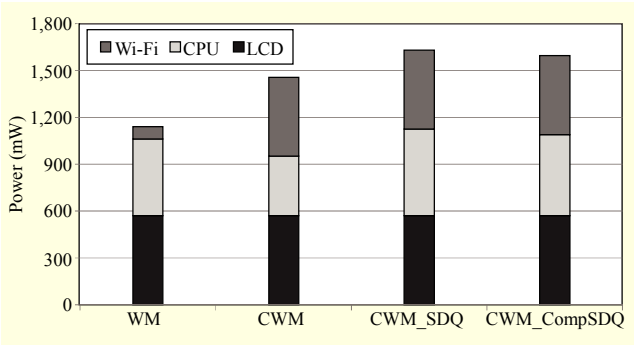

Fig. 13. Power consumption breakdown.

consumption breakdown is shown in Fig. 13. We measure power consumption in a unit time for three hardware components: LCD, CPU, and Wi-Fi. WM requires the least power at 1,139 mW since the Wi-Fi component is idle during matching. Among the CloudWM approaches, CWM consumes the least power than the others schemes utilizing SDQ. This is due to the fact that CWM should send much more data to the cloud server than the others and CPU is in the idle state during the sending time. In fact, the average power consumption of the CPU of CWM indicates 380 mW, while WM, CWM_SDQ, and CWM_CompSDQ score are at 489 mW, 553 mW, and 519 mW, respectively. Although WM and CWM show less power consumption than CWM_SDQ and CWM_CompSDQ, the CloudWM approaches utilizing SDQ consume less energy than WM and CWM for matching malicious patterns since their execution time is much less than WM and CWM.

We have measured the performance of CWM_CompSDQ on real cloud computing environments. The offloading servers are configured on the Amazon EC2 instances located in different cities all over the world, each of which has a different network bandwidth. Table 1 shows the execution times of CWM_CompSDQ as it cooperated with eight different

Table 1. Execution times for various environments.

Offloading Server (location)		Execution time (s)	Bandwidth (Mbits/s)	Distance (km)
Local server		43	14.80	0
Amazon EC2	Tokyo	53	9.70	1,163
	N. California	59	3.36	9,040
	Singapore	69	2.50	4,670
	Oregon	77	2.43	8,702
	Sydney	82	1.50	8,330
	N. Virginia	102	1.00	11,280
	Ireland	117	0.80	9,024
	Sao Paulo	127	0.72	18,340

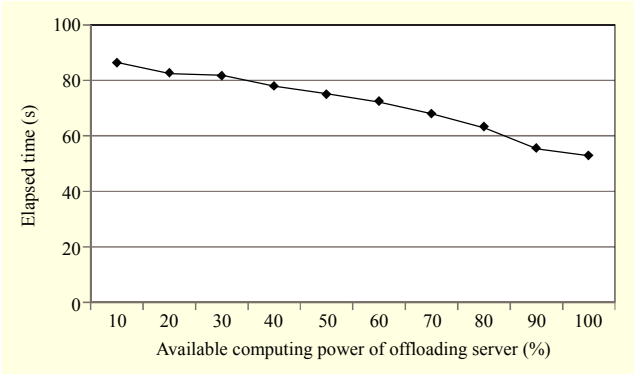

Fig. 14. Matching time of CWM_CompSDQ with various available computing power of the Amazon EC2 Tokyo server.

offloading servers. The bandwidths and distances from the client to one of the servers are also presented. In the table, the "Local server" item is the case that CWM_CompSDQ offloads the heavy computation part to the server connected in the same local network with the mobile client. The Amazon EC2 servers are named after the city in which they are located. For example, the offload server of "Tokyo" is located in Tokyo city in Japan.

The results show that the execution time is reduced as the bandwidth is increased. The "Local server" shows the highest bandwidth of 14.8 Mbits/s and the best performance — with an execution time of 43 s. Among the Amazon EC2 servers, Tokyo shows the best performance because of the shortest data transfer time due to the highest bandwidth among non-local servers. Note that the execution time is more sensitive to network bandwidth rather than physical distance between client and server. Although N. California is located the furthest from Singapore, since the bandwidth of N. California (3.36 Mbits/s) is larger than that of Singapore (2.50 Mbits/s), the execution time for N. California is 59 s compared to 69 s of Singapore.

It is interesting that even in the case of Sao Paulo, which shows the worst performance among the offloading servers, our approach still proves to be advantageous when compared to being executed only on a mobile device (218.12 s. of WM) with a speedup factor of 1.7 when matching 30,000 patterns. To summarize, our approach shows higher performance with an offloading server having low bandwidth, and when combined with offloading servers having higher bandwidth, it can achieve much better performance.

We have observed variations in the execution times in the case when an offloading server is processing multiple tasks concurrently and the partial portion of the total computation power of the server is available. Figure 14 shows the variation in execution time of CWM_CompSDQ with Tokyo for matching 30,000 patterns. With 100% server performance, CWM_CompSDQ completes the matching task in 52.9 s. On the other hand, when only 10% performance is allocated to the offloaded matching task, the execution time is increased to 86.4 s. This is 63% worse performance compared to the 100% case. However, with only 10% available computing power, CWM_CompSDQ still shows a speedup factor of 2.5 compared to WM running at the mobile device only.

V. Conclusion

In this paper, we proposed an efficient, secure mobile system assisted by cloud computing. To provide additional security on mobile devices, we designed a malware detection system based on a WM pattern matching algorithm. To optimize energy efficiency and maximize performance on mobile devices that have limited hardware resources, we presented a novel approach that offloads the heavyweight tasks of the algorithm to a resourceful cloud server. However, the overhead on data transfer between the mobile device and the server becomes a significant performance bottleneck in the system. To address this problem, an interleaved suspicious-data forwarding scheme was additionally proposed to minimize the size of the transmitted data. As a result, the proposed system achieves a speedup factor of 13.4 while consuming only 9% of power, compared to being run only on mobile devices.

References

[1] A. Mylonas, A. Kastania, and D. Gritzalis, "Delegate the Smartphone User? Security Awareness in Smartphone Platforms," *Comput. Security*, vol. 34, May 2013, pp. 47–66.

[2] A.P. Felt et al., "A Survey of Mobile Malware in the Wild," *ACM*

Workshop Security Privacy Smartphones Mobile Devices, Chicago, IL, USA, Oct. 17–21, 2011, pp. 3–14.

[3] M. La Polla, F. Martinelli, and D. Sgandurra, "A Survey on Security for Mobile Devices," *IEEE Commun. Surveys Tutorials*, vol. 15, no. 1, Feb. 2012, pp. 446–471.

[4] P.-C. Lin et al., "Using String Matching for Deep Packet Inspection," *Comput.*, vol. 41, no. 4, Apr. 2008, pp. 23–28.

[5] H. Kim and S.-W. Lee, "A Hardware-Based String Matching Using State Transition Compression for Deep Packet Inspection," *ETRI J.*, vol. 35, no. 1, Feb. 2013, pp. 154–157.

[6] R. Antonello et al., "Deep Packet Inspection Tools and Techniques in Commodity Platforms: Challenges and Trends," *J. Netw. Comput. Appl.*, vol. 35, no. 6, Nov. 2012, pp. 1863–1878.

[7] D. Oh and W.W. Ro, "Multi-threading and Suffix Grouping on Massive Multiple Pattern Matching Algorithm," *Comput. J.*, vol. 55, no. 11, Nov. 2012, pp. 1331–1346.

[8] L. Hoang and V.K. Prasanna, "A Memory-Efficient and Modular Approach for Large-Scale String Pattern Matching," *IEEE Trans. Comput.*, vol. 62, no. 5, May 2013, pp. 844–857.

[9] K. Kumar et al., "A Survey of Computation Offloading for Mobile Systems," *Mobile Netw. Appl.*, vol. 18, no. 1, Feb. 2013, pp. 129–140.

[10] K. Yang, S. Ou, and H.-H. Chen, "On Effective Offloading Services for Resource-Constrained Mobile Devices Running Heavier Mobile Internet Applications," *IEEE Commun. Mag.*, vol. 46, no. 1, Jan. 2008, pp. 56–63.

[11] M. Schmidt et al., "Malware Detection and Kernel Rootkit Prevention in Cloud Computing Environments," *Euromicro Int. Conf. Parallel, Distrib. Netw.-Based Process.*, Ayia Napa, Cyprus, Feb. 9–11, 2011, pp. 603–610.

[12] G. Chen et al., "Studying Energy Trade offs in Offloading Computation/Compilation in Java-Enabled Mobile Devices," *IEEE Trans. Parallel Distrib. Syst.*, vol. 15, no. 9, Sept. 2004, pp. 795–809.

[13] Y.-J. Hong, K. Kumar, and Y.-H. Lu, "Energy Efficient Content-Based Image Retrieval for Mobile Systems," *IEEE Int. Symp. Circuits Syst.*, Taipei, Taiwan, May 24–27, 2009, pp. 1673–1676.

[14] Z. Li and R. Xu, "Energy Impact of Secure Computation on a Handheld Device," *IEEE Int. Workshop Workload Characterization*, Nov. 25, 2002, pp. 109–117.

[15] S.H. Kim et al., "Offloading of Media Transcoding for High-Quality Multimedia Services," *IEEE Trans. Consum. Electron.*, vol. 58, no. 2, May 2012, pp. 691–699.

[16] I. Kim et al., "A Distributed Signature Detection Method for Detecting Intrusions in Sensor Systems," *Sensors*, vol. 13, no. 4, Mar. 2013, pp. 3998–4016.

[17] C.-H. Lin, P.-C. Hsiu, and C.-K. Hsieh, "Dynamic Backlight Scaling Optimization: A Cloud-Based Energy-Saving Service for Mobile Streaming Applications," *IEEE Trans. Comput.*, vol. 63, no. 2, Feb. 2014, pp. 335–348.

[18] S. Wu and U. Manber, "A Fast Algorithm for Multi-pattern Searching," University of Arizona, Techn. Rep., May 1994.

[19] J. Murty, "*Programming Amazon Web Services: S3, EC2, SQS, FPS, and SimpleDB*," Sebastopol, USA: O'Reilly Media, 2009.

[20] Z. Li, C. Wang, and R. Xu, "Computation Offloading to Save Energy on Handheld Devices: A Partition Scheme," *Int. Conf. Compilers, Archit., Synthesis Embedded Syst.*, Atlanta, GA, USA, Nov. 16–17, 2001, pp. 238–246.

[21] K. Kumar and Y.H. Lu, "Cloud Computing for Mobile Users: Can Offloading Computation Save Energy?," *Comput.*, vol. 43, no. 4, Apr. 2010, pp. 51–56.

[22] J. Oberheide et al., "Virtualized in-Cloud Security Services for Mobile Devices," *Workshop Virtualization Mobile Comput.*, Breckenridge, CO, USA, June 17, 2008, pp. 31–35.

[23] G. Portokalidis et al., "Paranoid Android: Versatile Protection for Smartphones," *Annual Comput. Security Appl. Conf.*, Austin, TX, USA, Dec. 6–10, 2010, pp. 347–356.

[24] A.-D. Schmidt et al., "Monitoring Smartphones for Anomaly Detection," *Int. Conf. MOBILe Wireless MiddleWARE, Operating Syst., Appl.*, Innsbruck, Austria, Feb. 13–15, 2008.

[25] J.-S. Lee, T.-H Kim, and J. Kim, "Energy-Efficient Run-Time Detection of Malware-Infected Executables and Dynamic Libraries on Mobile Devices," *Int. Workshop Softw. Technol. Future Dependable Distrib. Syst.*, Tokyo, Japan, Mar. 17–18, 2009, pp. 143–149.

[26] S.K. Cha et al., "Splitscreen: Enabling Efficient, Distributed Malware Detection," *J. Commun. Netw.*, vol. 13, no. 2, Apr. 2011, pp. 187–200.

[27] J. Cheng et al., "Smartsiren: Virus Detection and Alert for Smartphones," *Int. Conf. Mobile Syst., Appl. Services*, San Juan, PR, USA, June 11–14, 2007, pp. 258–271.

[28] X. Chen, B. Mu, and Z. Chen, "Netsecu: A Collaborative Network Security Platform for In-network Security," *Int. Conf. Commun. Mobile Comput.*, Qingdao, China, Apr. 18–20, 2011, pp. 59–64.

[29] S. Zonouz et al., "Secloud: A Cloud-Based Comprehensive and Lightweight Security Solution for Smartphones," *Comput. Security*, vol. 37, Sept. 2013, pp. 215–227.

[30] L. Yang, V. Ganapathy, and L. Iftode, "Enhancing Mobile Malware Detection with Social Collaboration," *IEEE Int. Conf. Privacy, Security, Risk Trust IEEE Int. Conf. Social Comput.*, Boston, MA, USA, Oct. 9–11, 2011, pp. 572–576.

[31] R.S. Boyer and J.S. Moore, "A Fast String Searching Algorithm," *Commun. ACM*, vol. 20, no. 10, Oct. 1977, pp. 762–772.

[32] T. Kojm, *Clam AntiVirus User Manual*, ClamAV, 2012. Accessed Dec. 14, 2013. http://www.clamav.net/doc/latest/clamdoc.pdf

[33] M. Gordon et al., *A Power Monitor for Android-Based Mobile Platforms*, 2013. Accessed Dec. 14, 2013. http://ziyang.eecs.umich.edu/projects/powertutor/

Doohwan Oh received his BS degree in electronic engineering from the College of Electronics and Information, Kyung Hee University, Suwon, Rep. of Korea, in 2007 and his MS degree in electronic engineering from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea, in 2010. He is currently enrolled in a PhD degree program at the School of Electrical and Electronic Engineering, Yonsei University. His current research interests are network security, pattern matching algorithms, and parallel processing on multi-core systems.

Ilkyu Kim received his BS and MS degrees in electrical and electronic engineering from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea, in 2011 and 2013, respectively. He currently works as a research engineer with the SoC Platform Technology Team, DTV SoC Developement Department, LG Electronics, Seoul, Rep. of Korea. His current research interests are system parts of current broadcasting systems and software optimization for data-intensive applications.

Keunsoo Kim received his BS degree in electrical and electronic engineering from the School of Electrical and Electronic Engineering, Yonsei University, Seoul, Rep. of Korea, in 2012. He is currently pursuing his PhD degree at the Embedded Systems and Computer Architecture Laboratory, School of Electrical and Electronic Engineering, Yonsei University. His industry experience includes a college internship at NAVER Corporation. His research interests are computer architecture and network applications.

Sang-Min Lee received his BS and MS degrees in electronic engineering from the School of Electrical Engineering and Computer Science, Kyungpook National University, Daegu, Rep. of Korea, in 1994 and 1996, respectively. He currently works as a principal researcher at the Electronics and Telecommunications Research Institute, Daejeon, Rep. of Korea. His current research interests are packet transport networks and smart Internet.

Won Woo Ro received his BS degree in electrical engineering from Yonsei University, Seoul, Rep. of Korea, in 1996. He received his MS and PhD degrees in electrical engineering from the University of Southern California, Los Angeles, USA, in 1999 and 2004, respectively. He has worked as a research scientist for the Electrical Engineering and Computer Science Department, University of California, Irvine, USA. He currently works as an associate professor at the School of Electrical and Electronic Engineering, Yonsei University. Prior to joining Yonsei University, he worked as an assistant professor at the Department of Electrical and Computer Engineering, California State University, Northridge, USA. His industry experience also includes a college internship at Apple Computer Inc., Cupertino, CA, USA and a work placement as a contract software engineer at ARM Inc., Irvine, CA, USA. His current research interests are high-performance microprocessor design, compiler optimization, and embedded system designs.