

Article

A Formal Approach to the Selection by Minimum Error and Pattern Method for Sensor Data Loss Reduction in Unstable Wireless Sensor Network Communications

Changhwa Kim * and DongHyun Shin

Department of Computer Science and Engineering, Gangneung-Wonju National University, 150, Namwonro, Heungeop-myeon, Wonju-si, Gangwon-do 26403, Korea; dhshin@cs.gwnu.ac.kr

* Correspondence: kch@gwnu.ac.kr; Tel.: +82-33-760-8663

Academic Editor: Leonhard M. Reindl

Received: 23 February 2017; Accepted: 6 May 2017; Published: 12 May 2017

Abstract: There are wireless networks in which typically communications are unsafe. Most terrestrial wireless sensor networks belong to this category of networks. Another example of an unsafe communication network is an underwater acoustic sensor network (UWASN). In UWASNs in particular, communication failures occur frequently and the failure durations can range from seconds up to a few hours, days, or even weeks. These communication failures can cause data losses significant enough to seriously damage human life or property, depending on their application areas. In this paper, we propose a framework to reduce sensor data loss during communication failures and we present a formal approach to the Selection by Minimum Error and Pattern (SMEP) method that plays the most important role for the reduction in sensor data loss under the proposed framework. The SMEP method is compared with other methods to validate its effectiveness through experiments using real-field sensor data sets. Moreover, based on our experimental results and performance comparisons, the SMEP method has been validated to be better than others in terms of the average sensor data value error rate caused by sensor data loss.

Keywords: sensor data loss reduction; wireless sensor networks; underwater acoustic sensor networks; unstable communications; SMEP method; average sensor value error rate

1. Introduction

There are wireless networks in which communications are typically unsafe. Most wireless sensor networks belong to this category of networks. Usually, they have several shortcomings caused by limited resources, such as short-range and low-speed communications, limited-power batteries, low memory capacity, and low-speed processors [1–5]. In particular, their communications are vulnerable to environmental changes, such as foggy, rainy, hot, cold, or dry weather conditions, or location, among other factors [6–12]. Another example of an unsafe communication network is an underwater acoustic sensor network (UWASN). In underwater acoustic communication, there are several barriers that need to be overcome: low bandwidth, high propagation delay, high error rate, multi-path propagation, strong signal attenuation, and time-variant channels [13–18]. The time variation of channels in particular is one of primary causes of unsafe underwater communications. Underwater environmental changes, such as temperature, salinity, pressure, underwater fluid, bubbles, and noise, among others, are the main causes of time variations of channels [13–18], but many other factors also influence channel time variations. Owing to the time-variant channels, communication failures occur frequently and the failure time durations may range from seconds up to a few hours, days, or even weeks. After the failure has ended, in most cases the communication resumes, but during the communication

failure time certain problems may occur. One of these is the loss of sensor data sampled during the failure. Although sensor data can be saved temporally in routers in the form of packets during the course of routing, most or all of them are typically lost. Because the routing queue is not designed for long-time data saving, but rather for short-time saving to facilitate fast routing service, its capacity is insufficient to include many packets. Moreover, even sensor data packets saved in the router will be lost as the saving time increases and the number of new packets for routing increases more and more. On the other hand, no routing can be so much as attempted in a case in which the sensor node does not communicate with any neighbor node. Nonetheless, there are many cases in which it is very important for sensor nodes to save sensor data sampled during communication failure and to resume transmitting them when the failure is recovered. The saved sensor data can also be read out after the sensor node is retrieved by its operators. By doing this, we can detect several facts or forecast future trends such as following, using the sensor data saved during communication failures:

1. *Detection of accidents or events*: What accidents or events occurred?
2. *Main agents of accidents or events*: What or who brought about the accidents or events?
3. *Cause, time, or place of accidents or events*: Why, when, or where did the accidents or events occur?
4. *Progress of accidents or events*: How did the accidents or events proceed?
5. *Analysis of the cause of accidents or events*: What or who was the cause of accidents or events?
6. *Analysis of effects of accidents or events*: What or whom did the accidents or events have effects on and what were these effects?
7. *Prediction of future trends*: What will be changed by accidents or events, and how, when, where, or why will these changes occur?

For longer communication failure durations, in many applications it is more important for us to know some or all of the above. However, as the failure duration is longer, networks or nodes have higher probabilities of sensor data loss in the course of transmission [19,20]. Accordingly, long-period communication failures cause a large amount of sensor data loss on sensor networks and such sensor data loss may cause disadvantages for human beings. For example, long-term sensor data loss in an ocean-area contamination disaster impedes not only the cause and origin identification of the contamination but also the analysis of its dissemination progress and pollution area. Successively, such blockings prevent humankind from the prediction and prevention of the future disasters and their movements [19–22]. Ultimately, the sensor data lossless preservation for long-term communication failure in the previous disaster is certainly helpful for us to predict and prevent the same or similar future disasters, even though we fail in the real-time monitoring and controlling for the current disaster.

Insufficient memory capacity of a sensor node, however, makes it impossible to save the entire sensor data set lost during a long-term communication failure. Therefore, it is necessary to minimize the loss of the original sensor data during communication failure and to restore the lost sensor data nearly to the level of the original sensor data after the communication failure recovery. In other words, we need a method to reduce the sensor data loss caused by the memory capacity limit of the sensor node during communication failure. To date, only a few such methods, including *winavg* [23], *delta* [23], *CQP* [24], and *2MC* [19,20] have been proposed. These methods use compression to replace two or more consecutive sensor data with an average value as a representative or to select a representative between two or three consecutive sensor data. All the sensor data apart from the representatives become lost in these methods. The remaining representatives are transmitted to the destination site, where the lost sensor data are recovered to these representatives (in *winavg*) or to the values interpolated using them. Most of these methods, however, do not show good performance in the average error rate as a measure of the data loss because these use their own very simple rules without consideration for performance measures such as that average error rate. The *2MC* method shows much better performance than *winavg*, *delta*, and *CQP* methods in the sensor data loss minimization, because this method uses an error rate as a performance measure in selecting representatives. However, since the *2MC* method can fail to find the absolute positions of the lost sensor data, it has fatal drawback of being unable

to recover the lost sensor data. Moreover, even though a communication framework for using these methods in wireless sensor networks is necessary, no framework for this has been presented.

Against this background, in this paper, we propose a framework to reduce sensor data loss during communication failures and we present a formal approach for the Selection by Minimum Error and Pattern (SMEP) method that plays the most important role in the reduction of sensor data loss in the proposed framework. The SMEP method uses a line interpolation error measure in selecting representatives to minimize errors in recovering lost sensor data. Compared with other methods using real-field sensor data sets, the experimental results show that the SMEP method has a much better average error rate within up to 87.5% sensor data loss (that is, 12.5% compression ratio). The SMEP method is also feasible, in that it can recover all lost sensor data.

The remainder of this paper is organized as follows. In Section 2, we introduce related works on technologies for sensor data compression, including *winavg* [23], *delta* [23], *CQP* [24], and *2MC* [19,20], which are mostly applicable for sensor data loss reduction in sensor networks. Section 3 proposes a communication framework for the reduction of sensor data loss during communication failures. The proposed SMEP method can be applied to this framework, playing an important role. In Section 4, the fundamental formal definitions and properties of sensor data, compression interval, and round compression are introduced. These are necessary to introduce the SMEP method. Before presenting a detailed description of the SMEP method, the basic formal concepts and properties of the SMEP method are introduced in Section 5. Based on these, Section 6 formally presents the main data structures and examines their properties. These data structures and properties are used not only for compression in the SMEP method algorithm, but also for decompression and recovery in the destination node. Section 7 presents the SMEP algorithms necessary to compress the original sensor data during communication failures, which use the data structures and properties introduced in previous sections. The decompression and lost sensor data recovery from compressed sensor data by the SMEP algorithms is introduced in Section 8. For decompressing and recovering, the main algorithm in Section 8 uses the compressed sensor data and compression information in the data structures introduced in Section 6. A simple line interpolation algorithm is applied to the recovery of the lost sensor data between two consecutive compressed sensor data. In Section 9, we show the performance results and analysis in comparison with the four aforementioned sensor data compression methods, *winavg*, *delta*, *CQP*, and *2MC*, and we conclude this paper in Section 10.

2. Related Work

Many methods for sensor data reduction in wireless sensor networks have been proposed, mainly in three categories: data aggregation, data approximation, and query-based processing. The data aggregation methods cause intermediate or energy-intensive sensor nodes to merge into one packet several sensor data in packets transmitted in the course of routing [25–27]. Consequently, by reducing the number of transmissions, they have the effect of increasing energy efficiency and prolonging network life in wireless sensor networks [25–27].

The data approximation methods make sensor nodes transmit reduced sensor data, instead of transmitting all the original sensor data [28–33]. As a result, these methods also reduce the number of sensor data transmissions, similarly to data aggregation methods. Mostly, these methods use correlation or regression models between two or more different types of sensor data originated from different sensors. They transmit the representatives instead of all sensor data, but the untransmitted sensor data are restored to the approximate values using these correlation or regression models. The data approximation methods have overheads in that they inevitably need a priori knowledge, such as correlation or regression models, depending on the relationships among the sensor data [28–33].

Query-based processing is an on-demand sensor data transmission method. In this category, when each sensor node receives queries from monitoring and control stations, it transmits only the resulting sensor data corresponding to each query as its responses. Thereby, this method reduces the number of sensor data transmissions, avoiding constant transmission [34–38]. Additionally,

query aggregation and results merging methods have been presented for avoiding redundancy in the response results [37,38].

Almost all the mechanisms in the above categories focus only on the energy efficiency resulting from reducing transmissions. Specifically, most of them are based on the premise that transmissions are in real-time and stable in communications. Therefore, in the cases of the absence of their recovery methods in these mechanisms, the sensor data sampled during a communication failure period are lost and the lost sensor data cannot be recovered [20–22,24].

A few mechanisms to recover the lost sensor data during transmission and routing have been proposed [39,40]. These mechanisms, however, require a priori knowledge obtained through preprocessing normal sensor data for detecting the locations of the lost sensor data and restoring them to their approximate values. Since these methods are based on real-time transmission, they too have the same problems described in the above three categories.

For the data loss reduction in unstable networks, delay/disruption tolerant network (DTN) techniques have been proposed [41–44]. These DTN techniques are asymmetric networking technologies that can be used in network environments where stable communication connections are not guaranteed. Generally, if sensor data were lost owing to network problems, it has been required to re-route from the source to the destination for its retransmission. Instead of doing this, these techniques preserve the transmission-intermittent sensor data packets in routers or intermediate nodes until these packets can be normally transmitted to the next node. In these methods, however, there high probabilities that, at the time of communication failure, bunches of sensor data packet streams gather at the specific router or intermediate node in a short period of time. Therefore, the routing queue capacity rapidly reaches its full state, and, thereafter, all the packets of sampled sensor data will be lost and not recovered, because of the absence of any compression and recovery method. Moreover, because packets with extra information such as headers or footers, and not bare data, are saved in routing queue space, the DTN techniques have high probabilities of more rapidly reaching the queue-full state, causing more data loss due to the inefficient use of memory space.

One of the scenarios for avoiding or resolving the problems discussed above is described as follows. Each sensor node is aware of communication failure. Thereafter, it saves its sampled sensor data to its own memory space without depending on other nodes like routers or intermediate nodes, and compresses them into the same memory whenever its memory space researches full state, until the communication failure is recovered. As soon as the sensor node is aware of communication failure recovery, the node starts transmitting the saved original or compressed sensor data to the destination and then, after receiving them, the destination decompresses and recovers the original and lost sensor data. This is the scenario that this paper is pursuing.

A framework relevant to the above scenario has been proposed in our previous work [21]. The conceptualized overview version on this framework is presented shortly in Section 3 of this paper. Meanwhile, the compression and recovery mechanisms capable of following our scenario have been presented in a small number of works in the literature. The *winavg* and *delta* methods are two of them and they have been introduced in [23]. Firstly, the *winavg* method [23] calculates an average value for a given number of sensor data (that is, window size) among sensor data in the queue-full state and saves this average value as the representative value of the sensor data in the window size. This method recovers all the lost sensor data within each window to the same unified values as the average value of the window. Accordingly, the bigger the window size and the standard deviation of the sensor data within each window, the larger the error size between the original sensor data and its recovered data is. The *delta* method, which is also very briefly introduced in [23], selects from two sensor data adjacent to the last selected sensor data, the one of which the value difference from the last selected sensor data is bigger than the other. Thus, the unselected sensor data between these two sensor data is discarded. The *delta* method recovers the sensor data lost between these two consecutive sensor data to the selected one. These two methods have advantages to restore the lost sensor data to the approximated values instead of losing all of them after the data queue becomes full. However, they tend to increase

the error, radically, if queue overflows by communication failures last for a long time or if sensor data values change irregularly with high deviation. The Circular Queue based on Period Compression (CQP) method [24] is also a simple method, and one of our previous works, which is capable of following our scenario to reduce sensor data loss. The CQP method is similar to but simpler than the *delta* method. The CQP method selects unconditionally the first sensor data at any time of the queue overflow. Then, it selects unconditionally the second sensor data between two sensor data located just next to the last selected sensor data in the sensor data sequence in the queue, using a circular queue. In spite of the simplest method, this method shows similar or better performance in the recovery of the lost sensor data compared with the *winavg* and *delta* methods. The 2MC method, another one among our previous works, has been presented in [19,20]. In the compression process, this method divides a sensor data sequence into three-point (sensor data) intervals and the last sensor data of each interval is shared with the first sensor data of its next interval. For each odd-numbered compression interval, it selects either a minimum-valued sensor data, if the sensor data shows a decreasing trend in the interval, or a maximum-valued sensor data, if they show an increasing trend. For each even-numbered compression interval, it selects the sensor data with minimum value difference from the sensor data value selected in the neighboring odd-numbered compression interval. Compared with the *winavg*, *delta*, and CQP methods, this method shows the best performance in the average error rate [19,20]. Recently, however, our team has found out its serious problems, that it is possible to select two sensor data for a compression interval and, in that case, the positions of all the sensor data after these sensor data are miscalculated. For this reason, the 2MC method is infeasible in the decompression and recovery of the compressed and lost sensor data. In fact, the main reason for such an infeasibility is that the 2MC method is just an idea-level method and lacks a mathematical basis. The 2MC method also selects, after the n th compression, only the 2^n th sensor data among the 2^n next original sensor data and discards $2^n - 1$ sensor data. Therefore, there has remained in the 2MC method room for performance improvement by saving all the new original sensor data sampled after each compression, so that they cannot be lost until the next compression. The main reason why this paper proposes the SMEP method is to resolve these problems through a mathematical approach and show its feasibility.

3. Communication Framework Overview for Sensor Data Loss Reduction

In this section, we present an overall framework to reduce sensor data loss in the course of communication failure. In fact, this framework provides for the requirement analysts or designers a guideline for the identification and realization of functionalities necessary to support the reduction of sensor data loss during communication failures in wireless sensor networks.

Our overall framework has five sequential modes over a sensor node, as shown in Figure 1, which are the *Normal Operation (NO)*, *Communication Failure Propagation (CFP)*, *Sensor Data Save and Compression (SDSC)*, *Communication Recovery Propagation (CRP)*, and *Compressed Data Transmission and Recovery (CDTR)* modes. The detailed explanation of the overall framework is as follows:

In the *NO* mode, each sensor node in a sensor network performs its operations normally, such as sampling, processing, sending, or receiving sensor data. There occurs no communication failure in this mode. When a communication failure occurs, the sensor network mode moves from the *NO* mode to the *CFP* mode. In this mode, the first sensor node to detect a communication failure propagates to its neighbor nodes a Communication Failure (CF) message to notify them of the communication failure occurrence. Every sensor node to receive a CF message tries to find an alternative routing path, other than via the sender node of the CF message. If the sensor node finds a new routing path, it invalidates the routing path via the sender node of the CF message and performs transmission operations through the new routing path. If any new routing path is not found, then the sensor node propagates to its neighbor nodes the Communication Failure (CF) message and its mode moves into *SDSC* mode. In this mode, the sensor node starts saving sensor data generated from its sensor into its data queue. The sensor node performs the compression operation over some or all of the sensor data in the queue when the data queue is full. In this mode, as soon as the sensor node receives from

the previous sensor node of the CF message a Communication Recovery (CR) message to notify that the previous communication failure has been recovered, the sensor node mode moves from the *SDSC* mode into the *CRP* mode. In *CRP* mode, the sensor node validates the invalidated routing path via the sender, propagates the CR message to its neighbor nodes again and then its mode moves to the *CDTR* mode. In the *CDTR* mode, the sensor node reverts to using the validated or recovered routing path from the previous mode as a new primary routing path. In this mode, the sensor node sends all the saved or compressed sensor data in data queue and the compression information through the recovered routing path. After that, the sensor node mode returns to the *NO* mode and proceeds with normal operations. The destination node to receive all the compressed sensor data and the compression information proceeds with restoring all the lost sensor data from the compression in *SDSC* mode to the approximated values, which are similar to their original sensor data.

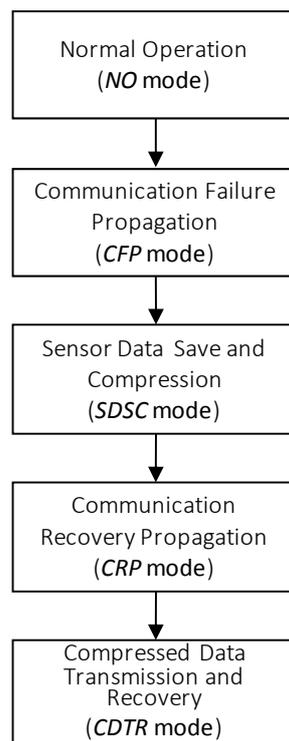


Figure 1. A communication framework for sensor data loss reduction.

There occurs sensor data loss due to the compression operations in *SDSC* mode. In fact, sensor data loss is inevitable in the communication failures with no alternative routing path. Specifically, in the long-term communication failures, without taking any action after the full state of the data or the routing queue, almost all of the new generated and routed sensor data will be lost inevitably. Therefore, lost sensor data cannot be restored to the same values as the original sensor data, or even to similar values. The only solution to resolve this problem is to devise a compression method to compress sensor data in a full queue state and restore the lost ones in the compression process to the same or similar values. For this reason, the compression in the *SDSC* mode is the most important task in our framework to reduce sensor data loss and the *SMEP* method presented from Section 4 to Section 8 is one of such compression methods.

4. Basic Definitions and Properties

Fundamental formal definitions and properties are presented in this section. These are necessary to introduce the *SMEP* method in Sections 5–7, and the decompression and recovery algorithms in Section 8.

Definition 1. The sensor data sequence S is a list of periodic sensor data, S_0, S_1, \dots, S_{n-1} such that for each i ($0 \leq i \leq n - 1$) S_i is a sensor data and if $j < k$ ($0 \leq j, k \leq n - 1$), then S_j is a sensor data generated before the sensor data S_k . We represent the sensor data sequence as $S = S_0, S_1, \dots, S_{n-1}$.

We call the number of sensor data in the sensor data sequence S the size of S or the length of S and represent it with $|S|$.

For simplicity, let us define terms, ground sequence and base sequence.

Definition 2. The ground sequence or ground sensor data sequence is defined as a sensor data sequence of which each sensor data does not pass through any compression after its generation from a sensor.

Definition 3. The base sequence is defined as a sensor data sequence of which each sensor data is selected through the same compressions and is a target for another compression.

Definition 4. Given a sensor data sequence, $S = S_0, S_1, \dots, S_{n-1}$, a compression interval I with the size m with respect to S is defined as a part of S , i.e., $I = S_i, S_{i+1}, \dots, S_{i+m}$, which consists of $m + 1$ consecutive sensor data in S and is a target for compression.

For simplicity, we represent a compression interval I with size m , $S_i, S_{i+1}, \dots, S_{i+m}$ as $[i, i + m]$ by using only the first and last subscripts of its $m + 1$ consecutive sensor data. We can know easily that if $I = [i, j]$ is a compression interval, then the size of I is $j - i$. The compression interval is a sensor data sequence, too. However, note that the size of the compression interval I is different from the size of the sensor data sequence I . In fact, the size of the compression interval I is 1 less than the size of a sensor data sequence I . A compression on the compression interval $I = [i, i + m]$ is an operation to select one or more sensor data from all sensor data in I and discard all other sensor data except the selected ones.

Definition 5. Consider two compression intervals $I = [i, j]$ and $J = [k, l]$ with respect to a sensor data sequence S . Then, if $j = k$ or $l = i$, I and J are defined as consecutive compression intervals in S . If each size of two consecutive compression intervals is the same as m then we call that as consecutive compression intervals with size m .

According to this definition, if two compression intervals are consecutive then the last sensor data of one of two compression intervals is the same as the first sensor data of the other.

Theorem 1. Let $I = [i, j]$ and $J = [k, l]$ be consecutive compression intervals with size m and n , respectively, in a sensor data sequence S . If $i < k$ or $i < l$ or $j < l$, then $J = [i, l]$ is a compression interval with the size $m+n$ with respect to S (For the proof refer to Appendix A).

Given two consecutive compression intervals, $I = [i, j]$ and $J = [k, l]$, of a sensor data sequence, if $i < k$ or $j < l$ (this is, $j = k$), then we represent it with $I < J$ and we call that I is smaller than J or that J is greater than I .

Definition 6. Suppose that $I = [i, j]$ and $J = [k, l]$ are consecutive compression intervals with respect to a sensor data sequence, S , and $I < J$. We define the merging operation \bullet of I and J as $I \bullet J = [i, l]$ and we call $I \bullet J$ the merged compression interval of I and J .

We can prove easily the following theorem, an association law of the merging operation.

Theorem 2. Let I, J , and K be consecutive compression intervals and $I < J < K$. Then, the merging operation satisfies the associative law. That is, $(I \bullet J) \bullet K = I \bullet (J \bullet K)$.

Definition 7. Consider consecutive compression intervals, I_0, I_1, \dots, I_{m-1} with respect to a sensor data sequence, S , where $I_0 < I_1 < I_2 < \dots < I_{m-1}$. If $S = I_0 \bullet I_1 \bullet I_2 \bullet \dots \bullet I_{m-1}$, then a set $C_S = (I_0, I_1, I_2, \dots,$

I_{m-1}) is called as a compression interval covering on S . When the size of every compression interval in the compression interval covering C_S is the same as k , we call C_S the k -size compression interval covering.

From definitions, we can get the relationship between a sensor data sequence and its 2-size compression interval covering as the below. For the proof refer to Appendix B.

Theorem 3. Let S and $C_S = (I_0, I_1, I_2, \dots, I_{m-1})$ be a sensor sequence and a 2-size compression interval covering on S , respectively. Then, the following propositions are true:

Proposition 1. $|S| = 2m + 1$.

Proposition 2. Given a compression interval, $I_i = S_p, S_q, S_r$, for some integer $i, 0 \leq i \leq m - 1$, positions of S_p, S_q , and S_r , is $2i, 2i + 1$, and $2i + 2$, respectively, in S .

In the SDSC mode already introduced in Section 3, in order for a sensor node not to loss sensor data generated periodically from its sensor, the sensor node has to have its own data queue to save lost sensor data in the SDSC mode. However, whenever the data queue is full in the SDSC mode, the 2MC method executes a compression on a sensor data sequence of its data queue. Now, let us define a *compression round* as follows:

Definition 8. A round compression is an operation defined as follows and a compression round is the number of the compression executions:

- (i) A ground sequence is in the 0-th round compression.
- (ii) The i -th round compression is a compression executed when S is a sensor data sequence corresponding to a full data queue and each sensor data in S have been derived through the i -1 th compression.

At the time when a compression is executed, some of sensor data sequence are selected and the others are discarded. Here, the selected become the compressed sensor data but the discarded becomes the lost sensor data. In order for each of lost sensor data to be restored, its position should be known or found. The SMEP method uses position information about the selected sensor data to find positions the sensor data lost in compression.

Besides the above definitions more definitions are introduced in the following sections whenever necessary. All of main notations and terms are listed in Table 1 with their definition numbers. Refer to definitions if necessary.

Table 1. Notations and terms.

Notations	Terms	Definitions
S	sensor data sequence	
S_i	i -th sensor data of S	Definition 1
-	ground sequence (ground data sequence)	Definition 2
-	base sequence	Definition 3
$I = S_i, S_{i+1}, \dots, S_{i+m} = [i, i + 1 \dots, i + m]$	compression interval	Definition 4
-	consecutive compression intervals	Definition 5
$I \bullet J$	merged compression interval of I and J	Definition 6
$C_S = (I_0, I_1, I_2, \dots, I_{m-1})$	compression interval covering on S	Definition 7
I_i	i -th compression interval in C_S	
-	round compression	Definition 8
S^c	sensor data sequence with the c -th round compression	-
$S_{c,i}$	i -th sensor data in S^c	-
$I_{c,i} (= [p, q]_c = [2i, 2i + 1]_c)$	i -th 2-size compression interval in C_S^c	-
-	covers	Definition 9, Definition 10
I -bit position($S_{c,i}$)	I -bit position of $S_{c,i}$	Definition 11
B_p	I -bit position sequence of S	Definition 12

Table 1. Cont.

Notations	Terms	Definitions
$BitSubstring(B_p, c, k)$	bit substring from the $c(k - 1)$ th bit to $ck-1$ th bit in B_p	Definition 13
-	absolute position of $S_{c,k}$	Definition 14
-	selection by compression interval pattern with respect to $I_{c,i}$	Definition 15
-	selection error of a sensor data	Definition 16
E_p	line interpolation errors of S_p	Definition 17
-	selection by minimum error rule	Definition 18
$Zones[0..m - 1]$	Zones with respect to the data queue $DataQueue[0..2^m]$	Definition 19
-	operations on $Zone[0..m-1]$	Definition 20
$ZoneBpSeq[0..m - 1]$	zone bit sequence array	Definition 21
E_{M,S^c}	average error rate (AER) of S^c with respect to S^0 in the compression method M	Definition 22

5. Basic Concepts for the SMEP Method

The SMEP method uses two selection rules for two consecutive compression intervals to cover a sensor data sequence for the compression; *selection by compression interval pattern* and *selection by minimum error*. This section presents basic SMEP method concepts helpful to understand its data structure elements and algorithms introduced in Sections 6–8 in detail, including two selection rules.

5.1. SMEP Compression Process Overview

The SMEP proceeds the compression with 2-size consecutive compression interval covering with respect to a sensor data sequence with the size $2^m + 1$ for in integer m , $m \geq 1$. Actually, the size $2^m + 1$ or 2^m is the same as the length of a part of data queue used for saving and compressing the sensor data generated from a sensor in the SDSC mode and the sensor data sequence for the compression is a sequence of sensor data in the queue. Depending on the $2^m + 1$ size, we get the below theorem (For the proof refer to Appendix C):

Theorem 4. Let S and C_s be a sensor data sequence S and a 2-size compression interval covering of S and $C_s = (I_0, I_1, \dots, I_l)$, respectively. If the size of S is $2^m + 1$ for an integer $m \geq 1$, then $l = 2^{m-1} - 1$, that is, the number of compression intervals of C_s is 2^{m-1} .

Whenever a compression round is proceeded, one sensor data in each compression interval is selected and the others are discarded in it. At this time, the discarded sensor data becomes the lost sensor data. Figures 2 and 3 show the compression process on a sensor data sequence in a data queue with the size $17 = 2^4 + 1$. As shown in Figure 2, in the c -th ($c = 1, 2, 3, \dots$) round compression, sensor data after the $c-1$ th round compression are selected as follows:

At first, the first sensor data $S_{c-1,0}$ is selected as $S_{c,0}$, unconditionally, where $S_{c,i}$ means the i -th sensor data of a sensor data sequence derived through the c -th round compression. Then, by selecting one sensor data per every compression interval, according to the order between compression intervals in the 2-size compression interval covering, the compression derives a new compressed sensor data sequence. One of both of the 2nd and the 3rd sensor data in each interval is selected so that the SMEP method should keep a selection rule, *one per compression interval*, for all compression intervals except the 1st compression interval. In selection, the i -th sensor data selected in the previous compression round becomes the i -th sensor data in the new compressed sensor data sequence. After the compression, the size of the new sensor data sequence shrinks to the half the previous one, this is, $2^{m-1} + 1$, occupying data queue space with the same size, and the $2^{m-1} + 1$ size queue space remains empty. After that, as sensor data generated newly from a sensor are inserted into the remaining queue space, the space becomes full like Figures 2 and 3 so that new sensor data should be compressed with the same way until all sensor data in the full state data queue are derived through c time compressions.

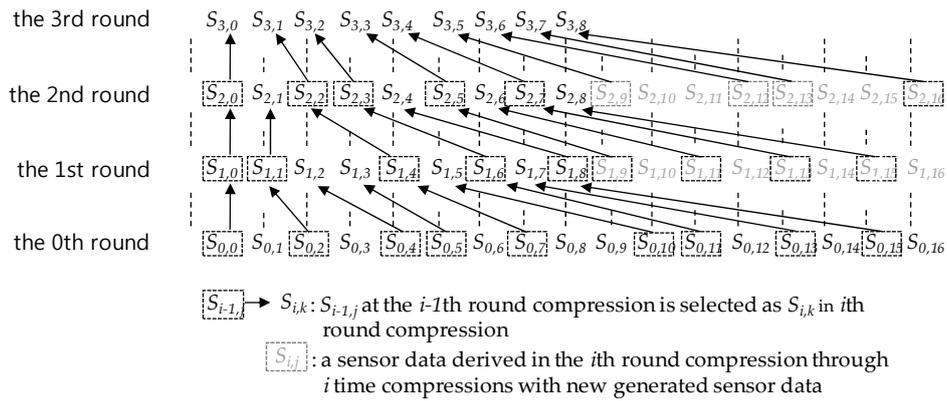


Figure 2. Sensor data selection in the 1st, 2nd and 3rd round compressions.

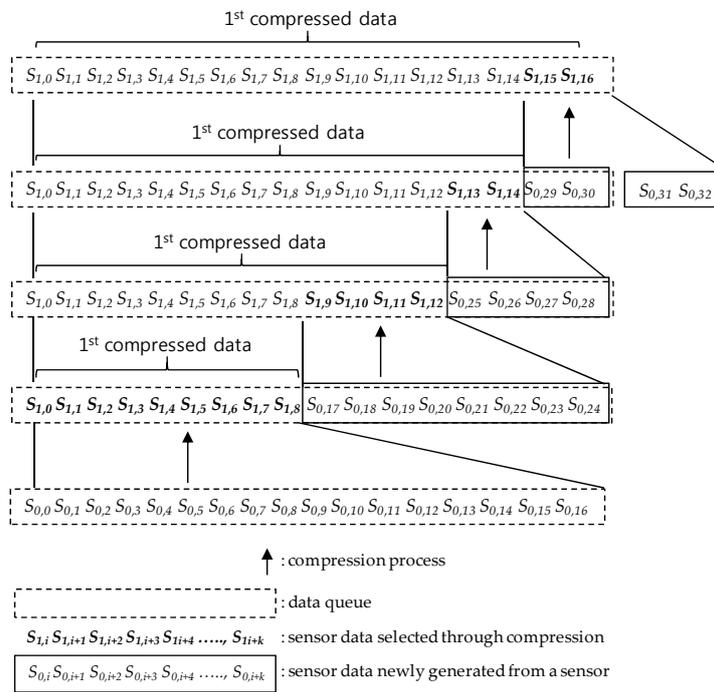


Figure 3. The first round compression process.

When all sensor data of a sensor data sequence in the full state data queue are derived through c time compressions, the c -th round compression has been completed and the new $c+1$ th round compression begins. Figure 3 shows only the first round compression process.

5.2. I-bit Position and Generation

Figure 4 shows the compression process in the aspect of the merging of two intervals. Compression is conducted by selecting just one from sensor data in a compression interval that corresponds to the merging of two sequent compression intervals of the base sequence. For example, in Figure 4, $S_{1,3}$, a sensor data, is selected as the one from a compression interval, $I_{0,2} = [4, 6]_0$. Here, $I_{c,i}$ means the i -th position compression interval in the 2-size compression interval covering on a sensor data sequence derived though the c -th round compression. Additionally, $[p, q]_c$ represents an compression interval, $[p, q]$ on a sequence derived through the c -th round compression. Shortly, $(I_{0,2}, S_{1,3})$ in the figure means the sensor data $S_{1,3}$ is selected in the compression interval $I_{0,2}$.

For further description, we define the term *cover*:

Definition 9. Let S_i and I be a sensor data and a compression interval with respect to a sensor data sequence, respectively. S_i covers I if and only if S_i is selected in I .

In Figure 4, $S_{2,2}$ covers $I_{1,1}$ and it is selected one of $S_{1,3}$ and $S_{1,4}$ to cover $I_{0,2}$ and $I_{0,3}$, respectively. Note that $S_{2,2}$ covers $I_{0,2} \bullet I_{0,3}$ in this figure. We define the term *cover* on the view of compression interval:

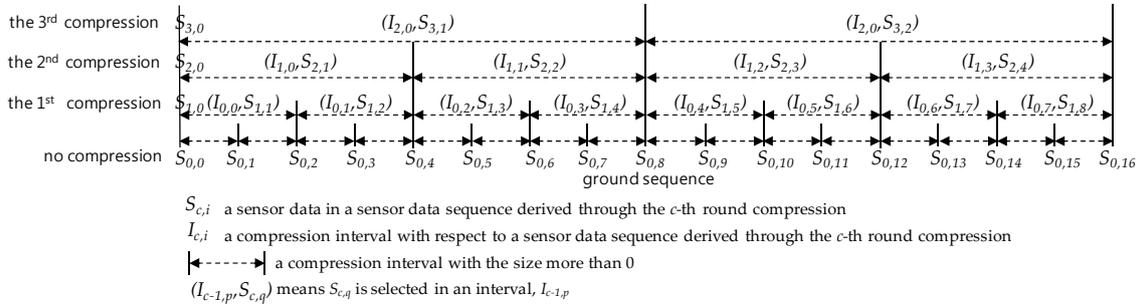


Figure 4. A sensor data selection in the merging of a pair of two compression intervals.

Definition 10. Given two compression intervals $I_{c-2,j}$ and $I_{c-2,j+1}$ and a selected sensor data $S_{c,k}$ to cover $I_{c-1,k}$ for $c \geq 2$, $I_{c-1,k}$ covers $I_{c-2,j} \bullet I_{c-2,j+1}$ if and only if $S_{c,k}$ covers $I_{c-2,j} \bullet I_{c-2,j+1}$.

According to the definition, $I_{1,1}$ covers $I_{0,2} \bullet I_{0,3}$ in Figure 4. Moreover, $I_{2,0}$ covers $I_{1,0} \bullet I_{1,1}$ and $I_{1,0}$ and $I_{1,1}$ also cover $I_{0,0} \bullet I_{0,1}$ and $I_{0,2} \bullet I_{0,3}$. Since $S_{3,1}$ covers $I_{2,0}$ and $S_{3,1}$ covers $I_{0,0} \bullet I_{0,1} \bullet I_{0,2} \bullet I_{0,3}$ $I_{2,0}$, $I_{2,0}$ also covers $I_{0,0} \bullet I_{0,1} \bullet I_{0,2} \bullet I_{0,3}$ $I_{2,0}$. For the generalization of this property, we introduce the below Lemma 1, Lemma 2 and Theorem 5. For their proofs refer to Appendix H, Appendix I, and Appendix D, respectively.

Lemma 1. Let S and $S^c = S_{c,0}, S_{c,1}, \dots, S_{c,2^m}$ for integer $m \geq 0$ be a ground sequence and a sensor data sequence derived through the c -th round compression on S , respectively. Let $C_s^c = (I_{c,0}, I_{c,1}, \dots, I_{c,2^{m-1}-1})$ be a 2-size compression interval covering. Then, in the SMEP method, if for $k \geq 0$, $I_{c,k}$ covers $I_{c-1,2k} \bullet I_{c-1,2k+1}$.

Lemma 2. Let S and $S^c = S_{c,0}, S_{c,1}, \dots, S_{c,2^m}$ for integer $m \geq 0$ be a ground sequence and a sensor data sequence derived through the c -th round compression on S , respectively. Let $C_s^c = (I_{c,0}, I_{c,1}, \dots, I_{c,2^{m-1}-1})$ be a 2-size compression interval covering on S^c . Then, in the SMEP method, if for $i \geq 0$ and $j \geq 1$ $S_{c,i}$ covers $I_{c-1,j}$ then $j = i - 1$ and $I_{c-1,j}$ covers $I_{c-2,2(i-1)} \bullet I_{c-2,2i-1}$.

Theorem 5. Let S and $S^c = S_{c,0}, S_{c,1}, \dots, S_{c,2^m}$ for integer $m \geq 0$ be a ground sequence and a sensor data sequence derived through the c -th round compression, respectively, based on S . Then, in the SMEP method, for integers $c \geq 1$ and for any integer $i \geq 1$, $S_{c,i}$ covers $I_{0,(i-1)2^{c-1}} \bullet I_{0,(i-1)2^{c-1}+1} \bullet \dots \bullet I_{0,i \cdot 2^{c-1}-1}$.

This theorem tells $S_{c,i}$ is the sensor data selected in $I_{0,(i-1)2^{c-1}} \bullet I_{0,(i-1)2^{c-1}+1} \bullet \dots \bullet I_{0,i \cdot 2^{c-1}-1}$. By the way, since $I_{0,k} = [2k, 2k + 2]_0$ and $I_{0,(i-1)2^{c-1}} \bullet I_{0,(i-1)2^{c-1}+1} \bullet \dots \bullet I_{0,i \cdot 2^{c-1}-1} = [2(I-1)2^{c-1}, 2(i-1)2^{c-1} + 2]_0 \bullet [2(i-1)2^{c-1} + 2, 2(i-1)2^{c-1} + 4]_0 \bullet \dots \bullet [i2^c - 2, i2^c]_0 = [(i-1)2^c, i2^c]_0$, $S_{c,i}$ is the sensor data selected in the compression interval $[(i-1)2^c, i2^c]_0$.

Corollary. Let $S_{c,i}$ be the i -th position sensor data in a sensor data sequence derived through the i -th round compression. Then, the position of $S_{c,i}$ exists in $[(I-1)2^c, i2^c]_0$.

As described already before, the positions of the selected data should be necessarily known in order that each of all lost sensor data is restored. The very function that lets the SMEP method know where each sensor data is either selected or lost is I-bit position.

Definition 11. *I-bit position is defined recursively as a function to generate the bit string about the position of $S_{c,i}$ in a compression interval according to the following rules:*

Rule 1. For any $c = 1, 2, \dots, n$, I-bit position($S_{c,0}$) is not defined.

Rule 2. In 1-th round compression, for each $I_{0,k}$ in C_{S_0} for $k = 0, 1, \dots, |C_{S_0}| - 1$ and for $S_{1,i}$ selected from $I_{0,i-1} = [2i - 2, 2i]_0 = S_{0,2i-2}, S_{0,2i-1}, S_{0,2i}$,

$$I\text{-bit position}(S_{1,i}) = \begin{cases} 0 & \text{if } S_{0,2i-1} \text{ is selected as } S_{1,i} \\ 1 & \text{if } S_{0,2i} \text{ is selected as } S_{1,i} \end{cases}$$

Rule 3. In c-th round compression, for each $I_{c-1,k}$ in $C_{S_{c-1}}$ for $k = 0, 1, \dots, |C_{S_{c-1}}| - 1$ and for $S_{c,i}$ selected from $I_{c-1,i-1} = [2i - 2, 2i]_{c-1} = S_{c-1,2i-2}, S_{c-1,2i-1}, S_{c-1,2i}$,

$$I\text{-bit position}(S_{c,i}) = \begin{cases} 0 \oplus I\text{-bit position}(S_{c-1,2i-1}) & \text{if } S_{c-1,2i-1} \text{ is selected as } S_{c,i} \\ 1 \oplus I\text{-bit position}(S_{c-1,2i-1}) & \text{if } S_{c-1,2i} \text{ is selected as } S_{c,i} \end{cases}$$

Shortly, we call I-bit position of $S_{c,i}$ -the value of *I-bit position*($S_{c,i}$). The I-bit position represents the relative position of a sensor data within a compression interval. In fact, given an *I-bit position*($S_{c,i}$), the real relative position of $S_{c,i}$ in its interval is (the value of *I-bit position*($S_{c,i}$)) + 1, As compression makes progress, two consecutive compression intervals that two competitive sensor data for selection cover is merged into one compression interval covered by the selected sensor data. The above definition lets us know how the position of a selected sensor data in a merged compression interval is determined by positions of two competitive sensor data. Figure 5 illustrates such a process to generate the I-bit position of a selected sensor data in compression. In the 1st round compression, $S_{0,0}, S_{0,2}, S_{0,4}, S_{0,5}$, and $S_{0,7}$ are selected as $S_{1,0}, S_{1,1}, S_{1,2}, S_{1,3}$, and $S_{1,4}$ from compression intervals, $I_{0,0}, I_{0,1}, I_{0,2}$, and $I_{0,3}$, respectively. Here, note that $S_{0,0}$ is unconditionally selected as $S_{1,0}$ and its I-bit position is undefined according the definition *Rule 1*, and $S_{0,2}$ is selected as $S_{1,1}$ in the compression interval $I_{0,0}$. According to the *Rule 2* of the above definition, their corresponding I-bit positions are *null* (i.e., undefined), 1, 1, 0, and 0 in the 1st round compression. As in the 2nd round compression, $I_{0,0}$ and $I_{0,1}$, are merged into $I_{1,0}$ and $I_{0,2}$ and $I_{0,3}$ are also merged into $I_{1,1}$, $S_{2,0}, S_{2,1}$ and $S_{2,2}$ are selected from $S_{1,0}, S_{1,1}$, and $S_{1,4}$, respectively. Each of them is selected as one per each compression interval after the 1st compression, except the unconditionally selected $S_{2,0}$ in $I_{1,0}$. $S_{1,0}$ and $S_{1,1}$ are positioned in the first compression interval $I_{1,0}$ and $S_{1,4}$ is positioned in the second one $I_{1,1}$. According to the definition *Rule 1*, I-bit positions of $S_{2,0}$ is undefined as *null*, again. In addition, by applying the first condition of the definition *Rule 3* to the first compression interval, I-bit position of $S_{2,1}$ are created by adding 0 bit to the front of the *I-bit position* of $S_{1,1}$ and, ultimately, they become to be 01 as a relative position bit string for the merged compression interval $I_{2,0}$. Meanwhile, $S_{2,2}$ selected from $S_{1,4}$ exists in the second compression interval $I_{1,1}$ among two consecutive compression intervals covered by $I_{2,0}$. In this case, we have to apply the second condition of the *Rule 3* and so we can get the bit string 10 by adding the bit 1 to the front of I-bit position of $S_{1,4}$. After that, $S_{3,0}$ and $S_{3,1}$ are selected from $S_{2,0}$ and $S_{2,2}$ located in the first and second compression intervals $I_{1,0}$ and $I_{1,1}$ covered by $I_{2,0}$. Therefore, we can apply the *Rule 1* and the condition of the *Rule 3* to the generation of I-bit positions of $S_{3,0}$ and $S_{3,1}$ so to get *null* and 110, respectively.

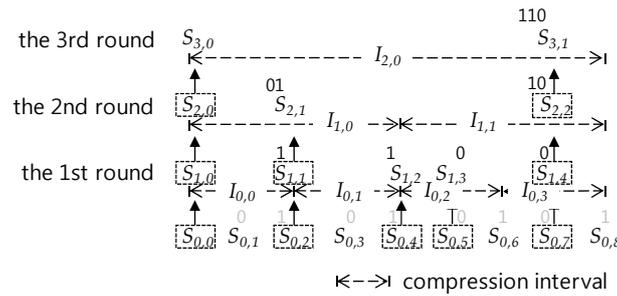


Figure 5. I-bit position generation process of sensor data in compression interval.

As a new round compression proceeds, the length of I-bit position of a selected sensor data makes one more increase in the new round compression than in the previous round compression, resulting to the c length bit string for the c -th compression, ultimately.

5.3. Absolute Position

As soon as a sensor data is selected during a compression, the SMEP method generates an I-bit position corresponding to the sensor data and updates *I-bit position sequence* defined in the below, appending its I-bit position to it:

Definition 12. Let S^0 and $S^c = S_{c,0}, S_{c,1}, \dots, S_{c,n}$ be a ground sequence and a sensor data sequence compressed from S^0 through the c -th round compression. I-bit position sequence of S^c is defined as a bit string sequence that consists of concatenations of I-bit positions of all sensor data except $S_{c,0}$ in S^c . That is,

$$I\text{-bit position sequence of } S^c = I\text{-bit position}(S_{c,1}) \oplus I\text{-bit position}(S_{c,2}) \oplus \dots \oplus I\text{-bit position}(S_{c,n}).$$

Moreover, the position of the first bit of the I-bit position sequence is defined as 0. From the above definition, note that the I-bit position corresponding to the first sensor data is excluded from I-bit position sequence. In fact, we can remove it because the position of the first sensor data is always fixed as the first for every compression. In Figure 5, I-bit position sequence of S^1 is 1100 (= $1 \oplus 1 \oplus 0 \oplus 0$) generated by concatenating I-bit positions of $S_{1,1}, S_{1,2}, S_{1,3}$, and $S_{1,4}$ except $S_{1,0}$. I-bit position sequences of S^1 , and S^2 are 0110 (= $01 \oplus 10$), and 110 by the same way, respectively.

Given an I-bit position sequence, B_p , of S^c , for any sensor data, $S_{c,k}$, we can get easily the *I-bit position* of $S_{c,k}$ by taking a substring from the $c(k - 1)$ th position bit to $ck - 1$ th position bit in B_p . So, let us define a function, *BitSubstring*, now:

Definition 13. Given an I-bit position sequence, B_p , of S^c , for any sensor data, $S_{c,k}$, *BitSubstring* is a function to take a bit substring from the $c(k - 1)$ th bit to $ck - 1$ th bit in B_p , this is, $BitSubstring(B_p, c, k) =$ a bit substring from the $c(k - 1)$ th bit to $ck - 1$ th bit in B_p .

Lemma 3. Given an I-bit position sequence, B_p , of S^c , for any sensor data, $S_{c,k}$,

$I\text{-bit position}(S_{c,k}) = BitSubstring(B_p, c, k)$ (For the proof refer to Appendix J). On the other hand, all data in S^c originate from S^0 . In fact, they are sensor data that compete to survive through selections in c time compressions. In order to restore the lost data in competences, for each sensor data in the compressed data sequence its position in the ground sequence, which we call the position the absolute position defined the below, has to be known:

Definition 14. Let $S_{c,k}$ be a sensor data that originates from $S_{0,i}$. Then, the absolute position of $S_{c,k}$ is defined as the position i of $S_{0,i}$ in S^0 .

The absolute position of a sensor data after c compressions can be known with its I-bit position by the following Lemma (For the proof refer to Appendix K):

Lemma 4. Let S^0 and $S^c = S_{c,0}, S_{c,1}, \dots, S_{c,n}$ be a ground sequence and a sensor data sequence derived from S^0 through the c -th round compression, respectively. Given B_p , an I-bit position sequence, for any $S_{c,k}$ originated from $S_{0,i}$ in S_0 , the absolute position i is computed as follows:

- (i) $i = 0$, if $k = 0$
- (ii) $i = (k-1)2^c + (I\text{-bit position}(S_{c,k}))_{10} + 1$, if $k \geq 1$

By the two above lemmas, the following theorem is trivially true:

Theorem 6. Given B_p , an I-bit position sequence of S^c , for any $S_{c,k}$ ($k \geq 1$), the absolute position of $S_{c,k}$ $i = (k-1)2^c + (\text{BitSubstring}(B_p, c, k))_{10} + 1$, if $k \geq 1$

The above theorem is important in that it means that we can find the absolute position of a compressed sensor data if we know the I-bit position sequence, the number of compressions and the position of the compressed sensor data in the compressed sensor data sequence.

5.4. Selection by Compression Interval Pattern Rule

For a compression interval, $I_c = [i, I + 2]_c = S_{c,i}, S_{c,i+1}, S_{c,i+2}$ with size 2, we can have one of nine cases depending on these sensor data values as follows:

- Case 1. (Pattern 1) $S_{c,i} > S_{c,i+1} > S_{c,i+2}$
- Case 2. (Pattern 2) $S_{c,i} < S_{c,i+1} < S_{c,i+2}$
- Case 3. (Pattern 3) $S_{c,i} = S_{c,i+1} = S_{c,i+2}$
- Case 4. (Pattern 4) $S_{c,i} > S_{c,i+1} < S_{c,i+2}$
- Case 5. (Pattern 5) $S_{c,i} < S_{c,i+1} > S_{c,i+2}$
- Case 6. (Pattern 6) $S_{c,i} > S_{c,i+1} = S_{c,i+2}$
- Case 7. (Pattern 7) $S_{c,i} < S_{c,i+1} = S_{c,i+2}$
- Case 8. (Pattern 8) $S_{c,i} = S_{c,i+1} > S_{c,i+2}$
- Case 9. (Pattern 9) $S_{c,i} = S_{c,i+1} < S_{c,i+2}$

Each of the above cases shows its own pattern for S_i, S_{i+1}, S_{i+2} as shown in Figure 6.

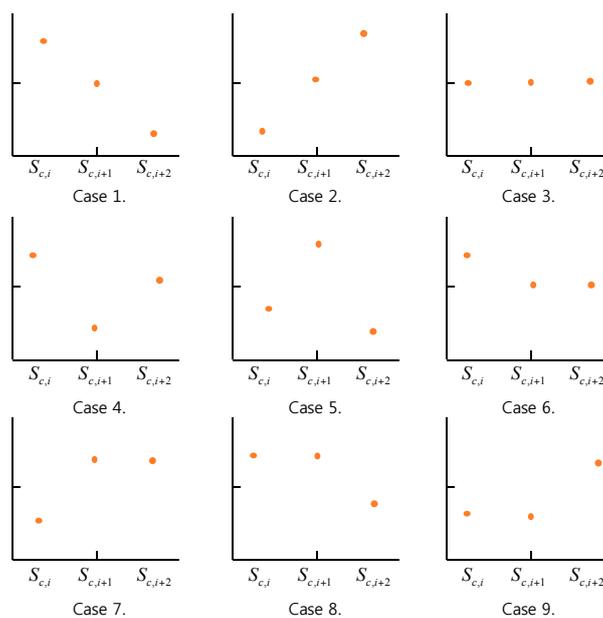


Figure 6. Nine patterns of sensor data in 2-size compression interval.

Assume that S^c is a sensor data sequence in a data queue for compression and $C_{S^c} = (I_{c,1}, I_{c,2}, \dots, I_{c,m})$ is a 2-size compression interval covering with respect to S^c . Then, we define the *selection by compression interval pattern* as follows:

Definition 15. Given a compression interval $I_{c,i} = S_{c,i}, S_{c,i+1}, S_{c,i+2}$ in C_{S^c} with respect to S^c , the *selection by compression interval pattern with respect to $I_{c,i}$* is defined as the selection of a sensor data among $S_{c,i}, S_{c,i+1}, S_{c,i+2}$ according to the following selection by pattern rule:

Selection by Pattern Rule: If Pattern 1 or Pattern 2 or Pattern 3 is true among nine patterns, then $S_{c,i+2}$ is selected, otherwise $S_{c,i+1}$ is selected.

Simply, we will call the *selection by compression interval pattern* rule the *pattern selection* rule or *selection by pattern* rule. The *selection by pattern* rule is derived not from noble theoretical results to minimize the errors between every lost sensor data and its corresponding original sensor data in the several compressions, but from empirical results through several experiments with several sets of real-field sensor data. Thus, the compression to use only the *pattern selection* method tends occasionally to show worse performance than the CQP method and others. For this reason, we have devised a method mixed up with the *pattern selection* method and the *selection by minimum error* method in Section 5.5.

On the other hand, in two consecutive compression intervals of 2-size compression interval covering, the first sensor data in the second interval is the same one with the third sensor data in the first interval according to Definition 5 in Section 4. If the third sensor data is selected in the first interval, the first sensor data cannot be selected in the second interval. For this reason, the *selection by pattern* rule selects only one of the 2nd sensor data ($S_{c,i+1}$) and the 3rd sensor data ($S_{c,i+2}$) except the 1st sensor data ($S_{c,i}$) in the compression interval.

5.5. Selection by Minimum Error Rule

In the SMEP method, given a 2-size compression interval covering and a compression interval for compression, the *selection by minimum error* rule uses two beforehand-selected sensor data from two neighbor compression intervals for sensor data selection. That is, this rule selects the sensor data to minimize error using those two beforehand-selected sensor data. Before the detailed description, we introduce the selection error of a sensor data, for the time being:

Definition 16. The *selection error of a sensor data* is defined as an error incurred by selecting a sensor data among two or more selectable sensor data.

Our purpose in compression is to minimize selection errors so that the lost sensor data, i.e., the unselected sensor data, should be restored to sensor data almost similar to the original sensor data. For our purpose, we need an estimation measure to allow compare their selection error sizes each other among sensor data. For this reason, we introduce a line interpolation error measure as one of estimation measures. Figure 7 shows the line interpolation error measure to be used as a selection error measure and it is defined at the below:

Definition 17. Let be I_{k-1}, I_k and I_{k+1} be compression intervals and let S_i and S_j be the beforehand selected sensor data in I_{k-1} and I_{k+1} , respectively. For selectable consecutive sensor data, S_p and S_q ($p \leq q$), consider two lines, $\overline{S_p S_j}$ and $\overline{S_i S_q}$ calculated using absolute positions of S_i and S_j . And consider two points, $a = (q', A)$ and $b = (p', B)$ on $\overline{S_p S_j}$ and $\overline{S_i S_q}$, respectively, where p' and q' are absolute position of p and q . Then, line interpolation errors, E_p and E_q , of S_p and S_q are defined as follows:

$$E_p = |S_q - A| \text{ and } E_q = |S_p - B|.$$

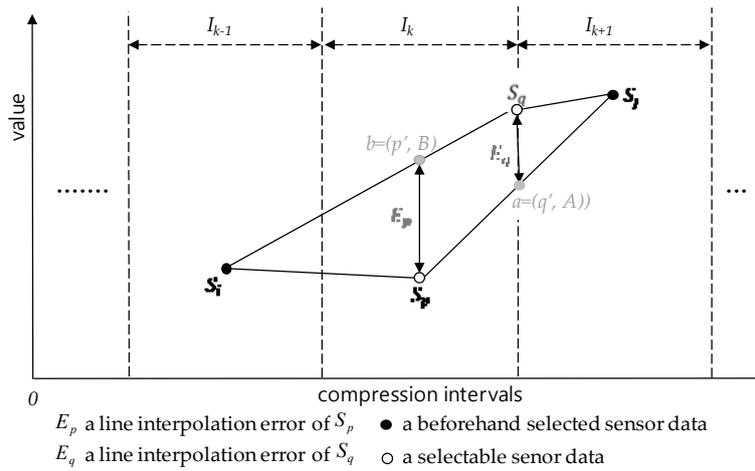


Figure 7. Line interpolation error measure.

From now on, we will regard the selection error as the line interpolation error. In the definition, the absolute position for each of all sensor data is used for calculating line interpolation errors. We have a valid reason why we have to use the absolute positions of sensor data in calculating the line interpolation errors: Consider three or more consecutive sensor data to take one or more compressions. They seem likely to have the same distance unit 1 each other at glance. Generally, however, there can be big differences in their absolute positions. For example, consider in Figure 2 three sensor data, $S_{2,0}$, $S_{2,1}$ and $S_{2,2}$, where they are consecutive each other with the same 1 unit distance in the 2nd round compressed sensor data sequence. Their corresponding original sensor data, however, are $S_{0,0}$, $S_{0,2}$ and $S_{0,7}$ and the distance units between them are different with 2 and 5 between $S_{0,0}$ and $S_{0,2}$ and between $S_{0,2}$ and $S_{0,7}$, respectively.

Exceptionally, the SMEP uses $S_{c,0}$ for the predetermined sensor data $S_{c,i}$ in the definition so to select one of $S_{c,1}$ and $S_{c,2}$ for $I_{c,0}$. The reason to do so is because there exists no interval for predetermined sensor data in its left-hand side but if any interval can exit the unique sensor data is the very $S_{c,0}$ that can be the left boundary sharable sensor data of $I_{c,0}$.

The SMEP executes the *selection by minimum error* rule for each I_{2k} and the *selection by pattern* rule for I_{2k+1} in C_{S_c} , for $k=0, 1, \dots, |C_{S_c}|$. Therefore, in the definition, the real compression intervals for the SMEP method to apply the *selection by minimum error* rule to the compression are I_{2k} for $k = 0, 1, \dots, |C_{S_c}|$. Now, focusing on I_{2k} ($k = 0, 1, \dots, |C_{S_c}|$), we investigate some properties for minimizing the selection error.

Lemma 5. Let be $I_{c,2k-1}$, $I_{c,2k}$ and $I_{c,2k+2}$ be consecutive compression intervals in C_{S_c} of S^c and let $S_{c,i}$ and $S_{c,j}$ be the beforehand selected sensor data in I_{2k-1} and I_{2k+2} , respectively. Given B_p , an I -bit position sequence of S^c , for i', j', p' and q' in the line interpolation error definition,

$$\begin{aligned} i' &= (i - 1)2^c + (\text{BitSubstring}(B_p, c, i))_{10}, \\ j' &= (j - 1)2^c + (\text{BitSubstring}(B_p, c, j))_{10}, \\ p' &= (p - 1)2^c + (\text{BitSubstring}(B_p, c, p))_{10}, \\ q' &= (q - 1)2^c + (\text{BitSubstring}(B_p, c, q))_{10}. \end{aligned}$$

Moreover, for A and B in the definition,

$$\begin{aligned} A &= \frac{S_{c,j} - S_{c,4k+1}}{j' - p'} (q' - p') + S_{c,4k+1} \\ B &= \frac{S_{c,i} - S_{c,4k+2}}{i' - q'} (p' - q') + S_{c,4k+2} \end{aligned}$$

For the proof of Lemma 5 refer to Appendix L.

Theorem 7. For selectable S_p and S_q in I_{2k} in the *line interpolation error* definition (Definition 17),

$$E_p = E_{4k+1} = \left| \frac{S_{c,j} - S_{c,4k+1}}{j' - p'} (q' - p') - (S_{c,4k+2} - S_{c,4k+1}) \right|$$

and

$$E_q = E_{4k+2} = \left| \frac{S_{c,i} - S_{c,4k+2}}{i' - q'} (q' - p') - (S_{c,4k+2} - S_{c,4k+1}) \right|$$

For the proof of Theorem 7 refer to Appendix E.

Until now, in this section we have used the terminology *selection by minimum error* rule without its exact formal definition. Now, the definition of this rule is formally defined as the below:

Definition 18. Given the two selectable sensor data, $S_{c,4k+1}$ and $S_{c,4k+2}$, in $I_{c,2k}$ in C_{S_c} of S^c , the selection by minimum error rule is defined as a rule to select $S_{c,i}$ as $S_{c+1,k}$ in the $c+1$ th round compression such that:

$$p = \arg \min_{i \in \{4k+1, 4k+2\}} (E_i)$$

6. Main SMEP Elements

SMEP algorithms create and use a few of data structures as main elements for the compression of sensor data sequences. These data structures are *DataQueue*, *Zones*, *ZoneBpSeq*, *ZoneCompHistory*, and *AuxDataQueue*. Moreover, each information in these main elements is transmitted to one or more consumer (i.e., destination) sites in order that they decompress and recover compressed and lost sensor data using this information. This section presents each of these data structures, their operations, and the related properties significant to the SMEP algorithms.

6.1. Data Queue and Zones

DataQueue is an array that saves a ground sensor data and compressed sensor data and the compression is executed on this *DataQueue*. Each compression makes the number of the targeted ground or compressed sensor data in *DataQueue* reduce to the half. For this reason, the size of *DataQueue* must be $2^m + 1$. Furthermore, *Zones* array information corresponding to *DataQueue* ranges is a basis on each compression. *Zones* data structure is defined as the below:

Definition 19. Given a data queue, $DataQueue[0..2^m]$, with the size $2^m + 1$, a *Zones* with respect to the data queue is defined as a data structure with the size m , this is, $Zones[0..m - 1]$, as follow:

- (1) $Zones[0]$ corresponds to the range $DataQueue[k]$ for $k, 0 \leq k \leq 2^{m-1}$.
- (2) $Zones[i]$ for $i = 1, \dots, m - 2$ corresponds to the range $DataQueue[k]$ for $k, \sum_{j=0}^{i-1} 2^{m-j-1} + 1 \leq k \leq \sum_{j=0}^{i-1} 2^{m-j-1} + 2^{m-i-1}$.
- (3) $Zones[m - 1]$ corresponds to the range $DataQueue[k]$ for $k, 2^m - 1 \leq k \leq 2^m$.
- (4) Each value of $Zones[m - 1]$ represents the number of compressions through which all sensor data in the data queue corresponding to the $Zones[m - 1]$ have been compressed.

When a zone corresponds to a range of *DataQueue*, we call *the zone covers the range*. Figure 8 illustrates $Zones[0..3]$ with respect to $DataQueue[0..2^4]$. In this figure $Zones[0]$ covers a range from $S_{3,0}$ to $S_{3,8}$. This range is relevant to one plus a half of *DataQueue* size as an area to save the selected sensor data when *DataQueue* is full of ground or compressed sensor data. In Figure 8, all sensor data in this

range have been selected through three compressions. *Zones*[1] covers a range from $S_{2,0}$ (*DataQueue*[9]) to $S_{2,4}$ (*DataQueue*[12]) taken through two compressions and the number of sensor data in this range is 2^2 . *Zones*[2] and *Zones*[3] cover a range between $S_{2,5}$ (*DataQueue*[13]) and $S_{2,6}$ (*DataQueue*[14]) and between a range of $S_{0,0}$ (*DataQueue*[15]) and $S_{0,1}$ (*DataQueue*[16]), respectively. In particular, the range that *Zones*[3] covers does not path through any compression in Figure 8.

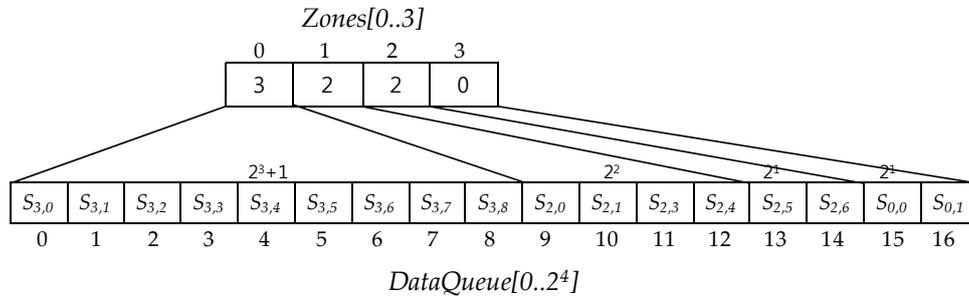


Figure 8. Zones example with respect to *DataQueue*[0..2⁴].

Now, we introduce operations on the *Zones* data structure.

Definition 20. The operations on *Zones*[0..m – 1] = [c₀, c₁, . . . , c_{m–1}] are defined as follows:

Rule 1. At first, ground sensor data are saved into *DataQueue* starting from *DataQueue*[0], this is, the first location of the *DataQueue* region corresponding to *Zones*[0], and the values of *Zones* are

$$[0, 0, \dots, 0]$$

Rule 2. Given *Zones* = [c₀, c₁, . . . , c_{m–2}, c_{m–1}], if c_{m–1} = c_{m–2} after a compression at the last zone *Zones*[m – 1], then

- (i) the c_{m–2}th compression on *DataQueue* is executed on the zones from c_k such that k = min{i | c_i = c_{m–1}} to c_{m–1},
- (ii) the result of *Zones* after the i) compression is *Zones* = [c₀, c₁, . . . , c_k + 1, 0, . . . , 0].
- (iii) Additionally, new ground sensor data are saved into *DataQueue* starting from *DataQueue*[$\sum_{j=0}^{j=k} 2^{m-j-1} + 1$], this is, the first location of *DataQueue* covered by *Zones*[c_{k+1}].

Figure 9 shows the changes of *Zones* values in compressions and round compressions according to the *Zones* operation rules. The arrow in the figure represent the compression executed when *DataQueue* is full. Note the round compression consists of one or more compressions and the round compression is finished when all values in *Zones* are same. At the beginning of the *SDSC* mode *Zones* values are 0,0,0,0 according to *Zones* operation Rule 1 and they are also the same values until *DataQueue* is full firstly. As soon as *DataQueue* is filled up, the first compression proceeds for whole sensor data in it, the first compressed sensor data are saved to the range of *DataQueue* covered by *Zone*[0] and the value of *Zones*[0] is 1, which means the *DataQueue* sensor data in *Zones*[0] have been compressed once. The number of compressed sensor data is the same as the size of the range covered by *Zones*[0]. After the compression, *Zones*[1], *Zones*[2] and *Zones*[3] values become all 0, which means sensor data in these zone ranges don't take any compression so to be able to save the new generated sensor data and to compress them. When ranges covered by these 0 value zones are full again, the second compression makes the second zone *Zones*[1] value be 1, the number of compressions, and it makes *Zones*[2] and *Zone*[3] values be all zero. When values of *Zones*[0..3] are all 1s, the 1st round compression is completed and the 2nd round compression begins. Generally, if all values of *Zones*[0..m – 1] are same with c in the c-th round compression is completed and the c + 1 th round

compression begins. In this case, the value of $Zones[0]$ is one up so to be $c + 1$, the other $Zones$ values are all 0s, and another new compressions begin whenever $DataQueue$ space covered by 0 value zones are full. These compression processes continues repeatedly until all $Zones$ values are same with $c + 1$.

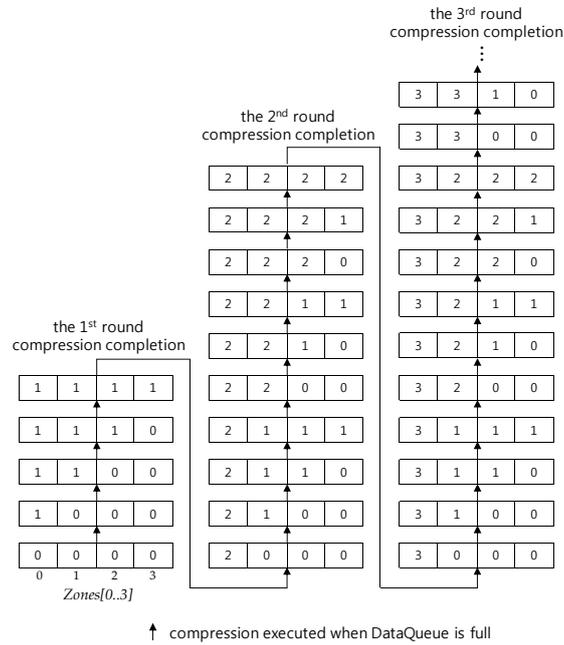


Figure 9. Changes of $Zones$ values in compressions and round compressions.

6.2. Zone Bit Sequence

Now, we introduce $ZoneBpSeq[0..m - 1]$ that is an array of references to I -bit position sequences of $Zones$. Numbers of compressions in zones of $DataQueue$ generally differ from each other and the length of I -bit position of sensor data in each zone can differ from other zone lengths, too. Therefore, we cannot handle all the sensor data in $DataQueue$ as a sensor data sequence in compressions. Consequently, we have no way but to treat $DataQueue$ sensor data with a sensor data sequence per each zone. I -bit position lengths of sensor data in the same zone are all the same since they have passed through the same number of compressions. For this reason, we use $ZoneBpSeq[0..m - 1]$ as an array of pointers to refer to I -bit position sequences of the sensor data sequences corresponding to each zones of $DataQueue$.

Definition 21. Given a zone of $DataQueue$, its zone bit sequence is defined as the I -bit position sequence corresponding to a sensor data sequence in the zone and we call $ZoneBpSeq[0..m - 1]$ a zone bit sequence array.

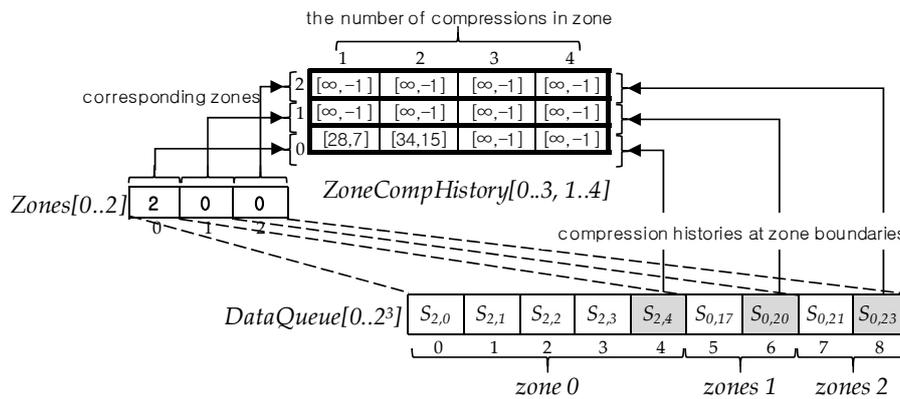
With the arrays $Zones$ and $ZoneBpSeq$ we can seek the absolute position of any sensor data in $DataQueue$ according to the following theorem (For the proof refer to Appendix F).

Theorem 8. Let P_i be the absolute position for the sensor data $DataQueue[i]$ for any i such that $1 \leq I \leq 2^m$. Given a $Zones[0..m - 1]$ and $ZoneBpSeq[0..m - 1]$,

- (i) i exists in the zone $k = \max\{ t \mid 0 < (i - 2^{m-t} \cdot (2^t - 1)) \}$ for $1 \leq I \leq 2^m - 1$ and i exists in the zone $m - 1$ for $i = 2^m$.
- (ii)
$$P_i = \begin{cases} 0 & \text{for } i = 0 \\ \sum_{j=0}^{j=k-1} 2^{m-j-1} \cdot 2^{Zones[j]} + (i - \sum_{j=0}^{j=k-1} 2^{m-j-1} - 1) \cdot 2^{Zones[k]} + \text{for } i \neq 0 \text{ and } k \geq 0 \\ (\text{BitSubstring}(\text{ZoneBpSeq}[k], \text{Zones}[k], i - \sum_{j=0}^{j=k-1} 2^{m-j-1}))_{10} + 1 \end{cases}$$

6.3. Zone Compression History

In every compression, the sensor data $S_{c,0}$ in the zone 0 is selected unconditionally in the compression interval sequence in the zone. In fact, $S_{c,0}$ in the zone 0 plays a role as a base in sensor data selection by the first application of *selection by minimum error* rule in the zone 0. We call such a sensor data like $S_{c,0}$ the *base sensor data* in the zone of *DataQueue*. In other words, a base sensor data is the first sensor data positioned in the first interval in a zone of *DataQueue* and it has a role as a base in applying the first *selection by minimum error* rule in the zone. By the way, a problem is that there is no base sensor data in all zones except the zone 0 so to prevent from selecting a new sensor data from the first interval in the first zone for the compression. To resolve this problem, we make the last selected sensor data in the previous zone be a base for the compression of a sensor data sequence after that zone. In spite of doing that, another problem occurs if there is among zones no history about sensor data selected in every compression. Figure 10 shows such an example. In this figure, the sensor data sequence between the zone 1 and the zone 2 should be compressed and the numbers of compressions in the zone 0 and in the zone 1 are different as 2 and 0, respectively. In this case, however, there is no way for compression without the last selected sensor and absolute position in the zone 0 as a historical sensor data in the 1st compression. Because there is no base sensor data for the 1th compression. For this reason, we introduce *ZoneCompHistory*, an array, as the very data structure to keep the historical information about zone compressions. As shown in Figure 10, the rows in the *ZoneCompHistory* array corresponds to zones of *DataQueue* and its columns correspond to the number of compressions. Moreover, its value is a pair of the value and absolute position of the sensor data selected in the zone through compression.



ZoneCompHistory[0, 2] = [34,15] means the original value and absolute position of a sensor data selected at the boundary of zone 0 in the 2nd compression are 34 and 15, respectively

Figure 10. Historical sensor data in compressions.

As an example, in Figure 10, *ZoneCompHistory*[0,1] tells the sensor data with 28 as its value and 7 as its absolute position had been selected as a last sensor data in the zone 0 through the 1st compression. This sensor data is the very base for the 1st compression of a sensor data sequence between zones 1 and 2.

6.4. Auxiliary Data Queue

As long as the number of compressions in the last zone of *DataQueue* is less than the number of compressions in its previous zone, there occurs no new compression with its neighbor zones even though *DataQueue* is filled up with compressed sensor data. In this situation, *AuxDataQueue*, an auxiliary data queue, is used for new ground sensor data to be inserted into and it is also used to adjust the number of compressions in the last zone until that number is to be equal to the number of

compressions in its previous zone. Using *AuxDataQueue*, the number of the last zone compressions increases one by one until it equals to the number of its previous zone compressions. Figure 11 shows such an example to use an auxiliary queue for a compression on the last zone. Figure 11 also illustrates in turn the main steps and changes of the last zone compression process with numbering.

Given $Zones[m - 1]$, the number of compressions in the last zone, $AuxDataQueue[0..2^{Zones[m-1]+1}]$, an array with the size $2^{Zones[m-1]+1} + 1$, is required to increase the number of the last zone compressions up one more ⁽¹⁾ in Figure 11). Note that in Figure 11 m , $Zones[m - 1]$, and $AuxDataQueue[0, \dots, 2^{Zones[m-1]+1}]$ are 3, 1, and $AuxDataQueue[0..4]$, respectively. Specially, $AuxDataQueue[0]$ is used as the base sensor data assigned from $ZoneCompHistory[m - 1, c]$ s for c between 1 and $Zones[m - 1]$ ⁽²⁾ in Figure 11). A new compression begins when $AuxDataQueue$ becomes full and continues until only one selected sensor data remains in the queue through $Zones[m - 1] + 1$ compressions. Of course, the rules of *selection by pattern* and *selection by minimum error* are applied alternately to auxiliary queue compressions ^(3), 4), 5) in Figure 11). Moreover, the final compression on auxiliary queue is always complete with the *selection by pattern* rule ⁽⁵⁾ in Figure 11). Next, one of two sensor data in the last zone is selected according to the *selection by minimum error* rule, using $ZoneCompHistory[m - 2, Zones[m - 1] + 1]$ and the remaining auxiliary queue sensor data as a base and a sensor data selected through the *selection by pattern* rule, respectively ⁽⁶⁾ in Figure 11). This selected sensor data and the remaining auxiliary queue sensor data are replaced as the values before and at the last position in $DataQueue$, respectively ⁽⁷⁾ in Figure 11). During these steps, *I-bit* positions of each selected sensor data are calculated according to *I-bit position* definition (Definition 11) and they are reflected to find their absolute positions and to make an *I-bit* position sequence of the last zone. Accordingly, $ZoneCompHistory$ is modified with the last selected sensor data value and its absolute position ⁽⁸⁾ in Figure 11). Finally, the last value of $Zones$ is also changed to the same value as the $Zones$ value corresponding to the zone before the last zone when the compression on the two last sensor data has been finished ⁽⁹⁾ in Figure 11).

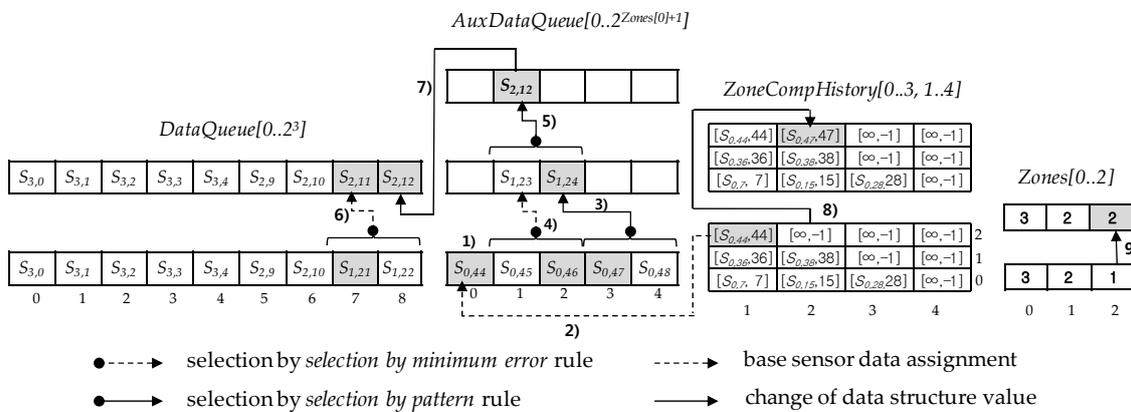


Figure 11. Compression process on last zone.

7. SMEP Algorithms

Compression algorithms to execute the SMEP method use the rules, data structures, operations, and theorems in Sections 5 and 6. For compressions, these algorithms keep and follow *Zones operations definition Rules* in Definition 20 partly or overall. The *AuxDataQueueCompression* algorithm introduced in Section 7.1 as a smallest procedure unit executes compressions on a ground sensor data sequence in *AuxDataQueue*. The *LastZoneCompression* algorithm introduced in Section 7.2 compresses two sensor data in the last zone using *AuxDataQueue Compression*. *EquivalentZonesCompression* algorithm introduced in Section 7.3 performs compression over consecutive zones with the same number of compressions according to *Zones operations definition rule* when $DataQueue$ is full and the number of compressions of the last zone is same as that of its neighbor zone. Following *overall Zones operations definition rules* the SMEP main algorithm introduced in Section 7.4 controls and manages

compressions in various cases using the above algorithms. Now, we introduce these algorithms in turn in this subsections.

7.1. AuxDataQueue Compression Algorithm

The *AuxDataQueueCompression* algorithm is the algorithm for compressing a sensor data sequence in *AuxDataQueue*. This algorithm is illustrated Algorithm 1.

Algorithm 1: *AuxDataQueueCompression(AuxDataQueue, m)*

Parameters

m: the number of zones such that $DataQueue[0..2^m]$

Global Variables

Local Variables

01 *zoneCValue*: the number of compressions of sensor data sequence in the $DataQueue[m - 1]$ zone

02 *c*: the number of compressions for sensor data sequence in *AuxDataQueue*

03 *DQMaxABP*: the greatest absolute position covered by *DataQueue*

04 *ZoneBpSeq0*, *ZoneBpSeq1*: bit string variables for an *I-bit* sequence generation

Procedure

01 Set $zoneCValue = Zones[m - 1]$;

02 Set $DQABP = \sum_{j=0}^{j=m-2} 2^{m-j-1} \cdot 2^{Zones[j]} + 2 \cdot 2^{Zones[m-1]}$

03 Initialize $c = 1$

04 Initialize $ZoneBpSeq0 = \epsilon b$ (an empty bit string)

05 while ($c \leq czoneCValue$) do

06 Set *C* to the 2-size compression interval covering on a sensor data sequence in $AuxDataQueue[0..2^{zoneCValue+1}]$

07 Initialize $ZoneBpSeq1 = \epsilon b$ (an empty bit string)

08 Set $AuxDataQueue[0] = ZoneCompHistory[m - 1, c].Value$

09 for each $I_{c-1,2k}$ in *C* such that $0 \leq k \leq 2^{zoneCValue}$ do

10 Select $S_{c-1,j}$ from $I_{c-1,2k+1}$ by the selection by pattern Rule

11 Select $S_{c-1,p}$ from $I_{c-1,2k+2}$ by the selection by minimum error Rule using

ZoneCompHistory[m - 1, c] as a base sensor data and its absolute position,

this is, $p = \arg \min_{i \in \{4k+1, 4k+2\}} (E_i)$

12 Set $DataQueue[k + 1] = S_{c-1,p}$

13 Set $DataQueue[k + 2] = S_{c-1,j}$

14 Set $ZoneBpSeq1 = ZoneBpSeq1 \oplus I\text{-bit position}(S_{c,k+1}) \oplus I\text{-bit position}(S_{c,k+2})$

by using *ZoneBpSeq0*

15 endfor

16 Set $ZoneBpSeq0 = ZoneBpSeq1$

17 Set $c = c + 1$

18 endwhile

19 Select $S_{c,j}$ from $I_{c,0}$ by the selection by pattern rule

20 Set $AuxDataQueue[1] = S_{c,j}$

21 Set $ZoneBpSeq0 = I\text{-bit position}(S_{c+1,1})$ by using *ZoneBpSeq1*

22 Set *j* = the position of $S_{c,j}$ with respect to *AuxDataQueue* by using *ZoneBpSeq0*

23 return [$AuxDataQueue[1]$, $DQMaxABP+j$, $ZoneBpSeq0$]

Endprocedure

This algorithm increases one more the number of compressions in the last zone of *DataQueue*. The local variable *zoneCValue* is used as the number of compressions in the last zone. The *zoneCValue-time* compression part over *AuxDataQueue* appears from the procedure line 5 to the line 18. Here, note that the *selection by pattern rule* and the *selection by minimum error rule* are applied alternately in lines 10 and 11. The lines 14, 16, 20, and 21 generates ultimately the *I-bit* position of the finally selected sensor data with respect to a ground sensor data sequence in *AuxDataQueue* before *AuxDataQueueCompression* algorithm execution. With this position and the *DQMaxABP* variable in the procedure line 2, the absolute position of the sensor data finally compressed through this algorithm can be obtained as $DQMaxABP+j$, one of return values in the line 23, where the *DQMaxABP* variable value is calculated in the line 2 as the greatest absolute position covered by the *DataQueue*. This

position is very important in that it is recorded into *ZoneCompHistory* with the selected sensor data value and it can be used as the absolute position of a base sensor data in the future compressions. Finally, the algorithm executes the *zoneCValue* th compression in the line 19 applying the selection by pattern rule, selects only one sensor data, and returns its value, absolute position and I-bit position.

7.2. Last Zone Compression Algorithm

The *LastZoneCompression* is a procedure that reads ground sensor data and compresses them using *AuxDataQueueCompression* procedure so to increase one more the number of compressions of the last zone. The *LastZoneCompression* procedure is called by the *SMEP* main procedure introduced in Section 7.4 when the last zone becomes full and the number of compressions in the last zone is less than the number in its previous neighbor zone. The last zone compression algorithm is shown in Algorithm 2.

Algorithm 2: *LastZoneCompression(DataQueue, m)*

Parameters

01 m : the number of zones such that $DataQueue[0..2^m]$

Global Variables

01 $AuxDataQueue[0..2^{Zones[m-1]+1}]$: an auxiliary data queue for compressing a sensor sequence in the last zone

02 Ap : a variable for the absolute position of the latest ground sensor data

Local Variables

01 c : the number of compressions for a sensor data sequence in $Zones[m - 1]$

02 $AuxDQinx$: an index for $AuxDataQueue$

03 $ZoneBpSeq0$: a temporal bit string variable for I-bit sequence

Procedure

01 Set $c = Zones[m - 1]$

02 Initialize $ZoneBpSeq0 = \epsilon b$ (an empty bit string)

04 Allocate memory $AuxDataQueue[0..2^{c+1}]$

05 Initialize $AuxDQinx = 1$

06 while($AuxDQinx \leq 2^{c+1}$) do

07 Set $Ap = Ap + 1$

08 Read a ground sensor data $S_{0,Ap}$ from a sensor

09 Insert $S_{0,Ap}$ into $AuxDataQueue[AuxDQinx]$

10 if ($receive(CRmessage) = true$)

11 then return 'FinishMode'

12 else Set $AuxDQinx = AuxDQinx + 1$

13 Continue

14 endif

15 endwhile

16 Set $[Value, j, ZoneBpSeq0] = AuxDataQueueCompression(AuxDataQueue, m)$

17 Select $S_{c,p}$ from the $2^{m-1} - 1$ positioned interval of $DataQueue$ by the selection by minimum error rule by using $ZoneCompHistory[m - 2, c + 1]$ as a base sensor data and its absolute position and by using $[Value, j]$ as the value and absolute position of the sensor data selected in the next interval in the $c+1$ th compression,

this is, Set $p = \arg \min_{i \in \{2^{m-1}, 2^m\}} (E_i)$

18 Set $DataQueue[2^m - 1] = S_{c,p}$

19 Set $ZoneBpSeq0 = I\text{-bit position}(DataQueue[2^m - 1]) \ominus ZoneBpSeq0$ by using $ZoneBpSeq[m - 1]$

20 Set $DataQueue[2^m] = Value$

21 Set $ZoneBpSeq[m - 1] = ZoneBpSeq0$

22 Set $ZoneCompHistory[m - 1, c+1] = [Value, j]$

23 return 'Continue'

Endprocedure

The *LastZoneCompression* procedure allocates the *AuxDataQueue* array space in order to make the last zone be taken to the one more compression (procedure line 4). Hence, the size of *AuxDataQueue* array space must be $2^{Zones[m-1]+1}$. Next, this procedure begins to read ground sensor data and inserts

them into *AuxDataQueue* (from procedure line 6 to line 15). Whenever the procedure reads a ground sensor data, it checks if the communication failure has been recovered using *Receive(CRmessage)* function (procedure line 10). If the check is true, the procedure returns to the calling procedure, the SMEP main procedure, and informs the recovery from the communication failure to that procedure by returning 'FinishMode' (procedure line 11). After filling up *AuxDataQueue* the procedure calls the *AuxDataQueueCompression* procedure to compress a ground sensor data in *AuxDataQueue* (procedure line 16). The *AuxDataQueueCompression* procedure returns a finally selected sensor data value, its absolute position, and its zone bit sequence (procedure line 16). The procedure uses the returned value and absolute position to select one between two last sensor data by the *selection by minimum error* rule (procedure line 17). The finally selected sensor data and the returned value are inserted into *DataQueue[2m - 2]* and *DataQueue[2m - 1]*, respectively, as the results of the last zone compression (procedure line 18 and line 20). The I-bit position of the finally selected sensor data and the returned bit sequence are used for a last zone bit sequence (procedure line 19 and line 21). Then, the new last sensor data value and its absolute position in the last zone are newly recorded into *ZoneCompHistory[m - 1, c + 1]* for the next compressions (procedure line 22), where *c* is the number of the last compression in the last zone before calling this procedure.

7.3. Consecutive Equivalent Zones Compression Algorithm

Let us say that the zones are *equivalent* when zones are same in numbers of compressions. *EquivalentZonesCompression* is a procedure that compresses the consecutive zones equivalent with the last zone, and that keeps *Rule 2 (i) and (ii)* in the *Zones operations* definition (Definition 20). This procedure also returns to the SMEP main procedure the value and absolute position of the last selected sensor data, where this sensor data is the last of the beginning zone among consecutive equivalent zones after compressing their sensor data. The SMEP main procedure calls this procedure to pass the beginning zone among consecutive equivalent zones to the parameter *StartZone*, as shown in the Algorithm 3.

DQGP is calculated as the greatest position (index) among *DataQueue* positions (indices) covered by zones before the *StartZone* zone. This position is used to calculate *DataQueue* positions of sensor data in *DataQueue* from the *StartZone* zone to the last $m - 1$ zone (procedure line 2). In fact, a sensor data sequence from the *StartZone* zone to the last zone is the same as the sensor data sequence from *DataQueue[DQGP + 1]* to *DataQueue[2^m]*. At this time, if *StartZone* is not zero, the *ZoneCompHistory[StartZone, c].Value* is used as a base sensor data and *ZoneCompHistory[StartZone, c].Position* is used as its absolute position (procedure line 4). The compression proceeds over the 2-size compression interval covering on this sensor data sequence (from procedure line 4 to line 11). In this compression, the procedure applies alternately the *selection by pattern* rule and the *selection by minimum error* rule for the odd number intervals and the even number intervals, respectively (procedure line 6 and line 7). Here, *DQGP* is used to calculate the *DataQueue* position to save the sensor data selected for each interval (procedure line 8 and line 9). The sensor data sequence from the *StartZone* zone to the last $m - 1$ zone is compressed to the *Startzone* zone, where this is proved in the last of this section. Additionally, the zone bit sequence about the *StartZone* zone is generated during compression using I-bit positions of $S_{c,p}$ and $S_{c,j}$ and the zone bit strings corresponding to them (procedure line 10 and line 12). The procedure calculates the absolute position of the last selected sensor data in the compression using *Zones[StartZone]* and *ZoneBpSeq[StartZone]* according to the Theorem 8 (procedure line 13), and returns this last sensor data value and absolute position which are to be saved to *ZoneCompHistory[StartZone, Zones[StartZone] + 1]* by the SMEP main procedure.

Finally, we prove the below theorem (For the proof refer to Appendix G):

Theorem 9. *In the compression for the sensor data sequence in consecutive equivalent zones from the StartZone zone to the last $m - 1$ zone in DataQueue, this sensor data sequence is compressed to the Startzone zone.*

Algorithm 3: *EquivalentZonesCompression(DataQueue, StartZone, m)*

Parameters

01 *StartZone*: a compression start zone02 *m*: the number of zones such that *DataQueue*[0..2^{*m*}]03 *c*: the number of compressions for a sensor data sequence in the zone *StartZone*

Global Variables

Local Variables

01 *DQGP*: the greatest position (index) among *DataQueue* positions (indices) covered by zones before the zone *StartZone*. This position is used to calculate *DataQueue* positions from the *StartZone* zone to the last *m* – 1 zone02 *APos*: an absolute position of the last sensor data in the zone *StartZone* after the compression

Procedure

01 Set *c* = *Zones*[*StartZone*]02 if *DQGP* = 0 then Set *DQGP* = 0 else Set $DQGP = \sum_{j=0}^{j=StartZone-1} 2^{m-j-1}$ endif03 Initialize *ZoneBpSeq*0 = ϵ b (an empty bit string)04 Set *C_{DQGP}* = a 2-size compression interval covering with respect to a sensor data sequence from *DataQueue*[*DQGP*+1] to *DataQueue*[2^{*m*}] (if *StartZone* ≠ 0, set *C_{DQGP}* including to this sensor data sequence *ZoneCompHistory*[*StartZone* – 1, *c* + 1].*Value* as a base sensor data and its absolute position as *ZoneCompHistory*[*StartZone* – 1, *c* + 1].*Position*)05 for each *I_{c,2k}* in *C_{DQGP}* such that *k* is an integer and $0 \leq k < 2^{m-StartZone-2}$ do06 Select *S_{c,j}* from *I_{c,2k+1}* by the selection by pattern rule07 Select *S_{c,p}* from *I_{c,2k}* the selection by minimum error Rule,

$$\text{this is, } p = \arg \min_{i \in \{4k+1, 4k+2\}} (E_i)$$

08 Set *DataQueue*[*DQGP*+2*k*+1] = *S_{c,p}*09 Set *DataQueue*[*DQGP*+2*k*+2] = *S_{c,j}*10 Set *ZoneBpSeq*0 = *ZoneBpSeq*0 ⊕ *I*-bit position(*S_{c+1, DQGP}*+2*k*+1) ⊕ *I*-bit position(*S_{c+1, DQGP}*+2*k*+2) using *I*-bit position(*S_{c,p}*), *I*-bit position(*S_{c,j}*) and the zone bit strings corresponding to *p* and *j*

11 endfor

12 Set *ZoneBpSeq*[*StartZone*] = *ZoneBpSeq*013 Set *APos* = an absolute position of *S_{c,j}* calculated by the theorem using *Zones*[*StartZone*] and *ZoneBpSeq*[*StartZone*]14 return [*S_{c,j}*, *APos*]

Endprocedure

7.4. Main Algorithm

The SMEP main procedure proceeds to save and compress the growing ground sensor data sequence controlling *Zones operations Rule 1* and *Rule 2* in the Definition 20 in Section 6.1. This procedure uses the data structures and procedures already introduced in previous sections. This algorithm is illustrated in Algorithm 4.

At first, the produce initializes *Zones* and *ZoneCompHistory* data structures, the absolute position variable *Ap*, the *DataQueue* index *DQinx* and the zone index variable *ZNinx* (from procedure line 1 to line 7). Then, the procedure reads and inserts a ground sensor data into *DataQueue* (procedure line 9 and line 10).

If *DataQueue* is not full but the communication failure is not still recovered (procedure line 12), then the procedure continues to read and insert a new sensor data into *DataQueue* (procedure line 14 and line 15). If the communication failure is recovered (procedure line 12) then the procedure returns all sensor data in *DataQueue*, the *Zones* data, the *ZonesBpSeq* data, the *ZoneCompHistory* data and the absolute position *Ap* of the last ground sensor data to CDTR mode via CRP mode (procedure line 13). The CDTR mode procedure will transmit them to the monitoring and control center (i.e., destination site), in which the original ground data and the lost sensor data are restored, ultimately. How to transmit in the CDTR mode via the CRP mode is beyond our issues.

If *DataQueue* becomes full, then the procedure continues to compress the last zone while the number of the compressions of the last zone is less than that of its previous zone (from procedure line 18 to line 24). For the last zone compression the procedure calls the *LastZoneCompression* procedure. As described before, the *LastZoneCompression* procedure returns 'FinishMode' when it has received CR message in the middle of reading and inserting a new ground sensor data (procedure line 10 and line 11

in Algorithm 4). At this time the procedure returns *DataQueue*, *Zones*, *ZonesBpSeq*, *ZoneCompHistory*, *Ap*, and *AuxDataQueue* essential for the original and lost sensor data to be recovered (procedure line 20, line 21, and line 22). If the procedure doesn't receive any CR message, it increases one more the number of compressions of the last zone since (procedure line 23).

When the last zone and its previous zone become equivalent, according to *Zones operations rule*, the procedure finds the start zone of the consecutive equivalent zones (procedure line 25).

Algorithm 4: Main SMEP (Selection by Minimum Error and Pattern)

Constant

01 *cmax*: the maximum number of compressions

Global Variables

01 *DataQueue*[0.. 2^m]: a data queue array for saving sensor data or compressing a sensor data sequence

02 *Zones*[0.. $m-1$]: an array of zones with respect to *DataQueue*[0.. 2^m], where $\text{Zone}[i] \leq cmax$

03 *AuxDataQueue*[0.. $2^{\text{Zones}[m-1]}$]: an auxiliary data queue for the compressing a sensor sequence in the last $m-1$ zone

04 *ZoneBpSeq*[0.. $m-1$]: an array of pointers to refer to 1-bit position sequences of the sensor data sequences corresponding to each zones

05 *ZoneCompHistory*[0.. $m-1$, 1..*cmax*]: an array of pairs, [S, p]s, where $\text{ZoneCompHistory}[i, c] = [S, p]$ and S and p are the sensor data value and its absolute position of the last c-th compressed sensor data in the zone i

06 *Ap*: a variable for the absolute position of the latest ground sensor data

Local Variables

01 *DQinx*: an index variable for the latest sensor data insertion to *DataQueue*

02 *ZNinx*: an index variable in *Zones* array

Procedure

01 for each *k* such that $0 \leq k \leq m-1$ do

02 Initialize $\text{Zones}[k] = 0$

03 for each *l* such that $0 \leq l \leq cmax$ do

04 Initialize $\text{ZoneCompHistory}[k, l] = [\infty, -1]$

05 endfor

06 endfor

07 Initialize $Ap = 0$, $DQinx = 0$, and $ZNinx = 0$

08 while(true) do

09 Read a ground sensor data $S_{0,Ap}$ from a sensor

10 Insert $S_{0,Ap}$ into $\text{DataQueue}[DQinx]$;

11 if ($DQinx \leq 2^m$)

12 cthen if ($\text{receive}(\text{CRmessage}) = \text{true}$)

13 then return [*DataQueue*, *Zones*, *ZonesBpSeq*, *ZoneCompHistory*, *Ap*]

14 else Set $Ap = Ap + 1$, $DQinx = DQinx + 1$

15 Continue

16 endif

17 else Set $ZNinx = m - 1$

18 while($\text{Zones}[ZNinx-1] < \text{Zones}[ZNinx]$) do

19 Set $RFstate = \text{LastZoneCompression}(\text{DataQueue}, m)$

20 if ($RFstate = \text{'FinishMode'}$)

21 then return [*DataQueue*, *Zones*, *ZonesBpSeq*, *ZoneCompHistory*, *Ap*, *AuxDataQueue*]

22 endif

23 Set $\text{Zones}[ZNinx] = \text{Zones}[ZNinx] + 1$

24 endwhile

25 Find $Start = \min\{k \mid \text{Zones}[k] = \text{Zones}[ZNinx]\}$

26 Set $\text{ZoneCompHistory}[Start, \text{Zones}[Start]] = \text{EquivalentZonesCompression}(\text{DataQueue}, \text{StartZone}, m)$

27 Set $\text{Zones}[Start] = \text{Zones}[Start] + 1$

28 for each *k* such that $Start + 1 \leq k \leq m - 1$ do

29 Set $\text{Zones}[k] = 0$

30 Set $\text{ZoneBpSeq}[k] = \text{null}$

31 for each *l* such that $0 \leq l \leq \text{Zone}[k]$ do

32 Initialize $\text{ZoneCompHistory}[k, l] = [\infty, -1]$

33 endfor

34 endfor

35 Set $DQinx = \sum_{j=0}^{j=Start} 2^{m-j-1} + 1$

36 endif

37 endwhile

Endprocedure

Then, the procedure calls the *EquivalentZonesCompression* procedure to compress these zones into the start zone and it saves the return values, which are the last selected sensor data and its absolute position, to *ZoneCompHistory* for the future compression preparation as the base sensor data and its absolute position of the next zone (procedure line 26). The procedure also increases one more the number of compressions of the last zone (procedure line 27) because the sensor data sequence in equivalent consecutive zones has been one more compressed into the start zone. As the sensor data sequence from the start zone to the last zone shrinks to only the start zone through the consecutive equivalent zones compression, the *DataQueue* memory spaces corresponding to the remaining zones are empty and the first position of the immediate zone of the start zone is the beginning *DataQueue* location to save the new ground sensor data. Thus, the procedure must initialize the data structure parts corresponding to these empty zones, such as the values from the immediate zone to the last zone in *Zones*, *ZoneBpSeq*, and *ZoneCompHistory*. The lines from procedure line 28 to line 34 reflect these initializations. The first position to save a new sensor data is calculated to the *DQinx* variable in procedure line 35.

8. Compressed Sensor Data Decompression and Lost Sensor Data Recovery

The final consumers of the return values from *SMEP* main are one or more destinations that receive the values via a wireless sensor network and one or more existing networks, and use them with their own purpose. One of main consumers is generally a monitoring and control center that monitors external or internal environments of each sensor nodes using sensor data transmitted by them, and controls them by sending control messages to sensor nodes if necessary. In order for the final consumer to use the *SMEP* main return values, the consumer must decompress them and recover the unselected sensor data during saving and compressing sensor data. In this section, we introduce algorithms for the decomposition of compressed sensor data and the recovery of lost sensor data.

8.1. Sensor Data Line Interpolation Algorithm

Sensor Data Line Interpolation Algorithm uses the line interpolation method to recover the lost sensor data. The line interpolation is a very simple method to interpolate the unknown values of points between two known endpoints. Algorithm 5 illustrates *SensorDataLineInterplolation*, a line interpolation algorithm for lost sensor data recovery:

Algorithm 5: *SensorDataLineInterplolation*(f_{value} , f_{pos} , l_{value} , l_{pos})

Parameters

01 f_{value} : the leftmost sensor data value of an interval for recovering compressed sensor data

02 f_{pos} : the leftmost absolute position of an interval for recovering compressed sensor data

03 l_{value} : the rightmost sensor data value of an interval for recovering compressed sensor data

04 l_{pos} : the leftmost absolute position of an interval for recovering compressed sensor data

Global Variables

Local Variables

01 x : the absolute position of a lost sensor data to be recovered

02 R_{value} : the value of a recovered lost sensor data with the absolute position x

Procedure

01 Output f_{pos} , f_{value}

02 Set $x = f_{pos} + 1$

03 while($x < l_{pos}$) do

04 Set $R_{value} = \frac{l_{value} - f_{value}}{l_{pos} - f_{pos}} \cdot (x - f_{pos}) + f_{value}$

05 Output x , R_{value}

06 Set $x = x + 1$

07 endwhile

Endprocedure

(f_{value} , f_{pos}) and (l_{value} , l_{pos}) are two known endpoints in the algorithm. These endpoints are two pair of value and absolute position about each of two consecutive compressed sensor data in

DataQueue. The procedure finds the line equation to pass two endpoints (procedure line 4). If there exist one or more absolute positions between these two endpoints, the sensor data at these absolute positions are those lost in compressions. Therefore, the procedure calculates approximately values at these absolute positions by substituting each of them into the line equation one by one (from procedure line 3 to line 7) and outputting each pair of the values and absolute positions of them (procedure line 5). This procedure is used in the *SensorDataRecovery* procedure introduced in the next section.

8.2. Sensor Data Recovery Algorithm

SensorDataRecovery is an algorithm that receives data structures (*DataQueue*, *Zones*, *ZoneBpSeq*, *ZoneCompHistory*, and *AuxDataQueue*) and a variable (*Ap*) from the SMEP main via a wireless sensor network and existing networks and decompresses or recovers the compressed and lost sensor data by the SMEP main sensor node during communication failure. In fact, all of the sensor data in each zone of *DataQueue* are the compressed or original ground sensor data finally selected in the same number of compressions. *SensorDataRecovery* algorithm finds absolute positions of remaining sensor data in each zone. At this time, the procedure uses *FindAbsoluteAddress* function and this function finds an absolute address using *DataQueue*, *Zones* and *ZoneBpSeq* arrays, when a sensor data position in *DataQueue* is given. Since how to calculate the absolute address using these arrays has already been introduced in the form of a formula in Theorem 8 in Section 6.2, we omit the detailed procedure about that function in this paper. With these sensor data values and found absolute positions, *SensorDataRecovery* procedure recovers the values of lost sensor data to the interpolated using *SensorDataLineInterpolation* procedure described in the previous section. The more detailed algorithm is presented in Algorithm 6. The procedure, at first, finds the last zone of which the zone value is not zero (procedure line 3). The zones from the zone 0 to the non-zero-valued zone are the target zones for decompression and recovery. The zones with 0 as their value includes the original ground sensor data without any compression. Every sensor data in the 0 valued zones doesn't need any decompression and recovery at all but it just needs output for its value and absolute position. The algorithm pseudo code about this corresponds to the part from the procedure line 29 to 40. The pseudo codes that executes decompression and recovery for none zero-valued target zones are shown from procedure line 5 to 27. Each zone needs a base sensor data for the first decompression and recovery. The pseudo codes for determination of a base sensor data are shown from the procedure line 6 to 9. The base sensor data in the 0 th zone is *DataQueue[0]* as $S_{c,0}$ and its absolute address is 0 (procedure line 7), since $S_{c,0}$ is always selected unconditionally in every compression. Meanwhile, the base sensor data for each target zones except the 0 th zone is the value and absolute position in *ZoneCompHistory* to have in its previous zone the same number of compressions with the target zone (procedure line 8). In order to execute decompression and recovery on *DataQueue* sensor data in each target zone with *DataQueue*, the procedure needs to know boundaries of *DataQueue* space corresponding to the target zone. The lowest boundary is determined by the greatest boundary of the previous target zone. Hence, the procedure determines only the greatest boundary as shown from the procedure line 10 to 13. Then, the procedure proceeds decompression and recovery for all sensor data, repeatedly (from procedure line 15 to 21). During doing this, the procedure finds an absolute position of each new sensor data in *DataQueue* through the *FindAbsolutePosition* function (procedure line 17). Then, the procedure calls *SensorDataLineInterpolation* with the values and absolute positions of two old and new sensor data so that each of lost sensor data between these two sensor data should be recovered to an line interpolated value and its absolute position and outputted (procedure line 18). The line interpolation needs two points. By the way, the last sensor data in each zone becomes one of two points as a partner of a previous or base sensor data but, in the next decompression and recovery, it needs a new partner for the line interpolation. So, the procedure uses as its new partner a sensor data of *ZoneCompHistory* of which the value and absolute position had played a role as a base sensor data for the next zone (from the procedure line 22 to 25). Then, the procedure recovers and outputs the compressed and lost sensor data between the last sensor data and its partner by calling *SensorDataLineInterpolation* (from the procedure line 26 to 29).

Algorithm 6: *SensorDataRecovery(DataQueue, Zones, ZoneBpSeq, ZoneCompHistory, Ap, AuxDataQueue)*

Parameters

01 *DataQueue, Zones, ZoneBpSeq, ZoneCompHistory, Ap, AuxDataQueue*: arrays or variables corresponding to SMEP main algorithm parameters

Global Variables

Local Variable

01 *DQinx*: an index variable for *DataQueue* array

02 *ZNMaxDQinx*: the maximum among indices of *DataQueue* space corresponding to a zone

03 *CompZoneMax*: the maximum among indices, *js*, such that *Zones[j]* is not zero

04 *CompZoneMaxAp*: the maximum among absolute positions covered by the *CompZoneMax* zone

05 *ZNinx*: an index variable in *Zones* array

06 *S_p, S_j*: the leftmost and rightmost values used to recover lost sensor data in an interval, [*p*, *j*]

07 *p, j*: the leftmost and rightmost absolute position used to recover lost sensor data in an interval, [*p*, *j*]

08 *ZNCV*: a value variable of *ZoneCompHistory[zone, Zones[zone]]*. Value for some zone

09 *ZNBP*: an absolute position variable of *ZoneCompHistory[zone, Zones[zone]]*. Position for some zone

Procedure

01 Set *DQinx* = 0

02 if (*Zones[0] ≠ 0*)

03 then Find *CompZoneMax* = $\max\{k \mid \text{Zones}[k] \neq 0\}$

04 Set *ZNinx* = 0, *CompZoneMaxAp* = $\sum_{j=\text{CompZoneMax}}^{m-j-1} 2^{\text{Zones}[j]}$

05 while ($0 \leq \text{ZNinx}$ and $\text{ZNinx} \leq \text{CompZoneMax}$) do

06 if (*ZNinx* = 0)

07 then Set [*S_p, p*] = [*DataQueue*[0], 0]

08 else Set [*S_p, p*] = *ZoneCompHistory*[*ZNinx* − 1, *Zones*[*ZNinx*]]

09 endif

10 if (*ZNinx* = *m* − 1)

11 then *ZNMaxDQinx* = *DQinx* +

12 else *ZNMaxDQinx* = *DQinx* + $2^{m-\text{ZNinx}-1}$

13 endif

14 Set *DQinx* = *DQinx* + 1

15 while (*DQinx* < *ZNMaxDQinx*) do

16 Set *S_j* = *DataQueue*[*DQinx*]

17 Set *j* = *FindAbsoluteAddress*(*DQinx*, *Zones*, *ZoneBpString*)

18 *SensorDataLineInterpolation*(*S_p, p, S_j, j*)

19 Set [*S_p, p*] = [*S_j, j*]

20 Set *DQinx* = *DQinx* + 1

21 endwhile

22 if (*ZNinx* = *CompZoneMax*)

23 then Set [*ZNCV, ZNBP*] = *ZoneCompHistory*[*ZNinx*, 1]

24 else Set [*ZNCV, ZNBP*] = *ZoneCompHistory*[*ZNinx*, *Zones*[*ZNinx*]]

25 endif

26 if (*DQinx* = *ZNMaxDQinx* and *j* ≠ *ZNBP*)

27 then *SensorDataLineInterpolation*(*S_j, j, ZNCV, ZNBP*)

28; else Output *S_j, j*

29; endif

30; endwhile

31 endif

32 if (*CompZoneMax* > 0 and *CompZoneMax* < *m* − 1 and *ZNBP* ≠ *CompZoneMaxAp*)

33 then *SensorDataLineInterpolation*(*ZNCV, ZNBP, DataQueue*[*DQinx* + 1], *CompZoneMaxAp* + 1)

34 else if (*CompZoneMax* = *m* − 1 and *ZNBP* ≠ *CompZoneMaxAp*)

35 then *SensorDataLineInterpolation*(*ZNCV, ZNBP, AuxDataQueue*[1], *CompZoneMaxAp* + 1)

36 endif

37 endif

38 if (*CompZoneMaxAp* < *m* − 1)

39 then Set *offset* = *Ap* − *CompZoneMaxAp*

40 for each *i* such that $1 \leq i \leq \text{offset}$ do

41 Output *DataZone*[*DQinx* + *i*], *CompZoneMaxAp* + *i*

42 endfor

43 else Set *offset* = *Ap* − *CompZoneMaxAp*

44 for each *i* such that $1 \leq i \leq \text{offset}$ do

45 Output *AuxDataQueue*[*i*], *CompZoneMaxAp* + *i*

46 endfor

47 endif

48 return

Endprocedure

Since if the zone is the last target zone then the first sensor data of the next zone is a ground sensor, the procedure selects, as a partner of the last sensor data, the sensor data of *ZoneCompHistory* to have the least compression 1 in that zone (procedure line 23). There may be the lost sensor data between the last partner selected from *ZoneCompHistory* and the first ground sensor data in the first 0 valued zone. Accordingly, this recovery process appears from the procedure line 32 to 37. This process is not necessary in the cases that the last zero-valued zone is the first zone 0 or the last partner selected from *ZoneCompHistory* is the greatest absolute positioned sensor data covered by the last target zone for decompression and recovery. The processes about these cases are reflected on the procedure lines 32 and 34, respectively. Meanwhile, if the last target zone is the last zone $m - 1$ of *DataQueue*, there exists in *AuxDataQueue* a next new partner of the last partner selected from *ZoneCompHistory*, because the new partner does not exist in the next zone any more. If not, however, the new partner exists in *DataQueue*. Each of recovery processes on these cases is reflected on procedure lines 34 and 35 and procedure lines 32, and 33. Furthermore, each case is involved in the next process for ground sensor data in the zero valued zones and *AuxDataQueue*. In other words, in the case that the last target is not the last zone of *DataQueue*, the procedure just outputs each remaining sensor data in *DataQueue* with its absolute position (from the procedure line 38 to 42) since all of ground sensor data remain in *DataQueue*. Otherwise, the procedure outputs each remaining sensor data in *AuxDataQueue* with its absolute position (from procedure line 38 to 42). Because ground sensor data do not exist in *DataQueue*, any more.

9. Performance Comparisons and Analysis

Until now, we have focused on minimizing error for the lost sensor data in compression intervals rather than on minimizing energy consumption, as have most of the current research works. Therefore, our performance evaluation focuses on the average error rate per recovered sensor data as one of performance factors. This section shows average error rates by experimental results based on real-field sensor data sets.

9.1. Experimental Environments and Evaluation Measure

9.1.1. Experimental Sensor Data Sets and Samples

Samples extracted from each of four real-field sensor data sets: underwater pH in the ocean, underwater temperature in the ocean, relative humidity in a city, and air temperature in a city. The underwater pH and underwater temperature sets were collected through a wireless sensor network developed by us from a real field: a littoral sea near Yokjido, which is a small island at Tongyeong-Si, Gyeongsangnam-do, Korea. The other sets, relative humidity and air temperature, were collected in Seoul, Korea, by the Meteorological Administration. Each sensor data in each of these four sets is a data collected every alternative hour.

In fact, before choosing these four sensor data sets, we had considered two types of sensor data sets as experimental sensor data sets: one is the type of sensor data set in which changes in the difference among sensor data values are frequent and high. The other is the type of sensor data set in which changes in the difference among sensor data values are infrequent and low. The reason for our consideration is that we had predicted higher error rates in sets with frequent and high value changes than in sets with infrequent and low value changes. Moreover, another reason was to ensure the validity of the average error rate of the SMEP method in sets with frequent and high value changes. Specifically, our main concerns are whether the average error rate in the SMEP method is valid in sets with frequent and high value changes, and until how many times compression is generally reasonable. Two sensor data sets, the relative humidity set and air temperature set, have more frequent and higher value changes than the two remaining sets, the underwater pH set and underwater temperature set. We chose different samples from these four sensor data sets, as shown in Table 2, depending on various experiments. How and why to choose them is described in detail in Sections 9.2.1 and 9.2.2.

Table 2. Experimental environments and performance evaluation measure.

Categories	Experimental Environments and Performance Evaluation Measure
Sensor Data Sets	Relative Humidity, Air Temperature, Underwater pH, Underwater Temperature
Samples and Sizes	consecutive 129 sensor data sequence extracted from each set, 25, 41, 73, 93 and 97 consecutive sensor data corresponding to 1-0-0-0, 2-0-0-0, 3-0-0-0, 3-2-2-1, and 3-0-0-0 Zones patterns, respectively, where $m-n-o-p$ Zones pattern means $Zones[0]=m$, $Zones[1]=n$, $Zones[2]=o$, and $Zones[3]=p$.
Experimental Tool	MATLAB R2014a
Recovery Method	linear interpolation
Comparison Targets	<i>winavg</i> , <i>delta</i> , <i>CQP</i> , <i>2MC</i> , and <i>SMEP</i> methods
Evaluation Measure	the average error rate (AER) $E_{M,S_c} = \frac{\sum_{i=1}^{i=n} S'_{0,i} - S_{0,i} }{\sum_{i=1}^{i=n} S_{0,i} }$

9.1.2. Comparison Target Methods

For comparison with other methods, we have chosen five compression methods, i.e., *winavg*, *delta*, *CQP*, *2MC*, and *SMEP*. We have applied their own methods to each sample in compression. We have applied a simple line interpolation to decompression and recovery in the *SMEP* method. We have not applied other interpolations such as two- or three-point spline interpolation, since we had used these interpolation methods but their average error rates were higher in each sample than average error rates by the line interpolation.

9.1.3. Performance Evaluation Measure

For performance evaluation, we use the average error rate defined as the below:

Definition 22. Let M and $S^0 = S_{0,0}, S_{0,1}, \dots, S_{0,n-1}$ be a compression method and a ground sensor data sequence, respectively. Given a compressed sensor data sequence of $S_0, S^c = S_{c,0}, S_{c,1}, \dots, S_{c,m}$, let $S^{0'} = S'_{0,0}, S'_{0,1}, \dots, S'_{0,n-1}$ be a decompressed and recovered sensor data sequence of S_c . Then, the average error rate (AER) E_{M,S^c} of S^c with respect to S^0 in the compression method M is defined as

$$E_{M,S^c} = \frac{\frac{\sum_{i=1}^{i=n} |S'_{0,i} - S_{0,i}|}{n}}{\frac{\sum_{i=1}^{i=n} |S_{0,i}|}{n}} = \frac{\sum_{i=1}^{i=n} |S'_{0,i} - S_{0,i}|}{\sum_{i=1}^{i=n} |S_{0,i}|}$$

From the above definition, note that an average error rate means an average ratio of the average difference value between the recovered sensor data and the original sensor data with respect to the average original sensor data value.

9.1.4. Experimental Tool and Method

We have used MATLAB R2014a as an experimental tool. With this tool, we have experimented with four compression methods per sample, while increasing the number of compressions one-by-one. With our experimental results, we have analyzed and evaluated our *SMEP* method, comparing it with other methods.

9.2. Experiments, Experimental Results and Analysis

In wireless sensor network literatures only a few methods for reducing the sensor data loss during communication failure have been proposed while a lot of the methods for reducing the number of sensor data transmissions with energy efficiency purpose have been mostly presented. They are *winavg*, *delta*, *CQP* and *2MC* methods as they have been briefly discussed in Section 2 and among the the

winavg and *delta* methods are other's works. In this section we show experimental results comparing the SMEP method with *winavg*, *delta*, *CQP* and *2MC* methods.

We performed experiments for two categories of analysis: the AER analysis in round compressions and the AER analysis in *Zones* value patterns. In Sections 9.2.1 and 9.2.2, we present the above two categories of experiments, their experimental results, and the analysis on them, respectively. In Section 9.2.2, moreover, we analyze characteristics of the SMEP method presenting graphs of *Zones* value patterns.

9.2.1. Average Error Rates in Round Compressions

We experimented with four samples for each of the five compression methods, which are *winavg*, *delta*, *CQP*, *2MC*, and *SMEP*. We constructed one sample from each of four real-field sensor data sets, which are relative humidity, air temperature, underwater pH, and underwater temperature sensor data sets. Each sample consisted of 129 consecutive ground sensor data extracted from their corresponding sensor data set. Varying the number of round compressions between one and four for each of four samples per method, we estimated individual AERs and we compared them for the five methods (here, the sizes of original sample in round compressions from one to four times are reduced to 50%, 25%, 12.5%, and 6.25% in compression ratios, respectively). In Table 3, we present the results.

Table 3. Average error rate (%) for each sensor data set in round compressions (CR: Compression Ratio (%)).

Sensor Data Sets		Relative Humidity				Air Temperature			
Num. of Round Comps.		1	2	3	4	1	2	3	4
Methods		CR: 50%	CR: 25%	CR: 12.5%	CR: 6.25%	CR: 50%	CR: 25%	CR: 12.5%	CR: 6.25%
<i>winavg</i>		4.60%	9.93%	14.30%	21.35%	5.70%	10.05%	19.66%	31.10%
<i>delta</i>		4.62%	12.61%	24.03%	38.40%	5.70%	15.29%	27.27%	50.44%
<i>CQP</i>		3.23%	6.57%	10.75%	21.57%	2.11%	4.55%	14.80%	34.02%
<i>2MC</i>		3.32%	7.43%	12.15%	25.25%	1.76%	4.39%	6.34%	29.99%
<i>SMEP</i>		2.80%	5.72%	10.75%	17.66%	1.24%	3.92%	5.83%	31.94%
Sensor Data Sets		Underwater pH				Underwater Temperature			
Num. of Round Comps.		1	2	3	4	1	2	3	4
Methods		CR: 50%	CR: 25%	CR: 12.5%	CR: 6.25%	CR: 50%	CR: 25%	CR: 12.5%	CR: 6.25%
<i>winavg</i>		0.46%	0.62%	0.82%	0.84%	0.22%	0.31%	0.44%	0.59%
<i>delta</i>		0.49%	0.94%	1.83%	3.39%	0.22%	0.51%	0.82%	1.44%
<i>CQP</i>		0.36%	0.68%	1.09%	1.84%	0.20%	0.34%	0.53%	0.71%
<i>2MC</i>		0.29%	0.44%	0.68%	0.81%	0.17%	0.34%	0.45%	0.82%
<i>SMEP</i>		0.24%	0.39%	0.62%	0.62%	0.16%	0.32%	0.45%	0.69%

Compared with the other methods, the SMEP shows for each round compressions better AER than the other methods, in the sensor data samples with more frequent and higher value changes, i.e., in the relative humidity and air temperature samples.

In the underwater pH and underwater temperature samples with infrequent and low value changes, though the differences among the AERs of five methods in the same round compressions are small, the SMEP shows mostly better AER than the other methods. Specifically, comparing with the *2MC*, note that the AER of the SMEP method is better than the *2MC*. Meanwhile, in Table 2 the SMEP and the *2MC* show that their AER differences between these methods and the other methods are much bigger in the relative humidity and air temperature samples than in the underwater pH and underwater temperature samples. As an example, in Table 2, while the AER difference between the SMEP and *delta* in the 3rd round compression of the underwater temperature sample is only 0.37%, while their AER difference in the same round compression of the air temperature sample is 21.44%. Therefore, in compressions of ground sensor data with more frequent and higher value changes, the SMEP and *2MC* methods are more effective than the other methods.

We can show that the number of round compressions until which a method is feasible depends definitely on the sensor data change properties in all the compression methods. As shown in Table 2,

in all methods, the underwater pH and underwater temperature samples with infrequent and low value changes show the AER difference between the 4th round compression and each of the other round compressions is so small that we can generally ignore it. In contrast to this, every method shows the 4th round compression is not feasible in the relative humidity and air temperature samples with frequent and high value changes. Of course, even if the maximum feasible number of round compressions depends on sensor data properties and sample periods, in these cases, Table 2 shows that the SMEP method is generally feasible up to 3rd round compressions in the case of the samples with frequent and high value changes. In fact, the maximum feasible number of round compressions depends entirely on how finely or frequently sensor data are sampled. The more frequently sampled sensor data are, the higher the maximum feasible number of round compressions is.

9.2.2. Average Error Rates in Zones Value Patterns

One among major merits of the SMEP method is *Zones* compression. Such a merit is based on *Zones* rules, in which the numbers of compressions can be different among zones of *DataQueue*. In fact, the round compressions in the experiments described in Section 9.2.1 are the special cases in which values of the non-zero value zones are all the same and there is no ground sensor data in zero-value zones. Hence, these cases are rare. The cases of *Zones* value patterns introduced in this section are much more general and frequent than the cases of round compressions. From now on, we let *m-n-o-p Zones* value pattern or, shortly, *m-n-o-p Zones pattern* mean that values of *Zones[0]*, *Zones[1]*, *Zones[2]*, and *Zones[3]* are *m*, *n*, *o*, and *p*, respectively, in the *DataQueue[0..16]*. In other words, *m-n-o-p Zones pattern* means that sensor data corresponding to *Zones[0]*, *Zones[1]*, *Zones[2]*, and *Zones[3]* in *DataQueue[0..16]* are compressed *m*, *n*, *o*, and *p* times, respectively.

We tried to carry out experiments for investigating the merits in the virtue of *Zones* patterns. For doing these, we prepared *DataQueue[0..16]*. Actually, the *DataQueue* size is much bigger than this size because the *DataQueue* size can be the space size capable of accommodating $2^m + 1$ sensor data from available free memory space. The real sizes of *DataQueue* in current technologies can generally range from dozens of kilobytes to hundreds of megabytes in available flash memory spaces. Despite this, the reason why we prepared such a small-sized *DataQueue* is to validate the merits of the SMEP method in *Zones* compressions, just with small *DataQueue* size. We also prepared five *Zones* patterns, i.e., 1-0-0-0, 2-0-0-0, 3-0-0-0, 3-2-2-1, and 3-2-2-2, by randomly choosing them because there are too many cases to investigate all cases in *Zones* patterns. The sample sizes corresponding to these patterns are 25, 41, 73, 93, and 97 in order that the ground sensor data should be full in each 0-valued zone. We experimented with five samples with these sizes for each of the five compression methods, which are *winavg*, *delta*, *CQP*, *2MC*, and *SMEP*. Five samples are constructed for each of the four real-field sensor data sets, which are the relative humidity, air temperature, underwater pH, and underwater temperature sensor data sets. According to sample sizes corresponding to the five *Zones* patterns, each of the five samples consisted of consecutive ground sensor data extracted randomly from each of these four sensor data sets. With these samples corresponding to five *Zones* patterns, we estimated the individual AERs and compared them for the five methods. In Table 4, we present the results.

Table 4 shows us that even in terms of the zones value patterns, the AER of the SMEP method is better than the AERs of the other methods, too. Figure 12 shows more explicitly the graph shape changes in accordance with *Zones* patterns. The figure shows not only graph patterns of sensor data recovered from relative humidity sensor data compressed by the *winavg*, *delta*, *CQP*, *2MC*, and *SMEP* methods, but also comparisons among between of these patterns and the original sensor data pattern.

Table 4. Average error rates in sensor data samples corresponding to *Zones* value patterns (*DataQueue* size: 16, CR: Compression Ratio (%)).

Sensor Data Sets		Relative Humidity					Air Temperature				
<i>Zones</i>		1-0-0-0	2-0-0-0	3-0-0-0	3-2-2-1	3-2-2-2	1-0-0-0	2-0-0-0	3-0-0-0	3-2-2-1	3-2-2-2
Methods		CR: 67%	CR: 40%	CR: 22%	CR: 17%	CR: 16.7%	CR: 67%	CR: 40%	CR: 22%	CR: 17%	CR: 16.7%
<i>winavg</i>		4.36%	11.03%	14.42%	14.36%	14.30%	5.78%	12.67%	20.64%	19.67%	19.66%
<i>Delta</i>		4.62%	11.93%	24.10%	25.10%	24.03%	5.93%	17.78%	28.29%	28.47%	27.27%
<i>CQP</i>		2.59%	6.71%	11.55%	11.27%	10.75%	2.40%	6.22%	16.70%	15.27%	14.80%
<i>2MC</i>		2.35%	5.15%	10.99%	12.05%	11.50%	1.65%	5.56%	6.97%	6.26%	6.08%
<i>SMEP</i>		2.10%	2.57%	7.75%	8.71%	8.64%	1.36%	3.64%	5.86%	5.42%	5.25%
Sensor Data Sets		Underwater pH					Underwater Temperature				
<i>Zones</i>		1-0-0-0	2-0-0-0	3-0-0-0	3-2-2-1	3-2-2-2	1-0-0-0	2-0-0-0	3-0-0-0	3-2-2-1	3-2-2-2
Methods		CR: 67%	CR: 40%	CR: 22%	CR: 17%	CR: 16.7%	CR: 67%	CR: 40%	CR: 22%	CR: 17%	CR: 16.7%
<i>winavg</i>		1.29%	1.20%	0.96%	0.86%	0.82%	0.25%	0.32%	0.50%	0.45%	0.44%
<i>Delta</i>		1.35%	1.95%	2.30%	1.91%	1.83%	0.27%	0.53%	0.91%	0.83%	0.82%
<i>CQP</i>		0.90%	1.25%	1.35%	1.14%	1.09%	0.34%	0.43%	0.61%	0.53%	0.53%
<i>2MC</i>		0.48%	0.70%	0.83%	0.74%	0.71%	0.28%	0.36%	0.52%	0.46%	0.46%
<i>SMEP</i>		0.36%	0.61%	0.68%	0.62%	0.64%	0.22%	0.36%	0.49%	0.45%	0.44%

m-n-o-p *Zones* pattern means *Zones*[0] = m, *Zones*[1] = n, *Zones*[2] = o, and *Zones*[3] = p.

In the 1-0-0-0 *Zones* pattern, sensor data from the 1st to the 17th covered by *Zones*[0] are one-time compressed data, but sensor data from the 17th sensor data to the 25th sensor data covered by *Zones*[1], *Zones*[2], and *Zones*[3], of which the values are 0, are ground sensor data. Figure 12a shows the 1-0-0-0 *Zones* pattern. In this graph, we can see that the *CQP*, *2MC*, and *SMEP* methods follow patterns similar to original sensor data.

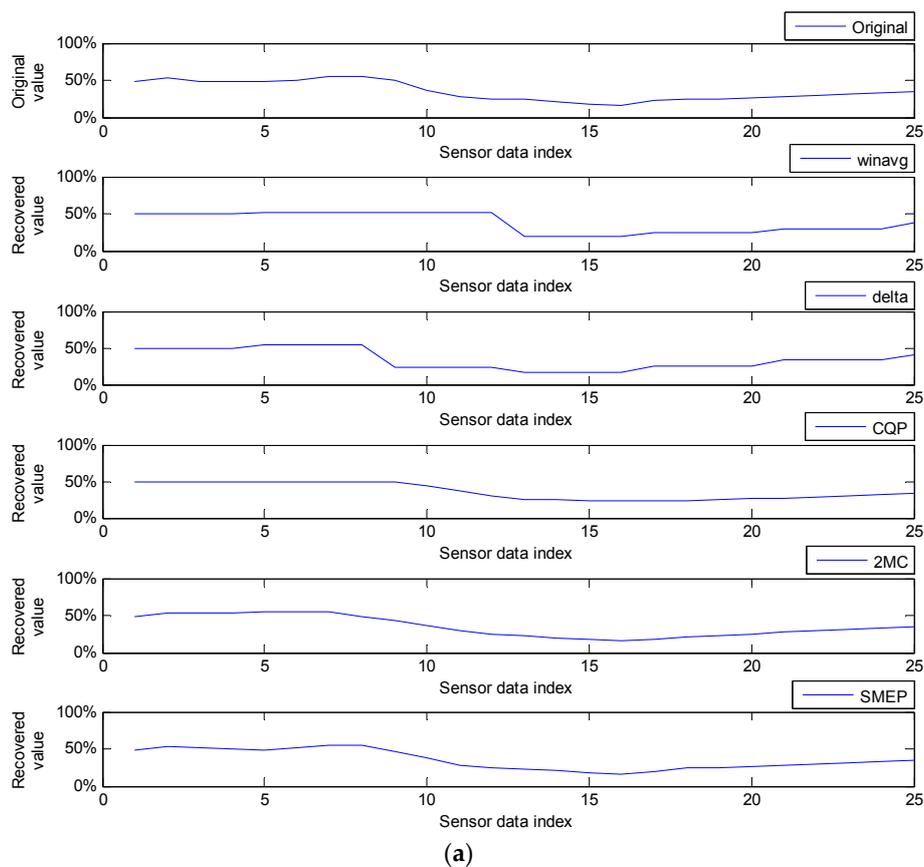
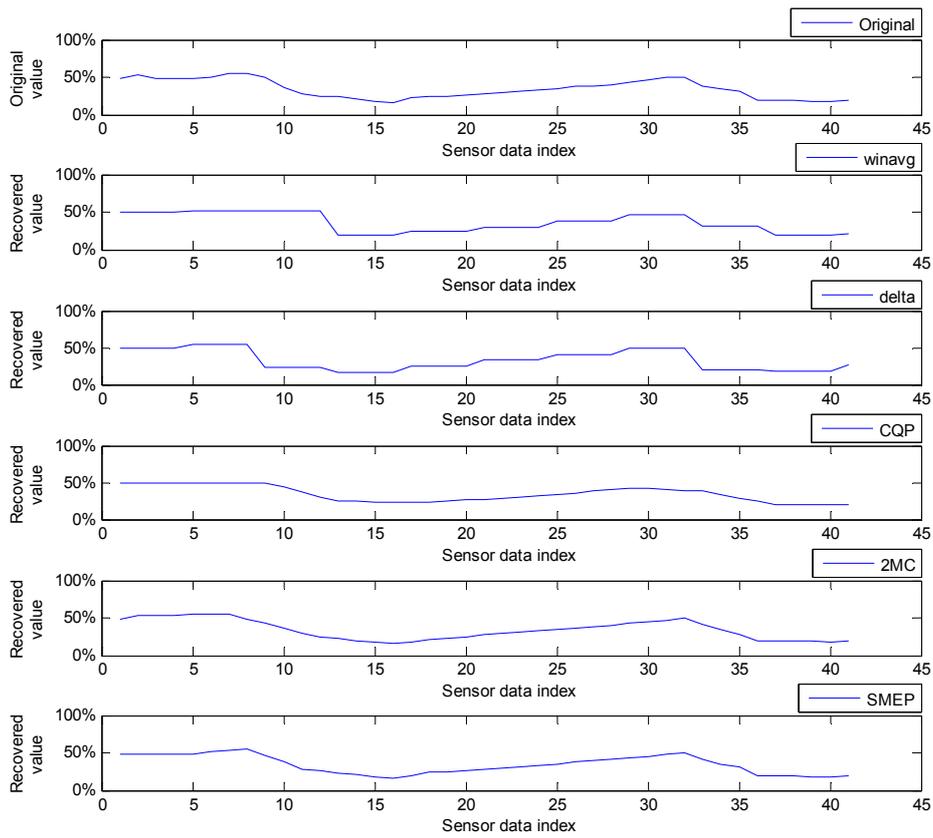
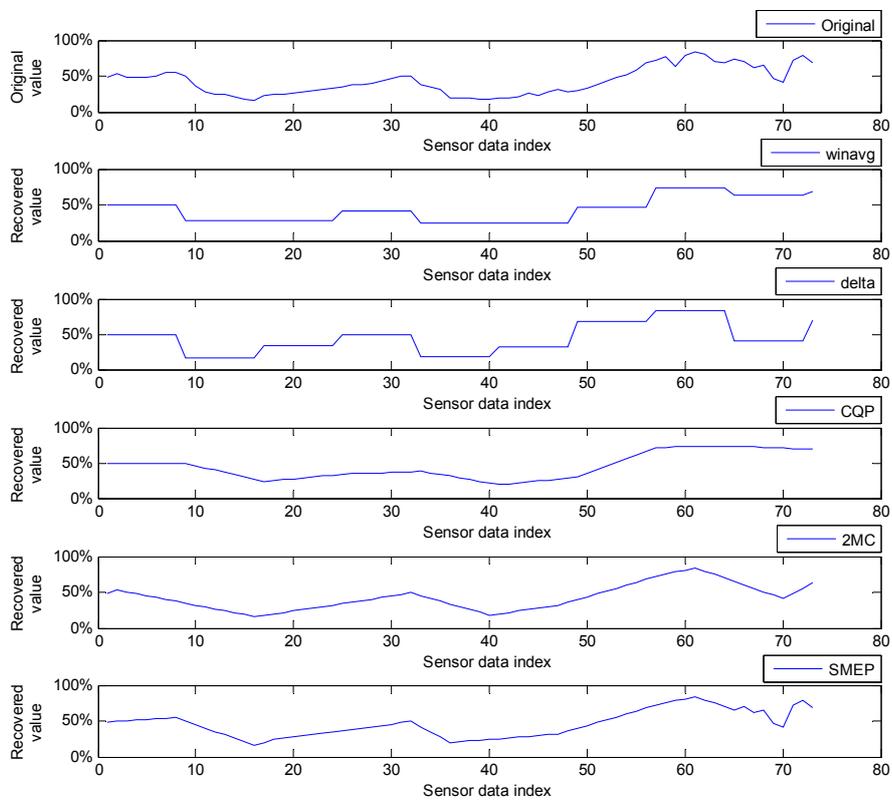


Figure 12. Cont.

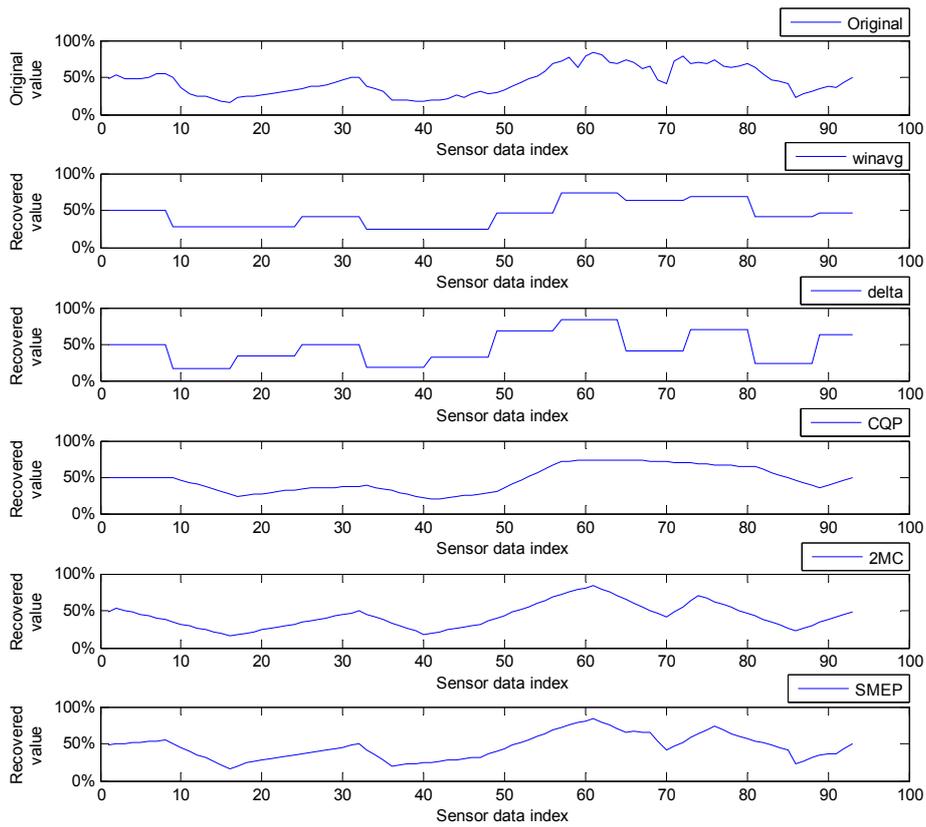


(b)

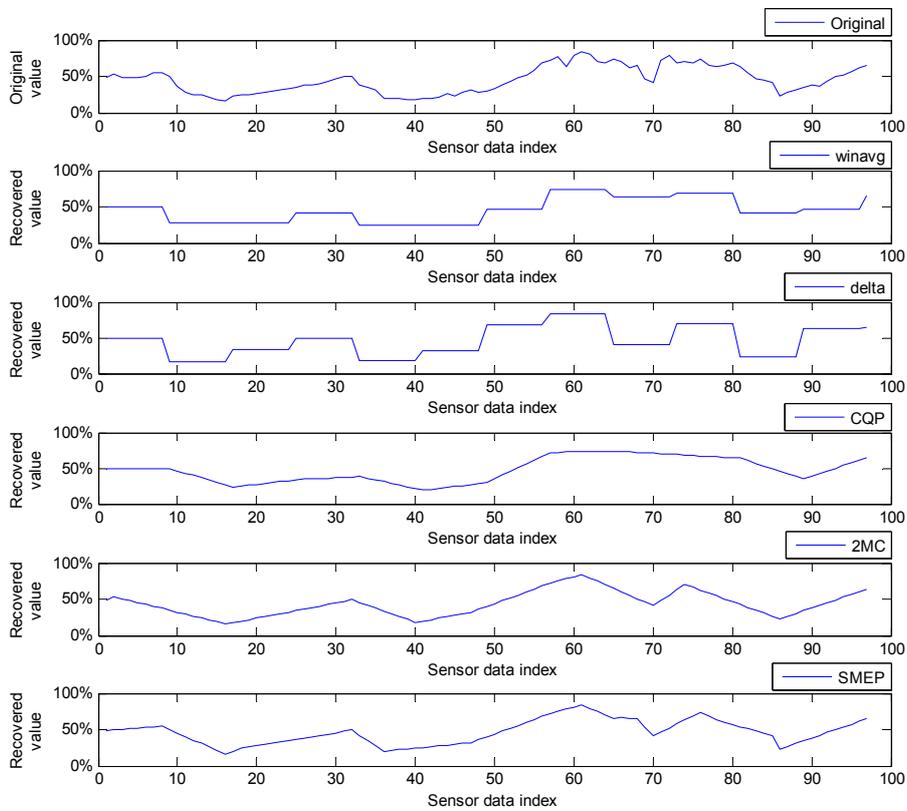


(c)

Figure 12. Cont.



(d)



(e)

Figure 12. Comparison of graph shape changes in accordance with *Zones* patterns in relative humidity. (a) *Zones* pattern: 1-0-0-0; (b) *Zones* pattern: 2-0-0-0; (c) *Zones* pattern: 3-0-0-0; (d) *Zones* pattern: 3-2-2-1; (e) *Zones* pattern: 3-2-2-2.

Figure 12b shows the 2-0-0-0 *Zones* pattern. In the 2-0-0-0 *Zones* pattern, sensor data between the 1st sensor data and the 33th sensor data are two-time compressed data covered by *Zones*[0], but sensor data between the 34th sensor data and the 41th sensor data are original sensor data. Like the 1-0-0-0 *Zones* pattern, the *CQP*, *2MC*, and *SMEP* methods follow patterns similar to original sensor data. Nonetheless, unlike the other methods, including *CQP* and *2MC*, the *SMEP* method shows that part of the graph between the 34th sensor data and the 41th sensor data is the same pattern as in the original sensor data graph. Actually, this graph part indicates that the *SMEP* method has a better AER between the original sensor data and its own sensor data part than the other methods.

In the 3-0-0-0 *Zones* pattern, *Zones*[0] sensor data between the 1st sensor data and the 65th sensor data and the sensor data between the 66th sensor data and the 73th sensor data are original sensor data covered by *Zones*[1], *Zones*[2], and *Zones*[3]. Figure 12c shows that the graph part covered by *Zones*[0] is more similar to the corresponding original sensor data part than for the other methods. In particular, the graph part covered by *Zones*[1], *Zones*[2], and *Zones*[3] is exactly the same as the corresponding original sensor data part.

In the 3-2-2-1 and 3-2-2-2 *Zones* patterns, Figure 12d,e show that graph shapes of the *SMEP* method are much more similar to the graph shape of the original sensor data than for the other methods. In fact, the graph shape becomes more similar to the original sensor data than the other methods if the *Zones*[0] value is greater than the values of the next non-zero value zones.

One of the reasons for this, as shown in Section 9.2.1, is that the *SMEP* method generally has a better AER performance than the other methods. Moreover, the more important reason is that in the *SMEP* method, the number of compressions of sensor data in one zone is different from that in the other zones if the *Zones*[0] value is greater than the values of the next non-zero value zones. In the other methods, however, every sensor data has the same number of compressions as *Zones*[0]. Thereby, the *SMEP* method tends to exhibit better AER performance effects in the zones with the lower number of compressions than that in *Zones*[0].

10. Conclusions

In this paper, we have proposed a communication framework and feasible method for reducing sensor data that can be lost during communication failure. Our formal approach has also come up with a theoretical basis for problems on the reduction of sensor data loss during communication failure. Consequently, in the comparison with current compression methods, the *SMEP* method has shown better performance in average error rate per sensor data than others.

Meanwhile, the current technologies have developed and commercialized such cheap micro controller units with flash memories into which program code or data can be permanently saved. Moreover, they are currently used in various wireless sensor network applications. For example, the ATmega128 and Cortex M3 have 128 KB and 512 KB flash memory, respectively. We can use not all of these memory spaces but some parts of them. Under such a possibility, the point is that most current sensor nodes can permanently save compressed and ground sensor data to its own flash memory. Accordingly, our proposed *SMEP* method is feasible in most of current sensor node technologies. Nonetheless, most sensor data will be lost during communication failures without taking any action.

In the implementation of the *SMEP* method, the *SMEP* functionalities cannot be loaded into the routing layer in the protocol stack, because sensor data compression is not the intrinsic function of the routing layer and, in the *SMEP* method, each compressed or ground sensor data is necessary to be handled not as a packet with extra information, such as a header or footer, but as bare data without any extra information. For these reasons, we recommend that the *SMEP* functionalities should be embedded into an upper layer of the routing layer. Several research works have introduced and implemented a kind of database layer, such as a query layer [23,34–36,45]. In these, the database layer commonly plays a role of data management between the routing layer and upper layers, such as application layer. For the detail on the data layer, refer to the above-referenced literature. Such a database layer or query layer could be one of our recommended upper layers. Another possibility for

implementation could be a cross layer to merge the functionalities of the database and routing layers, of course.

On the other hand, the SMEP method alternately applies the *selection by pattern* rule and the *selection by minimum error* rule for each compression interval. Therefore, given two consecutive compression intervals, one is applied by the *selection by pattern* rule and the other is applied by the *selection by minimum error* rule. Now, we expect that we can improve the average error rate if the *selection by minimum error* rule is applied to not one compression interval but several consecutive compression intervals. As further work, we are going to research not only how to design this method, but also how many consecutive compression intervals are optimal for performance and how much its performance is improved depending on the number of consecutive compression intervals.

Acknowledgments: This research was a part of the project titled Development of the wide-band underwater mobile communication systems funded by the Ministry of Oceans and Fisheries, Korea.

Author Contributions: Changhwa Kim proposed the idea of the SMEP method and carried out its formal approach; Changhwa kim and DongHyun Shin performed conceived and designed experiments; DongHyun Shin performed the experiments; Changhwa Kim and DongHyun Shin analyzed the data; Changhwa Kim wrote the paper.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Proof of Theorem 1

Proof. Since $I = [i, j]$ and $J = [k, l]$ are interval $i < j$ and $k < l$. Moreover, since compression intervals I and J are consecutive, either $j = k$ or $l = i$ is true. But, if $l = i$ -then $k < l = i < j$ and this is a contradiction to the conditions of the theorem, $i < k$ or $i < l$ or $j < l$. Thus, $j = k$ is true. Therefore, by the definition of consecutive compression interval, $S_i, S_{i+1}, \dots, S_{j-1}, S_j$ and S_k, \dots, S_{l-1}, S_l are consecutive sensor data, respectively, and since $S_j = S_k = S_{i+m}$, $[i, l] = S_i, S_{i+1}, \dots, S_{j+m-1}, S_j, S_{k+1}, \dots, S_{l+1}, S_l$ are success sensor data and the number of them is $n + m$. By definition, $J = [i, l]$ is a compression interval with the size $m + n$ with respect to S . \square

Appendix B. Proof of Theorem 3

Proof of Proposition 1. In initial step, if $m = 1$, $C_S = (I_0)$ and $I_0 = S$. Thus, $|I_0| = |S| = 3 = 2 \cdot 1 + 1$. So, when $m = 0$, *Proposition 1* is true. Assume that the proposition is true for an integer k , $1 < k < m$. For $k = m$, $|S| = |I_0 \bullet I_1 \bullet I_2 \bullet \dots \bullet I_m| = |(I_0 \bullet I_1 \bullet I_2 \bullet \dots \bullet I_{m-1}) \bullet I_{m-1}|$ (by Theorem 2) = $|I_0 \bullet I_1 \bullet I_2 \bullet \dots \bullet I_{m-1}| + |I_m|$ (by Theorem 1) = $(2(m - 1) + 1) + 2$ (since $|I_0 \bullet I_1 \bullet I_2 \bullet \dots \bullet I_{m-1}| = 2(m - 1) + 1$ by the assumption) = $2m + 1$. By induction, *Proposition 1* is true. \square

Proof of Proposition 2. For p, q, r of S_p, S_q, S_r in I_i , $q = p + 1$ and $r = q + 1 = p + 2$, since they are consecutive in a compression interval. $p = 0$ for $i = 0$ since I_0 is the first compression interval and S_p is the first sensor data and so, $p = 0 = 2 \cdot 0$, $q = 1 = 2 \cdot 0 + 1$, $r = 2 = 2 \cdot 0 + 2$. Thus, *Proposition 2* is true for $i = 0$. Assume that the proposition is true for $0 < k < i$. For $k = i - 1$, the position of the last sensor data in the previous compression interval I_{i-1} is $2(i - 1) + 2$ (by the assumption) = $2i$. Since I_{i-1} and I_i are consecutive compression intervals, the last sensor data in I_i is shared as the same one as the first sensor data in I_i . Thus, $p = 2i$. Consequently, $q = 2i + 1$ and $r = 2i + 2$. Thus, by induction, *Proposition 2* is true for an integer, i , $0 \leq i \leq m - 1$. \square

Appendix C. Proof of Theorem 4

Proof. Let the size of S be n . If $m = 1$, then $n = 3$, $I_0 = S = S_0, S_1, S_2$, $C_S = (I_0)$, and $k = 0 = 2^{1-1} - 1$. Thus, the theorem is true for $m = 1$. Assume that the theorem is true for any integer k , $k < m$. If $k = m$,

the size of S is $2^m + 1$. So, let $S = S_0, S_1, \dots, S_{2^{m-1}-1}, S_{2^{m-1}}, S_{2^{m-1}+1}, \dots, S_{2^m}$. Since $2m = 2 \cdot 2^{m-1} = 2^{m-1} + 2^{m-1}$, $S = S_0, S_1, \dots, S_{2^{m-1}}, S_{2^{m-1}+1}, \dots, S_{2^{m-1}+2^{m-1}}$. Now, divide S into two same size sensor data sequences, $P = S_0, S_1, \dots, S_{2^{m-1}}$ and $Q = S_{2^{m-1}}, S_{2^{m-1}+1}, \dots, S_{2^{m-1}+2^{m-1}}$, respectively, where $S_{2^{m-1}}$ is common. Therefore, each size of the two is $2^{m-1} + 1$. Consider two 2-size compression intervals, $C_P = (J_0, J_1, \dots, J_p)$ and $C_Q = (H_0, H_1, \dots, H_q)$ with respect to P and Q , respectively. Then, by assumption, $p = q = 2^{m-2} - 1$, thus number of compression intervals in both of C_P and C_Q is 2^{m-2} . Moreover, $(J_0, J_1, \dots, J_p, H_0, H_1, \dots, H_q) = C_s = (I_0, I_1, \dots, I_l)$, and since the number of compression intervals in C_s is $2^{m-2} + 2^{m-2} = 2^{m-1}$, $l = 2^{m-1} - 1$ and the theorem has been proved. \square

Appendix D. Proof of Theorem 5

Proof. We prove the theorem by induction. At first, for $k = 1$, $S_{1,i}$ covers $I_{0,i-1} = I_{0,(i-1)2^0} = I_{k,(i-1)2^{k-1}}$ by Lemma 1. Assume that the theorem is true for integer $k \leq c$. By Lemma 2, $S_{c,i}$ covers $I_{c-1,i-1}$ and $I_{c-1,i-1}$ covers $I_{c-2,2(i-1)} \bullet I_{c-2,2i-1}$. By assumption, $I_{c-2,2(i-1)}$ covers $I_{0,2 \cdot 2(i-1)2^{c-3}} \bullet I_{0,2 \cdot 2(i-1)2^{c-3}+1} \bullet \dots \bullet I_{0,2 \cdot (2(i-1)+1)2^{c-3}-1} = I_{0,(i-1)2^{c-1}} \bullet I_{0,(i-1)2^{c-1}+1} \bullet \dots \bullet I_{0,(2i-1)2^{c-2}-1}$ and $I_{c-2,2i-1}$ covers $I_{0,2(2i-1)2^{c-3}} \bullet I_{0,2(2i-1)2^{c-3}+1} \bullet \dots \bullet I_{0,2 \cdot 2i \cdot 2^{c-3}-1} = I_{0,(2i-1)2^{c-2}} \bullet I_{0,(2i-1)2^{c-2}+1} \bullet \dots \bullet I_{0,i2^{c-1}-1}$. Thus, $S_{c,i}$ covers $I_{c-2,2(i-1)} \bullet I_{c-2,2i-1} = I_{0,(i-1)2^{c-1}} \bullet I_{0,(i-1)2^{c-1}+1} \bullet \dots \bullet I_{0,(2i-1)2^{c-2}-1} \bullet I_{0,(2i-1)2^{c-2}} \bullet I_{0,(2i-1)2^{c-2}+1} \bullet \dots \bullet I_{0,i2^{c-1}-1}$ and the theorem has been proved. \square

Appendix E. Proof of Theorem 7

Proof. Since $p = 4k + 1$ and $q = 4k + 2$, by definition,

$$\begin{aligned} E_p = E_{4k+1} &= |S_{c,q} - A| = |S_{c,4k+2} - A| = \left| S_{c,4k+2} - \left(\frac{S_{c,j} - S_{c,4k+1}}{j' - p'} (q' - p') + S_{c,4k+1} \right) \right| \\ &= \left| - \frac{S_{c,j} - S_{c,4k+1}}{j' - p'} (q' - p') + (S_{c,4k+2} - S_{c,4k+1}) \right| \\ &= \left| \frac{S_{c,j} - S_{c,4k+1}}{j' - p'} (q' - p') - (S_{c,4k+2} - S_{c,4k+1}) \right| \\ &\quad \text{and} \\ E_q = E_{4k+2} &= |S_{c,p} - B| = |S_{c,4k+1} - B| = \left| S_{c,4k+1} - \left(\frac{S_{c,i} - S_{c,4k+2}}{i' - q'} (p' - q') + S_{c,4k+2} \right) \right| \\ &= \left| \frac{S_{c,i} - S_{c,4k+2}}{i' - q'} (q' - p') - (S_{c,4k+2} - S_{c,4k+1}) \right| \end{aligned}$$

\square

Appendix F. Proof of Theorem 8

Proof. In proof of (i), above all, for any j such that $0 \leq j \leq m - 1$ the *DataQueue* size covered by *Zones*[j] is 2^{m-j-1} . If *DataQueue*[i] exists in the zone k , then $\sum_{j=0}^{j=k-1} 2^{m-j-1} < i < \sum_{j=0}^{j=k+1} 2^{m-j-1}$ since $k - 1 < k < k + 1$. Accordingly, $i - \sum_{j=0}^{j=k-1} 2^{m-j-1} > 0$ and $i - \sum_{j=0}^{j=k+1} 2^{m-j-1} < 0$. Thus, k is the maximum number to satisfy $i - \sum_{j=0}^{j=k-1} 2^{m-j-1} > 0$, this is, $k = \max \left\{ t \mid 0 < i - \sum_{j=0}^{j=t-1} 2^{m-j-1} \right\}$. Since $\sum_{j=0}^{j=t-1} 2^{m-j-1} = 2^{m-1} + 2^{m-2} + \dots + 2^{m-t} = 2^m - 2^{m-t} = 2^{m-t} (2^t - 1)$, $k = \max \left\{ t \mid 0 < (i - 2^{m-t} \cdot (2^t - 1)) \right\}$. If $i = 2^m$, i is the last position. Hence, i exists in the last zone $m - 1$.

In the (ii) proof, at first, if $i = 0$, $P_i = 0$, since $S_{c,0} = S_{0,0}$ for any number of compressions c . Now, we prove the case for $i \neq 0$ and $k \geq 0$. Let k be the zone in which i position is located. The absolute address P_i of i position in *DataQueue* consists of a sum of the three parts:

$$P_i = P_B + P_{k,i-1} + P_{k,i} \quad (\text{A1})$$

where P_B is the number of absolute positions covered by zones from the zone 0 to the zone $k - 1$, $P_{k, i-1}$ is the number of absolute positions within a zone k which covered by sensor data from 1 st sensor data to the $i - 1$ th positioned sensor data in the zone, P_{k, I_i} is an I -bit position address of the i -th position within the compression interval in which the i -th position is located.

Each zone j has 2^{m-j-1} sensor data in *DataQueue* and each of them has been selected among $2^{Zones[j]}$. Therefore, each zone i covers $2^{m-j-1} \cdot 2^{Zones[j]}$ absolute positions. Since zones from the zone 0 to the zone $k - 1$ are located before the zone k ,

$$P_B = \sum_{j=0}^{k-1} 2^{m-j-1} \cdot 2^{Zones[j]} \quad (A2)$$

Meanwhile, the number of relative positions in the zone k for $I - 1$ position is $j - 1 - \sum_{j=0}^{k-1} 2^{m-j-1} \cdot 2^{Zones[j]}$ and each of $j - 1 - \sum_{j=0}^{k-1} 2^{m-j-1} \cdot 2^{Zones[j]}$ sensor data covers $2^{Zones[k]}$ in the zone k since it has passed through $Zones[k]$ compressions. Thus,

$$P_{k, i-1} = (j - 1 - \sum_{j=0}^{k-1} 2^{m-j-1}) \cdot 2^{Zones[k]} \quad (A3)$$

The relative position of the position i in the zone k is $j - \sum_{j=0}^{k-1} 2^{m-j-1} \cdot 2^{Zones[j]}$. Using this position, the I -bit position sequence $ZoneBpSeq[k]$ in the zone k , and the *BitSubstring* function,

$$P_{k, I_i} = (\text{BitSubstring}(\text{ZoneBpSeq}[k], \text{Zones}[k], i - \sum_{j=0}^{j=k-1} 2^{m-j-1})) + 1 \quad (A4)$$

Replacing Equation (1) with Equation (2)–(4) finally,

$$\begin{aligned} P_i &= P_B + P_{k, i-1} + P_{k, I_i} \\ &= \sum_{j=0}^{k-1} 2^{m-j-1} \cdot 2^{Zones[j]} + (j - 1 - \sum_{j=0}^{k-1} 2^{m-j-1}) \cdot 2^{Zones[k]} + \\ &\quad (\text{BitSubstring}(\text{ZoneBpSeq}[k], \text{Zones}[k], i - \sum_{j=0}^{j=k-1} 2^{m-j-1})) + 1 \end{aligned}$$

We have completed the proof of the theorem. \square

Appendix G. Proof of Theorem 9

Proof. Let L be the size of a sensor data sequence from the *StartZone* zone to the last zone. Then,

$$L = 2^{m-StatZone}.$$

Because $L = 2^m$ – the number of *DataQueue* positions in zones before the *StartZone* zone

$$= 2^m - \sum_{j=0}^{StartZone-1} 2^{m-j-1} = 2^m - (2^m - 2^{m-StartZone}) = 2^{m-StartZone}.$$

Given a sensor data sequence with even number size, the sensor data sequence after compression becomes to be reduced to the half size. Thus, the compressed sensor data sequence with the size L is reduced to the size $\frac{1}{2}L = \frac{1}{2} \cdot 2^{m-StatZone} = 2^{m-StatZone-1}$, which is the same as the size of *StartZone* zone. Thus, the sensor data sequence in consecutive equivalent zones from the *StartZone* zone to the last $m - 1$ zone is compressed to the *Startzone* zone. \square

Appendix H. Proof of Lemma 1

Proof. By induction, for $k = 0$, the theorem is true because a sensor data selected in $I_{c,0}$ is the one selected among sensor data selected in $I_{c-1,0}=I_{c,2 \cdot 0}$ and $I_{c-1,1} = I_{c,2 \cdot 0+1}$ so that $I_{c,0}$ covers $I_{c-1,2 \cdot 0} \bullet I_{c-1,2 \cdot 0+1}$. Assume that the theorem is true for any integer $i \leq k - 1$. Consider $I_{c-1,2k-2}$, $I_{c-1,2k-1}$, $I_{c-1,2k}$, and $I_{c-1,2k+1}$. They are consecutive depending on the order of compression intervals in C_s^c . Here, $I_{c-1,2k-2} = I_{c-1,2(k-1)}$ and $I_{c-1,2k-1} = I_{c-1,2(k-1)+1}$, $I_{c-1,2k-2}$ and therefore, $I_{c,k-1}$ covers $I_{c-1,2k-2} \bullet I_{c-1,2k-1}$. Thus, the $I_{c,k}$ must cover $I_{c-1,2k} \bullet I_{c-1,2k+1}$. Consequently, for $i \leq k - 1$, the theorem is true. \square

Appendix I. Proof of Lemma 2

Proof. $S_{c,i}$ is selected in $I_{c-1,i-1}$. Thus, $I_{c-1,i-1} = I_{c-1,j}$ and $j = i - 1$. Moreover, since $S_{c,i}$ covers $I_{c-1,i-1}$, by lemma 1, $I_{c-1,j}$ covers $I_{c-2,2(i-1)} \bullet I_{c-2,2i-1} = I_{c-2,2(i-1)} \bullet I_{c-2,2(i-1)+1}$. \square

Appendix J. Proof of Lemma 3

Proof. The lemma 3 is trivial. Because the I-bit position of $S_{c,k}$ is positioned at the k -th I-bit position in the I-bit position sequence, the length of I-bit position is c , and I-bit position of $S_{c,k}$ is located at the $c(k - 1)$ th position in the I-bit position sequence (note that the first position in I-bit position sequence is 0). \square

Appendix K. Proof of Lemma 4

Proof. For $k = 0$, $S_{c,0}$ is always selected as a first sensor data unconditionally from $S_{0,0}$. Therefore, $i = 0$. For $k \geq 1$, according to corollary of Theorem 5, the position of $S_{c,k}$ exists in $[(k - 1)2^c, k2^c]_0$, the $k - 1$ th compression interval of S^0 . Thus, the absolute position of the first sensor data in the interval $[(k - 1)2^c, k2^c]_0$ is $(k - 1)2^c$. Since the relative position of $S_{c,k}$ is $(I\text{-bit position}(S_{c,k}))_{10} + 1$, its absolute position i is $(k - 1)2^c + (I\text{-bit position}(S_{c,k}))_{10} + 1$. \square

Appendix L. Proof Lemma 5

Proof. Given B_p , in the definition, we can get the absolute position i' and j' for $S_{c,i}$ and $S_{c,j}$, respectively, using B_p by the theorem. Moreover, we can get the absolute position p' and q' for $S_{c,p}$ and $S_{c,q}$ in the same way. In fact, the absolute positions of i' , j' , p' and q' are

$$\begin{aligned} i' &= (i - 1)2^c + (\text{BitSubstring}(B_p, c, i))_{10}, \\ j' &= (j - 1)2^c + (\text{BitSubstring}(B_p, c, j))_{10}, \\ p' &= (p - 1)2^c + (\text{BitSubstring}(B_p, c, p))_{10}, \\ q' &= (q - 1)2^c + (\text{BitSubstring}(B_p, c, q))_{10}. \end{aligned}$$

Meanwhile, line equations $\overline{S_{c,p}S_{c,j}}$ and $\overline{S_{c,i}S_{c,q}}$ by using absolute positions are as follows:

$$\begin{aligned}\overline{S_{c,p}S_{c,j}} &= \overline{S_{0,p'}S_{0,j'}} = \frac{S_{0,j'} - S_{0,p'}}{j' - p'}(x - p') + S_{0,p'} = \frac{S_{c,j} - S_{c,p}}{j' - p'}(x - p') + S_{c,p} \text{ and} \\ \overline{S_{c,i}S_{c,q}} &= \overline{S_{0,i'}S_{0,q'}} = \frac{S_{0,i'} - S_{0,q'}}{i' - q'}(x - q') + S_{0,q'} = \frac{S_{c,i} - S_{c,q}}{i' - q'}(x - q') + S_{c,q},\end{aligned}\quad (\text{A5})$$

since $S_{c,i} = S_{0,i'}$, $S_{c,j} = S_{0,j'}$, $S_{c,p} = S_{0,p'}$ and $S_{c,q} = S_{0,q'}$.

Since in Figure 7a,b are the points q' and p' at on the line $\overline{S_{0,p'}S_{0,j'}}$ and $\overline{S_{0,i'}S_{0,q'}}$, respectively, now, we can compute A and B such that

$$A = \frac{S_{c,j} - S_{c,p}}{j' - p'}(q' - p') + S_{c,p} \text{ and } B = \frac{S_{c,i} - S_{c,q}}{i' - q'}(p' - q') + S_{c,q} \quad (\text{A6})$$

By the way, since S_p and S_q is the second and the third sensor data in $I_{c,2k}$, $p = 4k + 1$ and $q = 4k + 2$, replacing them with p and q in (Equation (6)), we can get

$$\begin{aligned}A &= \frac{S_{c,j} - S_{c,4k+1}}{j' - p'}(q' - p') + S_{c,4k+1} \\ B &= \frac{S_{c,i} - S_{c,4k+2}}{i' - q'}(p' - q') + S_{c,4k+2}\end{aligned}$$

□

References

1. Kumar, A.G.G.; Thiyagarajan, R.; Sripriya, N. Buckshot DV—A Robust Routing Protocol for Wireless Sensor Networks with Unstable Network Topologies and Unidirectional Links. *Int. J. Sci. Res. Publ.* **2014**, *4*, 60–65.
2. Lohs, S.; Karnapke, R.; Nolte, J. Link Stability in a Wireless Sensor Network—An Experimental Study. In Proceedings of the 3rd International Conference on Sensor Systems and Software, Lisbon, Portugal, 4–5 June 2012; pp. 146–161.
3. Lu, M.; Wu, J. Utility-Based Data-Gathering in Wireless Sensor Networks with Unstable Links. In Proceedings of the 9th International Conference ICDCN, Kolkata, India, 5–8 January 2008; pp. 13–24.
4. Woo, A.; Tong, T.; Culler, D. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In Proceedings of the 1st international conference on Embedded networked sensor systems (SenSys '03), Los Angeles, CA, USA, 5–7 November 2003; pp. 14–27.
5. Arora, A.; Dutta, P.; Bapat, S.; Kulathumani, V.; Zhang, H.; Naik, V.; Mittal, V.; Cao, H.; Demirbas, M.; Gouda, M.; et al. A Line in the Sand: A Wireless Sensor Network for Target Detection, Classification, and Tracking. *Comput. Netw.* **2004**, *46*, 605–634. [[CrossRef](#)]
6. Marfiievici, R.; Murphy, A.L.; Picco, G.P.; Ossi, F.; Cagnacci, F. How Environmental Factors Impact Outdoor Wireless Sensor Networks: A Case Study. In Proceedings of the IEEE 10th International Conference on Mobile Ad-Hoc and Sensor Systems (MASS), Hangzhou, China, 14–16 October 2013; pp. 565–573.
7. Anastasi, G.; Falchi, A.; Passarella, A.; Conti, M.; Gregori, E. Performance Measurements of Motes Sensor Networks. In Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM), Venice, Italy, 4–6 October 2004; pp. 174–181.
8. Bannister, K.; Giorgetti, G.; Gupta, E.K.S. Wireless Sensor Networking for “Hot” Applications: Effects of Temperature on Signal Strength, Data Collection and Localization. In Proceedings of the 5th Workshop on Embedded Networked Sensors (HotEmNets), Charlottesville, VA, USA, 2–3 June 2008.
9. Boano, C.A.; Brown, J.; Tsiftes, N.; Roedig, U.; Voigt, T. The Impact of Temperature on Outdoor Industrial Sensor Network Applications. *IEEE Trans. Ind. Inform.* **2010**, *6*, 451–459. [[CrossRef](#)]

10. Son, D.; Krishnamachari, B.; Heidemann, J. Experimental Analysis of Concurrent Packet Transmissions in Wireless Sensor Networks. In Proceedings of the 4th Conference on Embedded Networked Sensor Systems (SenSys), Boulder, CO, USA, 31 October–3 November 2006; pp. 237–249.
11. Lin, S.; Zhou, G.; Al-Hami, M.; Wu, Y.; Stankovic, J.A.; He, T.; Wu, X.; Liu, H. Toward Stable Network Performance in Wireless Sensor Networks: A Multilevel Perspective. *ACM Trans. Sens. Netw. (TOSN)* **2015**, *11*, 42. [[CrossRef](#)]
12. Lin, S.; Zhang, J.; Zhou, G.; Gu, L.; Stankovic, J.A.; He, T. ATPC: Adaptive Transmission Power Control for Wireless Sensor Networks. In Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, Boulder, CO, USA, 1–6 November 2006.
13. Xu, L.; Xu, T. *Digital Underwater Acoustic Communications*; Elsevier Inc.: Amsterdam, Netherlands, 2017.
14. Mo, H.; Le, S.; Peng, Z.; Shi, Z.; Cui, J.H. Aqua-OS: An Operating System for Underwater Acoustic Networks. In Proceedings of the Wireless Algorithms, Systems, and Applications Proceedings—7th International Conference, Huangshan, China, 8–10 August 2012; pp. 561–573.
15. Akyildiz, I.F.; Pompili, D.; Melodia, T. Underwater Acoustic Sensor Networks: Research Challenges. *Ad Hoc Netw.* **2005**, *3*, 257–279. [[CrossRef](#)]
16. Climent, S.; Sanchez, A.; Capella, J.V.; Meratnia, N.; Serrano, J.J. Underwater Acoustic Wireless Sensor Networks: Advances and Future Trends in Physical, MAC and Routing Layers. *Sensors* **2014**, *14*, 795–833. [[CrossRef](#)] [[PubMed](#)]
17. Urick, R. *Principles of Underwater Sound*; McGraw-Hill: New York, NY, USA, 1983.
18. Milica, S. On the Relationship between Capacity and Distance in an Underwater Acoustic Communication Channel. *ACM SIGMOBILE Mob. Comput. Commun. Rev.* **2007**, *11*, 34–43.
19. Shin, D.H.; Kim, C.H. A Method for Sensor Data Compression Using Maximum/Minimum Values within Compression Interval Unit in WSN Communication Faults. *KIPS* **2015**, *22*, 301–304.
20. Shin, D.H.; Kim, C.H. Data compression method for reducing sensor data loss and error in wireless sensor networks. *J. Korea Multimedia Soc.* **2016**, *19*, 360–374. [[CrossRef](#)]
21. Shin, D.H.; Kim, C.H. A 2MC-based Framework for Sensor Data Loss Decrease in Wireless Sensor Network Failures. *J. Korea Soc. Simul.* **2016**, *25*, 31–40. [[CrossRef](#)]
22. Shin, D.H.; Kim, C.H. Considerations for On-the-spot Application of Ocean Sensor Network Technologies. *KIPS* **2015**, *22*, 351–354.
23. Madden, S.; Franklin, M.J.; Hellerstein, J.M.; Wei, H. *The Design of an Acquisitional Query Processor for Sensor Networks*; SIGMOD: San Diego, CA, USA, 2003; pp. 491–502.
24. Shin, D.H.; Kim, C.H. A Method for Storing and Recovering Sensing Data using Circular Queue in Wireless Sensor Network Communication Failures. In Proceedings of the 2014 Fall Conference of the KIPS, Anaheim, CA, USA, 20–24 September 2014; pp. 207–210.
25. Krishnamachari, L.; Estrin, D.; Wicker, S. The impact of data aggregation in wireless sensor networks. In Proceedings of the 22nd International Conference on Distributed Computing Systems Workshops, Vienna, Austria, 2–5 July 2002; pp. 575–578.
26. Al-Karaki, J.N.; Ul-Mustafa, R.; Kamal, A.E. Data aggregation and routing in Wireless Sensor Networks: Optimal and heuristic algorithms. *Comput. Netw.* **2009**, *53*, 945–960. [[CrossRef](#)]
27. Patil, N.S.; Patil, P.P.R. Data Aggregation in Wireless Sensor Network. In Proceedings of the IEEE International Conference on Computational Intelligence and Computing Research, Coimbatore, India, 29 December 2010.
28. Chu, D.; Deshpande, A.; Hellerstein, J.M.; Hong, W. Approximate Data Collection in Sensor Networks using Probabilistic Models. In Proceedings of the 22nd International Conference on Data Engineering (ICDE), Atlanta, GA, USA, 3–8 April 2006; pp. 48–59.
29. Kamal, A.R.M.; Razzaque, M.A.; Nixon, P. 2PDA: Two-phase Data Approximation in Wireless Sensor Network. In Proceedings of the 7th ACM International Workshop on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks, Bodrum, Turkey, 17–21 October 2010; pp. 1–8.
30. Rooshenas, A.; Rabiee, H.R.; Movaghar, A.; Naderi, M.Y. Reducing the Data Transmission in Wireless Sensor Networks Using the Principal Component Analysis. In Proceedings of the 6th International Conference on Intelligent Sensors, Sensor Networks and Informarion Processing (ISSNIP'10), Brisbane, Australia, 7–10 December 2010; pp. 133–138.

31. Mohamed, M.I.M.; Wu, W.; Moniri, M. Data Reduction Methods for Wireless Smart Sensors in Monitoring Water Distribution Systems. *Procedia Eng.* **2014**, *70*, 1166–1172. [[CrossRef](#)]
32. Kamala, A.R.M.; Hamid, M.A. Reliable Data Approximation in Wireless Sensor Network. *Ad Hoc Netw.* **2014**, *11*, 2470–2483. [[CrossRef](#)]
33. Morell, A.; Correa, A.; Barcelo, M.; Vicario, J.L. Data Aggregation and Principal Component Analysis in WSNs. *IEEE Trans. Wirel. Commun.* **2016**, *15*, 3908–3919. [[CrossRef](#)]
34. Yao, Y.; Gehrke, J. The Cougar Approach to In-network Query Processing in Sensor Networks. *ACM Sigmod Rec.* **2002**, *31*, 9–18. [[CrossRef](#)]
35. Madden, S.; Franklin, M.J.; Hellerstein, J.M.; Hong, W. TAG: A Tiny Aggregation Service for Ad-hoc Sensor Networks. *ACM SIGOPS Oper. Syst. Rev.* **2002**, *36*, 131–146. [[CrossRef](#)]
36. Yao, Y.; Gehrke, J. Query Processing for Sensor Networks. In Proceedings of the 2003 CIDR Conference, Asilomar, CA, USA, 5–8 January 2003; pp. 233–244.
37. Yu, W.; Le, T.N.; Xuan, D.; Zhao, W. Query Aggregation for Providing Efficient Data Services in Sensor Networks. In Proceedings of the IEEE International Conference on Mobile Ad-Hoc and Sensor Systems, Fort Lauderdale, FL, USA, 25–27 October 2004; pp. 31–40.
38. Kim, Y.S.; Park, S.H. A Query Result Merging Scheme for Providing Energy Efficiency in Underwater Sensor Networks. *Sensors* **2011**, *11*, 11833–11855. [[CrossRef](#)] [[PubMed](#)]
39. Kong, L.; Xia, M.; Liu, X.Y.; Chen, G.; Gu, Y.; Wu, M.Y.; Liu, X. Data Loss and Reconstruction in Wireless Sensor Networks. *IEEE Trans. Parallel Distrib. Syst.* **2014**, *25*, 2818–2828. [[CrossRef](#)]
40. Uk, U.S.; Kim, S.H. Data Reconstuction Scheme Using PCA in Sensor Network Environment. In Proceedings of the Institute of Control Robotics and Systems Conference, Jeonbuk, Korea, 13 December 2007; pp. 20–24.
41. Ehsan, S.; Bradford, K.; Brugger, M.; Hamdaoui, B.; Kovchegov, Y.; Johnson, D.; Louhaichi, M. Design and Analysis of Delay-Tolerant Sensor Networks for Monitoring and Tracking Free-Roaming Animals. *IEEE Trans. Wirel. Commun.* **2012**, *11*, 1220–1227. [[CrossRef](#)]
42. Fraire, J.A.; Madoery, P.G.; Finochietto, J.M. On the design and analysis of fair contact plans in predictable delay-tolerant networks. *IEEE Sens. J.* **2014**, *14*, 3874–3882. [[CrossRef](#)]
43. Bedon, H.; Miguel, C.; Alcarria, R.; Fernández, A.; Ruiz, F.J. Message Fragmentation Assessment in DTN Nanosatellite-based Sensor Networks. *Ad Hoc Netw.* **2016**, *44*, 76–89. [[CrossRef](#)]
44. Filho, J.G.; Patel, A.; Batista, B.L.A.; Junior, J.C. A Systematic Technical Survey of DTN and VDTN Routing Protocols. *Comput. Stand. Interfaces* **2016**, *48*, 139–159. [[CrossRef](#)]
45. Gehrke, J.; Madden, S. Query Processing in Sensor Networks. *Pervasive Comput.* **2004**, *3*, 46–55. [[CrossRef](#)]



© 2017 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).