Hindawi
Wireless Communications and Mobile Computing
Volume 2018, Article ID 4734527, 11 pages
https://doi.org/10.1155/2018/4734527

Research Article

Pattern Matching Based Sensor Identification Layer for an Android Platform

Hong Min[1], **Taesik Kim**[2], **Junyoung Heo**[3], **Tomas Cerny**[4], **Sriram Sankaran**[5], **Bestoun S. Ahmed**[6] and **Jinman Jung**[7]

[1]*Division of Computer and Information Engineering, Hoseo University, Republic of Korea*
[2]*Department of Civil Engineering, Hongik University, Republic of Korea*
[3]*Division of Computer Engineering, Hansung University, Republic of Korea*
[4]*School of Engineering and Computer Science, Baylor University, USA*
[5]*Center for Cybersecurity Systems and Networks, Amrita Vishwa Vidyapeetham, Amritapuri, Kollam, India*
[6]*Department of Computer Science, Czech Technical University in Prague, Czech Republic*
[7]*Department of Information and Communication Engineering, Hannam University, Republic of Korea*

Correspondence should be addressed to Jinman Jung; jmjung@hnu.kr

Received 30 April 2018; Revised 18 August 2018; Accepted 23 September 2018; Published 10 October 2018

Academic Editor: Jun Cai

As sensor-related technologies have been developed, smartphones obtain more information from internal and external sensors. This interaction accelerates the development of applications in the Internet of Things environment. Due to many attributes that may vary the quality of the IoT system, sensor manufacturers provide their own data format and application even if there is a well-defined standard, such as ISO/IEEE 11073 for personal health devices. In this paper, we propose a client-server-based sensor adaptation layer for an Android platform to improve interoperability among nonstandard sensors. Interoperability is an important quality aspect for the IoT that may have a strong impact on the system especially when the sensors are coming from different sources. Here, the server compares profiles that have clues to identify the sensor device with a data packet stream based on a modified Boyer-Moore-Horspool algorithm. Our matching model considers features of the sensor data packet. To verify the operability, we have implemented a prototype of this proposed system. The evaluation results show that the start and end pattern of the data packet are more efficient when the length of the data packet is longer.

1. Introduction

Modern smartphones are equipped with various sensors such as an accelerometer, gyroscope, and Global Positioning System (GPS) that can track and monitor dynamic environmental information including user's behavior [1], health [2], traffic [3], noise pollution [4], and social interactions [5]. Nowadays, smartphones are not only a communication device but also a monitoring device for various purposes [6]. With the rapid development of sensor devices, wired or wireless external sensors can be connected to a smartphone in the Internet of Things (IoT) environments. For example, an application using Bluetooth Low Energy (BLE) beacons can support several services including localization detection, interaction, and activity sensing [7]. Smartphones can easily collect data from the external sensors deployed around a user and physical space because it has an interface and communication capabilities [8]. In broader aspects, smartphones can be used as gateways in Smart Cities, which are a key application domain for the Internet of Things (IoT) [9]. Smartphones cooperate with various types of sensors in this large-scale system and it is impractical to recognize and interpret meaningful information from each sensor's data packet. In addition, smartphones and IoT devices are always connected to the Internet and their software must be changed to update system files or fix bugs [10]. During the data collection process and software management, many security and privacy-related issues are also considered [11].

Interworking a smartphone among sensors has various merits. However, the packet format of each sensor device

- TELAiRE T6703 Carbon Dioxide Sensor

0	1	2	3	4	5	6	7
Start		Addr	Length	Data		CSum	
0xFF	0xFF	0xFE	0x02	CO_2_H	CO_2_L	Csum_H	Csum_L

- Epson M-G350-PD Triple Gyroscopes Sensor

0	1	2	3	4	5	6	7
Start	Xgyro		Ygyro		Zgyro		End
0x02	Xgyro_H	Xgyro_L	Ygyro_H	Ygyro_L	Zgyro_H	Zgyro_L	0x0D

- Texas Instruments OPT3001 Light Sensor

0	1	2	3	4	5	6	7	8
Start			Addr	Data				CSum
0x80	0x07	0x3A	0x62	Lux_0	Lux_1	Lux_2	Lux_3	CSum

-HANTECH PPD4NS Dust Sensor

0	1	2	3	4	5	6	7	8	9
Start		ID	Data				Pad	CSum	End
0xAA	0xB4	Dev_ID	Dust_0	Dust_1	Dust_2	Dust_3	FF	CSum	0xAB

- Crossbow NAV420 GPS Sensor

0	1	2	3	4	5	6	7	8	9	10	11	12
Start		Type	Longitude				Latitude				CSum	
0x55	0x55	'N'	Long_0	Long_1	Long_2	Long_3	Lat_0	Lat_1	Lat_2	Lat_3	Csum_H	Csum_L

- Hanback HBE-701 Blood Pressure Sensor

0	1	2	3	4	5	6	7	8	9	10	11	12
Start		ID		Data						CSum	End	
0x76	0x00	0x11	0x08	High_PH	High_PL	Low_PH	Low_PL	Pules_H	Pules_L	Csum	0xF0	0x0F

- EM MICROELECTRONIC EMBC-MN01 Temperature and Humidity Sensor

0	1	2	3	4	5	6	7	8	9	10	11	12	13
Start		ID		Address			Length	Temperature		Humidity		End	
0x0C	0xF3	0x15	0x0C	Addr_0	Addr_1	Addr_2	0x04	Temp_H	Temp_L	Hum_H	Hum_L	0xEC	0xEC

Figure 1: Data packet format of each manufacturer.

is different even among personal health devices. There are messaging standards for personal health devices, such as the ISO/IEEE 11073 Personal Health Device (PHD) [12] and Health Level Seven [13]. These standards define a common framework for data interoperability in order to establish connections between personal health sensors or devices to meet the IoT quality aspects [14]. Yet, many manufacturers usually provide many different message formats for each device to support certain applications. This, in turn, leads to interoperability issues when the number of sensors increases.

In this paper, we propose a client-server-based sensor adaptation layer to support interoperability of nonstandardized sensor devices. The server plays the role of identifying sensor devices by using a pattern matching process based on the Boyer-Moore-Horspool (BMH) algorithm to recognize the received data packet in the proposed system. The server also manages and stores the profile of each sensor that is used to identify an unknown data packet to its database. This profile reflects two features of a data packet: unique start and end pattern and the range value between the minimum and maximum value measured by a sensor. The client running on a smartphone receives the result from the server. The client plays the role of extracting desirable value from the data packet and reformatting the standard message based on ISO/IEEE 11073 PHD by using the result. In the case of an application, it is not changed when a new sensor device is connected because the client decodes the data packet and sends a reformatted standard packet to the application.

The rest of this paper is organized as follows. Section 2 introduces the background and related information. Section 3 presents the client-server-based sensor adaptation layer for interoperability among different sensors. Section 4 is an evaluation of the availability of the proposed system and time efficiency of sensor identification. Finally, Section 5 summarizes the paper and gives concluding remarks.

2. Background and Related Works

2.1. Data Packet Formats and Standard. In practice, sensors are having different data format. In fact, each sensor has its own data packet format even though sensors are made by the same manufacturer. Figure 1 shows different packet formats from various manufacturers. These data packets have different features such as the length, the start and end pattern, and the range of data values. Application developers understand the data packet format of each sensor and modify source code when a sensor is changed. The mobile device that connected to the sensors cannot only interwork with unknown sensors but also install new applications whenever a new sensor is attached.

invok_id (2bytes)	msg_type (2bytes)	length_event (2bytes)	obj_handle (2bytes)
currentTime (4bytes)		event_type (2bytes)	length_value (2bytes)
addr_value (pointer/4bytes)			

Figure 2: ISO/IEEE 11073 data packet structure.

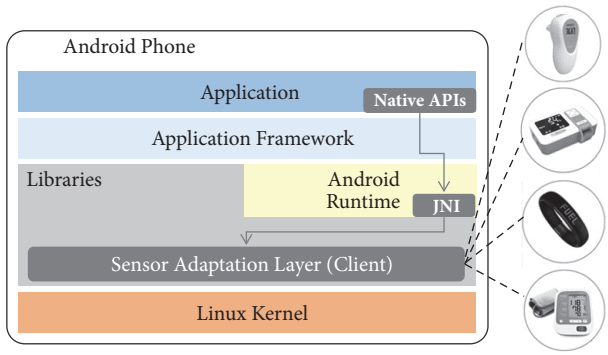


Figure 3: Android architecture overview and sensor adaptation layer.

To support interoperability among various PHDs, the ISO/IEEE 11073 was proposed. The standard specifies how personal health data should be exchanged between sensors and a monitor such as a smartphone and what standard data format should be used [15]. This standard is available for various u-health services such as disease management, health and fitness, and antiaging applications. The standard consists of a technical report, device specifications, and an optimized exchange protocol between a sensor device and a monitor. Figure 2 shows the data packet structure of an open platform based on ISO/IEEE 11073 [16].

In spite of the ISO/IEEE 11073 committee efforts to standardize various data packet formats, the data format of existing sensors is nonstandard. Many manufacturers and u-health service providers still use their own data formats, and their sensor devices cannot be regarded as interoperable. This causes increases in maintenance costs and prevents integrated system implementation [17].

2.2. Sensors' Connection to Android Platform. The Android platform is composed of components and layered in five layers. As shown in Figure 3, these layers are applications, application framework, Android runtime, native libraries, and Linux kernel [18]. The application layer may include different Android applications written in Java, such as email client, Short Message Service (SMS), calendar, maps, web browser, contacts, and others. The application framework provides an application programming interface (API) to developers, which is designed to simplify the reuse of components and to communicate with various devices. Android runtime (ART) includes core libraries and virtual executable environments of Dex bytecode as its predecessor Dalvik. The Android also includes a set of core libraries that provides most of the functionalities available in the core libraries of the Java programming language. The ART is similar to the Java Virtual Machine (JVM) and every application needs ART in order to run. Libraries include a set of C/C++ libraries used by various components of the Android system. The Android relies on Linux for the core system services such as security, memory management, process management, the network stack, and device drivers.

In this study, we use the Java Native Interface (JNI) framework that enables Java application running in a JVM to call native libraries written in C and C++ for interworking an Android-based smartphone with external sensors. The sensor adaptation layer (SAL) monitors serial ports and Bluetooth interface to detect connection requests of PHDs and sends ISO/IEEE 11073 compatible messages to the application by assisting the server. The details are described in the next section.

2.3. Packet Matching Algorithms. Sensor devices periodically send numerous data packets to the smartphone. An Android application that receives data packets from sensor devices unpacks meaningful value such as temperature, blood pressure, and heart rate. It is possible to recognize the meaningful value if the application is already known in the data packet structure. However, it is impossible to decompose the data packet if the application receives an unknown data packet. Therefore, the server part of the proposed SAL uses a packet matching algorithm called Boyer-Moore-Horspool (BMH) [19], when there is no predefined information and profile about a new coming data packet.

A packet matching problem is a kind of pattern matching. An exact string matching algorithm is needed to recognize an unknown data packet from an unidentified sensor. There are two types of string matching algorithms: online and offline algorithms. In the case of an online scheme, the compared text is not known in advance, whereas in the case of an offline scheme, the compared text can be known and preprocessed [20]. The BMH algorithm is an online scheme that requires $O(mn)$ in the worst case but $O(n/\#\ of\ entire\ compared\ characters)$ in the average case, where m, n represent the length of each pattern. These characteristics of the BMH algorithm are proper to our data packet matching module.

2.4. Sensor Identification Schemes. Fountain et al. proposed a technique of automatically identifying relationships among different positioning sensors [21]. This technique allows for subsequent automatic alignment of the sensor systems and increasing the usability of modular sensor systems. To improve the accuracy of identifying, the authors use an invariant functional method and calibration error method. The invariant functional method uses a sliding window mechanism to discard noises and the calibration error method is used to find matching part of sensor streams. This scheme can improve its performance by increasing window size but our scheme focus on maximizing the number of shifts (jumping compared characters).

A clustering algorithm for unbounded sensor data is designed by Mirsky et al. to accurately infer the present context [22]. This clustering algorithm is suitable for sensor data streams because it can detect overlapping clusters. The authors consider the temporal relation of the arriving points in the clustering decision and clusters' correlated distributions. By using these mechanisms, sensor data stream is partitioned dynamically according to both temporal and spatial domains during the clustering decision process and correlated distributions. The clustering algorithm is efficient in terms of the completion time but the accuracy is fluctuated as input data.

Perera et al. proposed a tool called SmartLink that can be used to discover and configure sensors in heterogeneous IoT environments [23]. SmartLink follows Context-aware Dynamic Discovery of Things (CADDOT) model that consists of 8 phases: detect, extract, identify, find, retrieve, register, reason, and configure. SmartLink becomes an open wireless hotspot and detects sensors. Next, SmartLink extracts information from detected sensors and sends extract information to a cloud server. After the cloud server identifies the server, a plugin that describes how to communicate with identified sensors is installed. Once all the information about a detected sensor has been collected, registration takes place in the cloud. In our approach, we use regular expression-based string match algorithm to identify sensor data but previous studies are based on probability mechanisms. We also design the client-server-based system architecture as well as the sensor identification mechanism.

Liu et al. [24] proposed a naming, addressing, and profile server called NAPS to give a unique name to each IoT device. The role of NAPS is similar to Domain Name Server (DNS). When a device registers into NAPS, NAPS generates the profile of the registered device as XML format. If a user requests a profile to NAPS by using RESTful interface, a user can receive all information about the device including name, address, communication protocol, device type, sampling interval, and so on. This system provides the detailed information of an identified sensor device by using a predefined profile, but the profile format is different from our approach.

Liu et al. [25] presented a centralized event detection algorithm based on Minimum Cut (Min-cut) theory and Support Vector Machine- (SVM-) based pattern recognition technique to reduce communication overhead during data collection. In the proposed algorithm, Min-cut theory is used for reducing the number of transmissions of redundant data and SVM is used for determining that a participant is in the event region or not. The time for training and accuracy for new sensor data formats cannot be exactly estimated because this scheme employs machine learning-based algorithms.

3. Client-Server Based Sensor Adaptation Layer

3.1. The Client-Server Architecture. The proposed client-server system is composed of several modules. Figure 4 shows these modules in the architecture clearly. The client consists of three modules and the connection management module is used for sending a data packet stream that is received from an unknown sensor device to the server and receiving the matching results from the server. To implement the connection management module, we use Transmission Control Protocol/Internet Protocol (TCP/IP) based socket library running on Linux. The sensor detection module periodically monitors serial ports and the Bluetooth interface. If a sensor device is attached to the Android platform, a new device file is created in the /dev directory. In the case of the Bluetooth interface, we use the BlueZ library and its utilities for detecting a connection over the Bluetooth interface. BlueZ provides functions to detect Bluetooth devices and collect Bluetooth data packets [26]. The reformatting data packet module repacks raw data packets from a sensor device to ISO/IEEE 11073 compatible data packets by using the matching result from the server. Reformatted data packets are sent to the upper layer by using JNI and the application can extract meaningful values from the standardized data packet.

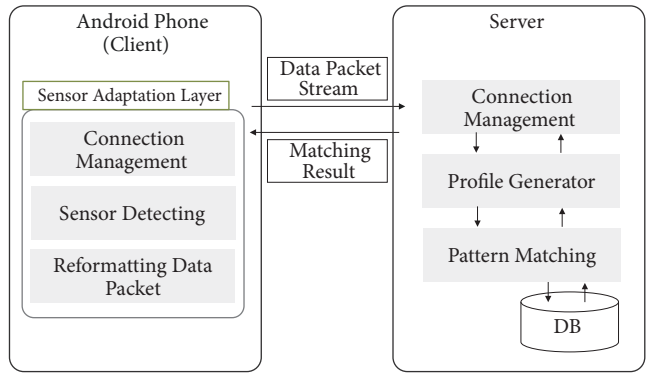


Figure 4: Android architecture overview and sensor adaptation layer.

The server also consists of three modules. The connection management accepts requests from multiple clients and prepares to receive the data packet stream from clients. The profile generator plays an important role in the performance of sensor identification. The profile means unique characteristics of each data packet such as a fingerprint and is represented as a regular expression format [27]. The data packet stream from a sensor device has some unique pattern that is repeatedly shown such as the start and end pattern, type, and length. The profiles of each sensor are shown in Figure 1 and Table 1. We found two features from the profiles. In the first feature, the regular expression of each sensor data packet starts with a unique pattern and includes a different number of wildcard characters. In the second feature, the rage of sensors' data values is different. For instance, the value range of the GPS sensor is from -180 to 180, and the blood pressure sensor is from 20 to 300. These two features of the profile are used to improve the efficiency of the pattern matching process. The pattern matching module compares a sensor's profile with the database that stores the raw data packet of all kinds of sensors. This pattern matching process is described in the next modeling subsection.

Table 1: Sensor profiles.

Sensor	Profile (regular expression)
TELAiRE T6703Carbon Dioxide Sensor	FFFF([0-9A-F]{2})02[400-5000]([0-9A-F]{2})
Epson M-G350-PD Triple Gyroscopes Sensor	02([-300-300]{3})0D
Texas Instruments OPT3001 Light Sensor	80073A([0-9A-F]{2})[250-600].
HANTECH PPD4NS Dust Sensor	AAB4([0-9A-F]{2})[0-255]([0-9A-F]{2})FFAB
Crossbow NAV420 GPS Sensor	55{2}.[-180-180]{2}) ∗ ∗
Hanback HBE-701 Blood Pressure Sensor	7600([0-9A-F]{4})([20-300]{2})[40-199] ∗ 00F
EM MICROELECTRONIC EMBC-MN01 Temperature and Humidity Sensor	0CF3([0-9A-F]{10})04[0.25-127][0-100](EC{2})

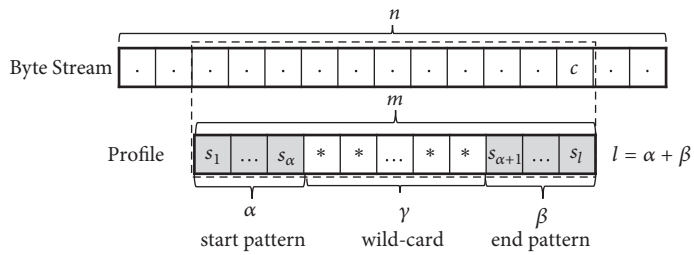


Figure 5: Profile for sensor identification.

3.2. Modeling and Analysis. We present an estimate for the efficiency of our modified BMH algorithm for identifying sensor device. We model the efficiency in terms of the expected value of a shift that determines how much to shift each pattern when a mismatch occurs. We can simplify the analysis by assuming uniform distributions of characters in both the byte streams and all profiles. In practice, these are not random, but its probabilistic performance gives a rough idea.

The key insight of the BMH algorithm is to match on the tail of the pattern rather than the head and to skip along the text in jumps of multiple characters rather than checking every character of the byte streams. Jumping along the text rather than checking every character decreases the number of comparisons that have to be performed, which is a key to the efficiency of the algorithm. In a BMH algorithm, the amount that we can jump can be computed by using the formula *J(c)*, which is the distance from *c*'s rightmost occurrence in pattern among its first *m-1* characters to its right end.

In a sensor's profile, we are given a sensor profile *m* characters long with $S[s_1 \dots s_\alpha]$ at the start and $S[s_{\alpha+1} \dots s_{\alpha+\beta}]$ at the end, which may contain wildcard characters as shown in Figure 5. Note that the wildcard characters are used to substitute any other character(s) in a string. There is a key difference to generate jump tables that determine how much to shift each pattern when a mismatch occurs. Our goal is to find out the first match of the string profile in the byte stream n characters in length (alternatively called byte stream).

Therefore, we modify the jump table's construction part of the BMH to handle the sensor's profiles. We preprocess the sensor patterns and build profiles for the sensor identification. Then, we feed the byte stream to the server, which reports the matching result whenever one is found.

Here, we denote the size of the set of the characters by *s*. We assume that both byte streams and all profiles are random strings with uniform distributions.

$$p_r\,(\textit{any given character in byte stream}) = \frac{1}{s} \text{ where the size of the characters is } s \quad (1)$$

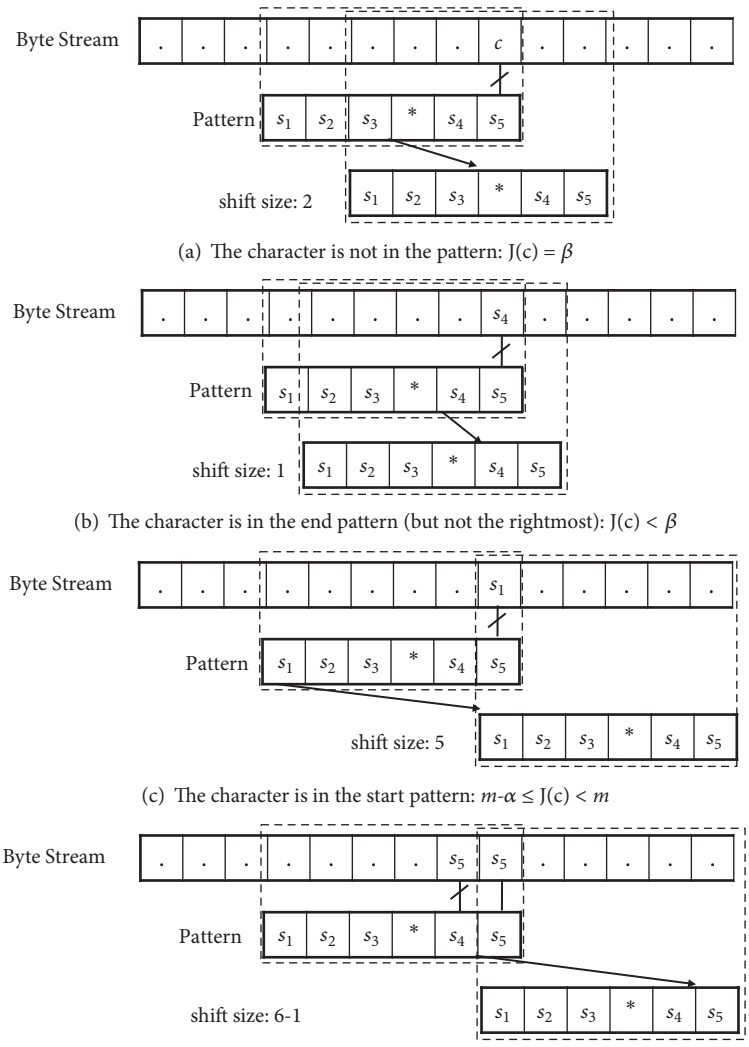
We first consider the probability of *J(c)* and then estimate its expected value. The possible cases range from $J(c) = 1$ to $J(c) = m$. We can summarize the value for four possible cases as shown in Figure 6.

(a) The character c is the end pattern (but not the rightmost). In that case, the probability that the function $J(c) = x\ (0 < x < \beta)$ is equal to the probability that each the last $(x - 1)$ characters differs from the corresponding characters of end patterns (except for the rightmost) until the first *c*' occurrence.

(b) The character c is not in both start and end patterns. In that case, the value of J(*c*) is identical to the number of characters in end pattern, β. $J(c) = \beta$ occurs when all l characters in both start and end patterns mismatch.

(c) The character *c* is in the start pattern. The *J(c)* range from $m - \alpha$ to $m - 1$ for s_α and s_1, respectively. The probability that the function $J(c) = x\ (m - \alpha \le x < m)$ is equal to the probability that does not occur in $(x - \gamma - 1)$ characters of the pattern at all (except for the rightmost) and one final character matching. Recall that the gamma is the number of wildcards and it can substitute any other character(s) in our algorithm.

(d) The character *c* is only in the rightmost character of the pattern. In that case, *J(c)* is the maximum possible shift



(a) The character is not in the pattern: J(c) = β

(b) The character is in the end pattern (but not the rightmost): J(c) < β

(c) The character is in the start pattern: m-$\alpha \leq$ J(c) < m

(d) The character is in the rightmost character of pattern: J(c) = m

FIGURE 6: An example of a profile matching (m=6, α=3, β=2, γ=1).

distance, m. The probability is when the character c only occurs in the rightmost of end pattern.

$$p_r(J(c) = x) = \begin{cases} p \bullet (1-p)^{x-1} & \text{if } 0 < x < \beta \\ (1-p)^{\alpha+\beta} & \text{else if } x = \beta \\ p \bullet (1-p)^{x-\gamma-1} & \text{else if } m - \alpha \leq x < m \\ p \bullet (1-p)^{\alpha+\beta-1} & \text{else if } x = m \end{cases} \quad (2)$$

Thus, it is clear that the sum of the probabilities of all x outcomes must be 1 (see Appendix):

$$\sum_{x=1}^{m} p_r(J(c) = x) = 1 \quad (3)$$

Accordingly, the expected value of the shift function can be computed as (see Appendix):

$$E[J(c)] = \left[\frac{1}{p} \bullet \left\{1 - (1-p)^{\beta-1}\right\} - (\beta - 1) \bullet (1-p)^{\beta-1}\right] + \beta \bullet (1-p)^{\alpha+\beta} + (1-p)^{\beta-1} \bullet \left[\left\{(\beta+\gamma) + \frac{1}{p} \bullet (1-p)\right\} - (1-p)^{\alpha} \bullet \left\{\frac{1}{p} + (m-1)\right\}\right] + m \bullet p \bullet (1-p)^{\alpha+\beta-1} \quad (4)$$

Figures 7(a)–7(c) show the expected number of shifts with various γ with given α and β that range from 1 to 3, respectively. With a given β, expectations increase with γ and the slope of each curve increases with α. The value of β has a significant impact on the y-intercept, which is approximately the same as the value of β. Note that the expectation of a naive algorithm is a constant value of 1. Accordingly, the proposed algorithm herein has an advantage over the naive algorithm.

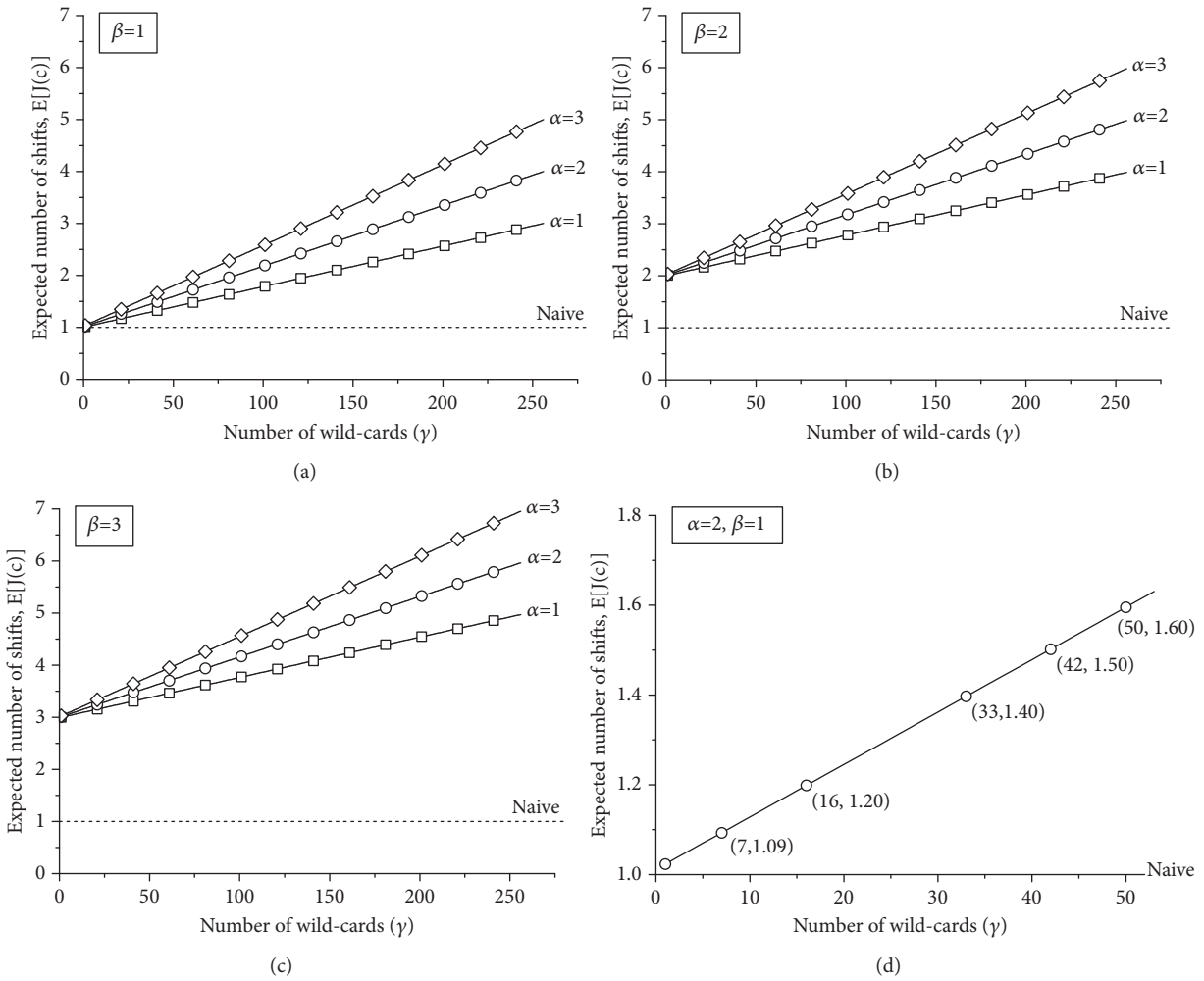


FIGURE 7: Expected number of shifts with various γ: (a) $\alpha = 1, 2, 3$ and $\beta = 1$, (b) $\alpha = 1, 2, 3$ and $\beta = 2$, (c) $\alpha = 1, 2, 3$ and $\beta = 3$, and (d) $\alpha = 2$ and $\beta = 1$.

Figure 7(d) shows the results of the practical case in that the frequently used values of α and β are 2 and 1, respectively. As shown in the figure, when the value of γ is 16, the proposed mechanism is 20% better than the naive. In our investigations, when the value of β is 2 or more or γ is significantly large, the proposed algorithm herein is powerful.

4. Prototype Implementation and Performance Evaluation

Figure 8 shows the hardware components of our prototype and Table 2 shows the client and server specifications. Temperature, dust, and carbon dioxide sensors are connected to an Android-based development board with serial cables. A blood pressure sensor is connected to the development board over the Bluetooth communication. The SAL read data packet streams from device files whenever one of the sensors is attached. After receiving a pattern matching result from the server, the SAL parses the sensing value from the data packet stream and repacks the message based on the ISO/IEEE 11073 standard.

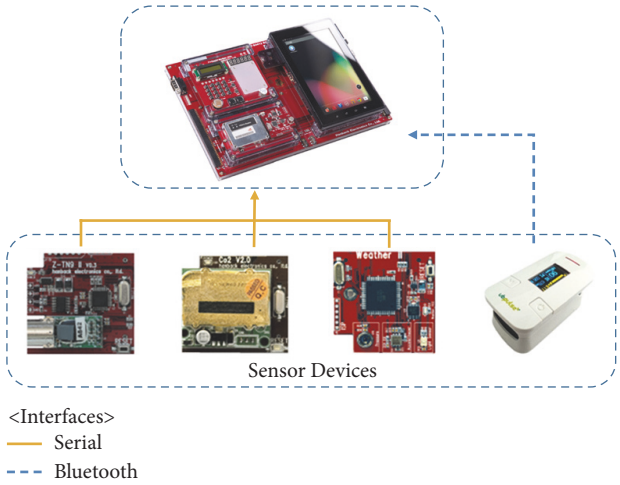


FIGURE 8: Hardware components of our prototype.

After receiving the standard message from the SAL, an application is running on the board and plots the graph with

Table 2: System specification.

Items		Specification
Android platform (Client)	Model	HBE-SM7-S4412
	CPU	Samsung Exynos 4412 1.7GHz
	Memory	2GB LP-DDR2 880MHz
	GPU	ARM Mali 400MP 440MHz
	LCD	7 inch 800X1280
	Storage	eMMC 16GB
	Wireless	802.11b/g/n Wireless LAN, Bluetooth 4.0
	OS	Android 4.1.2 (Jelly Bean)
	Sensors	(i) CO_2: TELAiRE T6703 (ii) Dust: HANTECH PPD4NS (iii) Blood pressure: Hanback HBE-701 (iv) Temperature: EM MICROELECTRONIC EBMC-MN01
Server	CPU	Intel Xeon E3-1220v3 4-Core 3.10GHz
	Cache	8MB L3 Cache
	Memory	4GB
	Storage	1TB
	OS	Ubuntu 16.04 LTS
	DB	MySQL Community Server 5.7

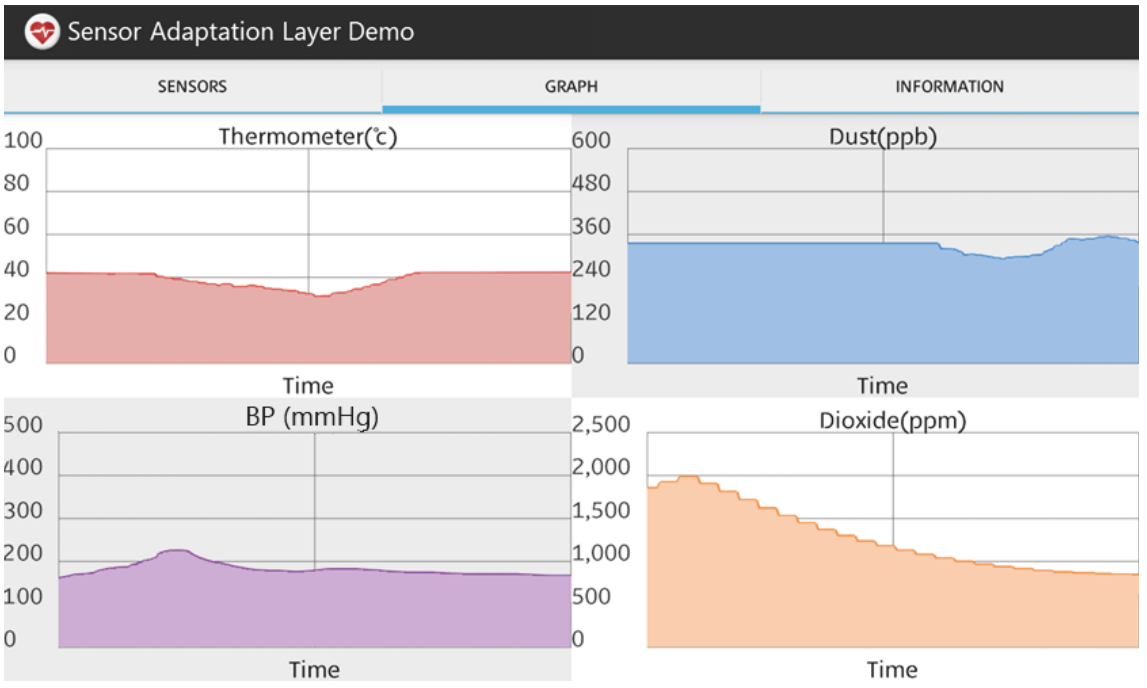


Figure 9: Screen capture of our prototype.

the sent value from the SAL as shown in Figure 9. If we randomly turn on and off one of the sensors, the application starts and stops to plot the graph immediately. The application is not modified even though a new sensor is attached.

Figure 10 shows server side execution time comparison result of each scheme. The total number of sample data packets is between 1000 and 10000. Sequential scheme compares the profile data packet with a sample data packet per a byte. In the case of sliding window [21], we set the window size as 1/3 of the profile data packet. Clustering algorithm [22] selects a sample data packet that has the highest similarity to the profile data packet. In probability scheme [23], positions for comparing the profile packet and sample data packets are randomly selected. Our scheme

Table 3: Accuracy comparison.

	Sequential				Sliding window				Clustering				Probability (random)				Our scheme			
	CO_2	Dust	BP	TEMP	CO_2	Dust	BP	TEMP	CO_2	Dust	BP	TEMP	CO_2	Dust	BP	TEMP	CO_2	Dust	BP	TEMP
Min (%)	100	100	100	100	70	85	60	45	45	50	15	15	10	10	5	5	65	85	85	85
Max (%)	100	100	100	100	85	90	75	65	55	65	35	40	35	40	25	20	80	95	95	95
Average (%)	100	100	100	100	76	87	65	57.5	48.5	57	22	26	20.5	19.5	12.5	10.5	76.5	89	89	91.5
STD deviation	0	0	0	0	5.83	2.45	6.7	8.4	3.2	6	7.14	8.6	7.23	9,6	6.4	6.5	4.5	3.74	3	3.2

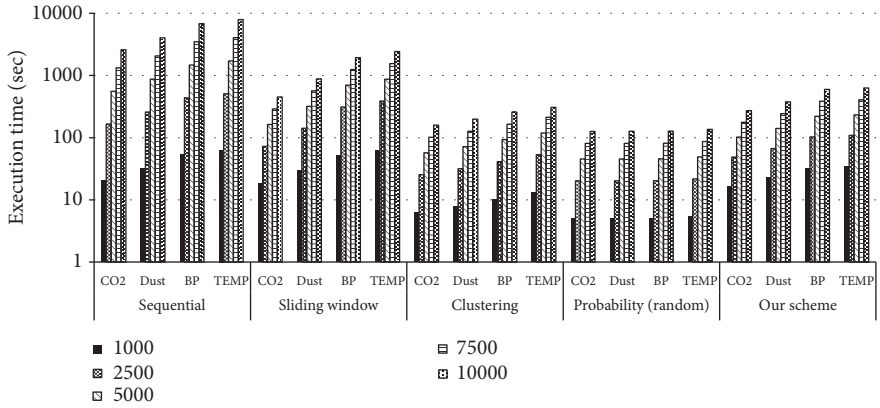


Figure 10: The execution time of profile matching process.

compares the proposed regular expression-based profile with sample data packets.

The execution time of the sequential scheme increases dramatically as the packet length and the total number of sample data packets increase. Sliding window scheme shows better performance than sequential scheme but worse than other schemes. In this scheme, the windows size is an important role in determining the execution time. It is difficult to decide proper window size considered variable data packet size and changes of sensing values. Clustering and probability schemes are scalable because these schemes do not check the entire data packet but only some parts of the data packet. Our scheme is worse than clustering and probability schemes but better than sequential and sliding window schemes. Our scheme is also scalable because it skips comparing some parts by using the proposed regular expression-based profile.

Table 3 summarizes the pattern matching accuracy of each scheme. We tested under conditions that the total number of sample data packets is 5,000 including 20 data packets for each sensor (CO_2, dust, blood pressure (BP), and temperature (TEMP), and total 80 packets) and tried 10 times for each scheme. The accuracy is the number of collect recognition over the number of data packets of each sensor (20). In the case of the sequential scheme, all sensors are intensified accurately. In the sliding window scheme, as window size decreases and the length of data packet increases, the accuracy decreases. In the case of clustering and probability scheme, these schemes show low accuracy to recognize a sensor data packet because it is difficult to set up proper classification criteria and a probability model for sensor data stream. Our scheme shows similar accuracy to the sequential scheme except for CO_2 sensor that does not have the end pattern of the data packet.

5. Conclusions and Future Work

Sensor devices are used for many applications, especially with smartphones. Numerous manufacturers provide diverse sensor devices and use their own data packet formats. Applications that work with these sensor devices are modified whenever the attached sensor device is changed because the data format is different. We proposed a client-server-based sensor adaptation layer for an Android platform to solve this problem. The proposed interface, called SAL, reforms the nonstandard data packet to the standard data packet by assisting the server. The server defines profiles that are used for identifying sensor devices and represents a regular expression and conducts a pattern matching process based on the modified Boyer-Moore-Horspool algorithm. Our model and analysis results show that the profiles reflect features of the sensor data packet and the length of the end pattern highly affects the performance of the matching process. We also implement a prototype to show that our system design is operable. The future work of our system is generating regular expressions to add new types of sensors. We predefine and register regular expressions of known sensors manually. To make our system more practical, we should automatize recognizing sensor data stream and generate regular expression from a given sensor data stream.

Appendix

(i) Equation (3)

$$\begin{aligned}
&\sum_{x=1}^{m} p_r\left(J(c)=x\right) \\
&= \sum_{x=1}^{\beta-1} p\bullet(1-p)^{x-1} + (1-p)^{\alpha+\beta} \\
&\quad + \sum_{x=m-\alpha}^{m-1} p\bullet(1-p)^{x-\gamma-1} + p\bullet(1-p)^{\alpha+\beta-1} \\
&= \sum_{x=1}^{\beta-1} p\bullet(1-p)^{x-1} + (1-p)^{\alpha+\beta} + p \\
&\quad \bullet\left\{\sum_{x=1}^{m-1}(1-p)^{x-\gamma-1} - \sum_{x=1}^{m-\alpha-1}(1-p)^{x-\gamma-1}\right\} + p \\
&\quad \bullet(1-p)^{\alpha+\beta-1} \\
&= \left\{1-(1-p)^{\beta-1}\right\} + (1-p)^{\alpha+\beta}\, p \\
&\quad \bullet\left\{\frac{1-(1-p)^{m-\gamma-1}}{p} - \frac{1-(1-p)^{\beta-1}}{p}\right\} + p \\
&\quad \bullet(1-p)^{\alpha+\beta-1} \\
&= \left\{1-(1-p)^{\beta-1}\right\} + (1-p)^{\alpha+\beta} \\
&\quad + \left\{(1-p)^{\beta-1} - (1-p)^{m-\gamma-1}\right\} + p \\
&\quad \bullet(1-p)^{\alpha+\beta-1} \\
&= 1 + (1-p)^{\alpha+\beta} - (1-p)^{\alpha+\beta-1} + p \\
&\quad \bullet(1-p)^{\alpha+\beta-1} \\
&= 1 + (1-p)^{\alpha+\beta} - (1-p)^{\alpha+\beta-1}(1-p) = 1
\end{aligned} \tag{A.1}$$

(ii) Equation (4)

$$\begin{aligned}
E\left[J(c)\right] &= \sum_{x=1}^{m} x\bullet p_r\left(J(c)=x\right) = \sum_{x=1}^{\beta-1} x\bullet p \\
&\quad \bullet(1-p)^{x-1} + \beta\bullet(1-p)^{\alpha+\beta} + \sum_{x=m-\alpha}^{m-1} x\bullet p \\
&\quad \bullet(1-p)^{x-\gamma-1} + m\bullet p\bullet(1-p)^{\alpha+\beta-1} = \left[\frac{1}{p}\right. \\
&\quad \left.\bullet\left\{1-(1-p)^{\beta-1}\right\} - (\beta-1)\bullet(1-p)^{\beta-1}\right] + \beta \\
&\quad \bullet(1-p)^{\alpha+\beta} + (1-p)^{\beta-1} \\
&\quad \bullet\left[\left\{(\beta+\gamma) + \frac{1}{p}\bullet(1-p)\right\} - (1-p)^{\alpha}\right. \\
&\quad \left.\bullet\left\{\frac{1}{p} + (m-1)\right\}\right] + m\bullet p\bullet(1-p)^{\alpha+\beta-1}
\end{aligned} \tag{A.2}$$

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research is supported by the Academic Research Fund of Hoseo University in 2014 (2014-0433).

References

[1] O. Yurur, C. H. Liu, Z. Sheng, V. C. Leung, W. Moreno, and K. K. Leung, "Context-Awareness for Mobile Sensing: A Survey and Future Directions," *IEEE Communications Surveys & Tutorials*, vol. 18, no. 1, pp. 68–93, 2016.

[2] J. P. Higgins, "Smartphone Applications for Patients' Health and Fitness," *American Journal of Medicine*, vol. 129, no. 1, pp. 11–19, 2016.

[3] S. Hu, L. Su, H. Liu, H. Wang, and T. F. Abdelzaher, "Smartroad: smartphone-based crowd sensing for traffic regulator detection and identification," *ACM Transactions on Sensor Networks*, vol. 11, no. 4, pp. 1–27, 2015.

[4] S. S. Kanhere, "Participatory sensing: crowdsourcing data from mobile smartphones in urban spaces," in *Proceedings of the 12th IEEE International Conference on Mobile Data Management (MDM '11)*, vol. 2, pp. 3–6, June 2011.

[5] B. Guo, Z. Yu, D. Zhang, H. He, J. Tian, and X. Zhou, "Toward a group-aware smartphone sensing system," *IEEE Pervasive Computing*, vol. 13, no. 4, pp. 80–88, 2014.

[6] W. Z. Khan, Y. Xiang, M. Y. Aalsalem, and Q. Arshad, "Mobile phone sensing systems: a survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 1, pp. 402–427, 2013.

[7] K. E. Jeon, J. She, P. Soonsawad, and P. C. Ng, "BLE Beacons for Internet of Things Applications: Survey, Challenges and Opportunities," *IEEE Internet of Things Journal*, vol. 5, no. 2, pp. 811–828, 2018.

[8] A. Depari, C. M. Cominics, A. Flammini et al., "Integration of Bluetooth Hands Free Sensors into a Wireless Body Area Network Based on Smartphone," *Sensors*, vol. 162, pp. 547–551, 2013.

[9] C. Pereira, J. Rodrigues, A. Pinto et al., "Smartphones as M2M gateways in smart cities IoT applications," in *Proceedings of the 2016 23rd International Conference on Telecommunications (ICT)*, pp. 1–7, Thessaloniki, Greece, May 2016.

[10] J. Cai, X. Huang, J. Zhang et al., "A Handshake Protocol With Unbalanced Cost for Wireless Updating," *IEEE Access*, vol. 6, pp. 18570–18581, 2018.

[11] B. Zhang, C. H. Liu, J. Lu et al., "Privacy-preserving QoI-aware participant coordination for mobile crowdsourcing," *Computer Networks*, vol. 101, pp. 29–41, 2016.

[12] ISO/IEEE 11073 Personal Health Data, http://www.11073.org/.

[13] Health Level Seven, http://www.hl7.org/.

[14] K. S. Nikita, *Biosensor Communication Technology and Standards*, Wiley-IEEE Press, 2014.

[15] A. M. Nia, M. Mozaffari-Kermani, S. Sur-Kolay, A. Raghunathan, and N. K. Jha, "Energy-Efficient Long-term Continuous Personal Health Monitoring," *IEEE Transactions on Multi-Scale Computing Systems*, vol. 1, no. 2, pp. 85–98, 2015.

[16] Antidote: IEEE 11073 stack, http://oss.signove.com/index.php/Antidote:_IEEE_11073-20601_Library.

[17] J. G. Pak and K. H. Park, "Advanced Pulse Oximetry System for Remote Monitoring and Management," *Journal of Biomedicine and Biotechnology*, vol. 2012, Article ID 930582, 8 pages, 2012.

[18] N. B. Thakkar, "Google Android: An Emerging Software Platform for Mobile Device," *International Journal of Environmental Science and Technology*, vol. 1, pp. 272–278, 2014.

[19] R. N. Horspool, "Practical fast searching in strings," *Software: Practice and Experience*, vol. 10, no. 6, pp. 501–506, 1980.

[20] P. D. Michailidis and K. G. Margaritis, "On-line string matching algorithms: survey and experimental results," *International Journal of Computer Mathematics*, vol. 76, no. 4, pp. 411–434, 2001.

[21] J. Fountain and S. P. Smith, "Automatic identification of rigidly linked 6DoF sensors," in *Proceedings of the 18th IEEE Virtual Reality Conference, VR 2016*, pp. 175-176, USA, March 2016.

[22] Y. Mirsky, B. Shapira, L. Rokach, and Y. Elovici, "pcStream: A stream clustering algorithm for dynamically detecting and managing temporal contexts," *Lecture Notes in Computer Science: Preface*, vol. 9078, pp. 119–133, 2015.

[23] C. Perera, P. P. Jayaraman, A. Zaslavsky, D. Georgakopoulos, and P. Christen, "Sensor Discovery and Configuration Framework for The Internet of Things Paradigm," in *Proceedings of IEEE World Forum on Internet of Things*, pp. 94–99, 2014.

[24] C. H. Liu, B. Yang, and T. Liu, "Efficient naming, addressing and profile services in Internet-of-Things sensory environments," *Ad Hoc Networks*, vol. 18, pp. 85–101, 2014.

[25] C. H. Liu, J. Zhao, H. Zhang, S. Guo, K. K. Leung, and J. Crowcroft, "Energy-Efficient Event Detection by Participatory Sensing under Budget Constraints," *IEEE Systems Journal*, vol. 11, no. 4, pp. 2490–2501, 2017.

[26] J. Frisby, V. Smith, S. Traub, and V. L. Patel, "Contextual Computing: A Bluetooth based approach for tracking healthcare providers in the emergency room," *Journal of Biomedical Informatics*, vol. 65, pp. 97–104, 2017.

[27] C. C. Aggarwal, *Managing and Mining Sensor Data*, Springer, 2013.