TOC: Lightweight Event Tracing Using Online Compression for Networked Embedded Systems

Chenhong Cao, Wen Liu, Wei Dong, and Jiajun Bu

College of Computer Science and Technology, Zhejiang University, Zheda Road, No. 38, Hangzhou 310027, China

Correspondence should be addressed to Wei Dong; dongw@zju.edu.cn

Received 29 January 2015; Accepted 29 April 2015

Academic Editor: Chen Qian

Many trace-based diagnostic techniques have been proposed for abnormal detection and fault diagnosis in networked embedded systems such as wireless sensor networks (WSNs). Event tracing is a nontrivial task for resource-constrained embedded devices. Existing tracing approaches employ compression algorithms to reduce the trace size. However, these approaches either are inapplicable or perform poorly. In this paper, we propose TOC, a novel event tracing technique using online compression. TOC combines periodical pattern mining and efficient token assignment, effectively reducing the trace size with acceptable execution overhead. We implement TOC based on TinyOS 2.1.2 and evaluate its effectiveness by case studies in sensor network applications. Results show that TOC reduces the trace size by 52.2%, compared with LIS—a state-of-the-art event tracing method.

1. Introduction

The deep integration of embedded systems and networked systems has promoted the rapid development of networked embedded systems such as wireless sensor networks (WSNs). These systems have been successfully applied in various scientific as well as industrial domains to support numerous applications such as environment monitoring [1], structural protection [2], and ecosystem management [3].

WSNs are highly susceptible to deployment failures as they are deployed in austere environments such as volcanoes or mountains. Unexpected failures have been observed in many deployments despite thorough in-lab testing prior to deployment [4–6]. Therefore, run-time debugging tools are required to detect and diagnose these failures in the post-deployment phase. For deployed projects, an effective way to troubleshoot the root cause of failures is to trace important system events.

Recently, many run-time diagnostic tracing techniques have been proposed for WSNs that enable postmortem diagnosis [7–9]. However, the collected trace sizes by these techniques usually increase rapidly with time and the number of events, posing a significant challenge to the storage and computation capabilities of sensor platforms.

An intuitive method to mitigate the overhead of tracing is to compress the traces. However, the highly limited storage and computational resource constraints of the WSN node make it almost infeasible to directly apply existing compression approaches [10–13]. These approaches [10–13] either have high runtime computational overhead, which would probably introduce the Heisenbugs into system, or require a large trace buffer, which is not supported by current WSN node platforms.

Some recent works and our empirical results show the following observations, which could be potentially exploited for efficient trace compression on WSN nodes:

(i) A small number of events occur frequently in most cases.

(ii) The behavior of WSN applications is highly repetitive in a short time.

(iii) The repetitive patterns of WSN application remain stable and do not change much over time.

Motivated by the above observations, we propose TOC, a novel event tracing approach using online compression. The key idea of TOC is to capture the repetitive patterns for increasing the compression ratio, thus allowing efficient
compression on the limited buffer size. There are two key problems in TOC. First, TOC needs to quickly identify the runtime repetitive trace patterns from a small number of input traces. We address this problem as follows. For capturing runtime patterns, we apply the online compression scheme [14], considering that the traditional offline compression is based on source code analysis, which is unaware of runtime behaviors. For quick pattern identification, we employ the lightweight Fast Fourier Transform (FFT) method. Second, the event IDs should be properly assigned. Though the repetitive patterns can be used to increase the compression ratio, different event ID assignments will result in different compressed trace sizes. For example, if a frequent event is assigned a large ID (using more bits compared with a smaller ID), the compressed trace size would be large. To address this problem, we design a frequency aware event ID assignment scheme, which assigns smaller IDs to more frequent events. With this scheme, the compressed trace size could be minimized. With the above designs, the compression ratio could be efficiently increased.

We implement TOC based on TinyOS 2.1.2 [15] and evaluate its effectiveness by three case studies in sensor networks. Results show that TOC reduces the trace size by 52.2%, compared with LIS—the state-of-the-art event tracing method.

The contributions of this work are summarized as follows:

1. We propose a novel event tracing method, which can efficiently reduce event trace size based on runtime system behaviors.

2. We propose a lightweight method to mine the system periodical pattern accurately from the limited input trace buffer.

3. We implement our method and demonstrate its effectiveness in reducing trace size in real world sensor networks.

The rest of this paper is structured as follows. Section 2 describes the related work. Section 3 presents the motivations of this work. Section 4 introduces the design principles. Section 5 shows the evaluation results. Finally, Section 6 concludes this paper and gives directions of future works.

2. Related Work

There are numerous works focusing on the reduction of system runtime trace. We classify existing works into two main categories: static methods and dynamic methods.

2.1 Static Methods. The main idea of this category of methods is to assign each event that needs to be tracked a fixed length token by static analysis on the source code.

LIS [10] is a typical method in this category. LIS is a runtime logging framework designed specifically for WSNs debugging. It provides a language to gather runtime information efficiently by using local namespaces and bit-aligned logging. To achieve effective reduction of log size, LIS defines three kinds of tokens (global, local, and point). The global token is the mark of the start of a function and the point token is the mark of the end of a function. These two tokens constitute a function namespace scope. In such a scope, local type token can be parsed into a unique token in the whole system space. Due to the reuse of local type token, the bits of each local token, as well as global token, can be saved, and a significant reduction of trace size would be achieved, compared to the method of assigning each system event a global token. In addition, the width of local tokens can be variable according to their corresponding global scope. This variability makes the log more compact. However, when the number of events which are needed to be traced is few, the performance of LIS would degrade rapidly.

Although this kind of methods can save trace size by static analysis on source code and have no additional calculation overhead after deployment, they do not consider the runtime behaviors of WSN applications, such as sequence and frequent pattern, to further optimize the trace size.

2.2 Dynamic Methods. Techniques in this category reduce trace size by compressing trace at runtime.

One kind of methods in this category directly employs off-the-shelf compression algorithms on runtime trace. The compressing progress is performed once the trace buffer is full. Such methods include two widely used Unix utility programs: gzip [11] (LZ77) and compress (LZW) [13]. These two methods have also been used for trace compression. However, these techniques incur considerable computational and storage overhead during compression. Hence the above methods are inapplicable for WSNs due to the extreme resource constraints.

To deal with such challenges, several compression algorithms are proposed for the purpose of reducing the amount of calculation and memory consumption at runtime.

FCM [12] is a value prediction technique that can be used for compression. FCM learns a prediction table from the training set to predict the following value based on a fixed number of preceding values. SLZW [16] is a modified version of LZW for sensor nodes. SLZW is a dictionary-based compression algorithm which builds a dictionary of repetitive patterns while scanning the input. The patterns found in the input are replaced (encoded) with indices according to the dictionary. Since a pattern can be the prefix of another pattern, the pattern search continues until the longest pattern is found before encoding.

Although FCM and SLZW can save considerable calculation and storage resource compared with the direct compress way, they both have a fatal limitation. The RAM on WSNs node is very limited, so the trace which can be buffed by node is also limited in a few days. FCM and SLZW need training data to learn a model to predict value or look up encoded index during the processing of trace. When the model cannot be obtained or the generated model fails to reflect the comprehensive pattern of trace, the performance of such algorithms would drop rapidly. In some extreme cases, the compressed trace may consume more space than the original one.

A novel hybrid method has been proposed recently, called Prius [14]. Firstly, it collects enough traces from runtime WSN applications and sends the trace to PC; then, it performs offline algorithms such as FCM and LZW to learn the system
trace model based on these collected traces; finally, WSNs
node utilizes the learned trace model to perform online trace
compression. This novel method can significantly reduce the
size of runtime trace while consuming limited system RAM
resource. However, Prius still needs additional calculation
during online trace compression. What is more, Prius needs
to collect massive trace for offline analysis at the very first step,
which limits its scalability in some scenarios.

3. Motivation

In this section, we illustrate the necessity of the requirements
described in Section 1.

3.1. Online Compression. In most cases, WSN application’s
behavior is highly repeated. This repeated pattern can be
captured at runtime to further reduce the system trace size.

However, such repeated pattern cannot be obtained accu-
rately by the static analysis of source code. This is the reason
why static methods such as LIS can only achieve limited
reduction of trace size. To achieve a better performance in
terms of trace compression ratio, the running information of
applications such as periodical pattern and frequent sequence
pattern must be utilized.

In this scenario, an online compression method would be
helpful in capturing such information of WSN applications.

3.2. Mining System Pattern on Limited Training Data. In
order to obtain those patterns of WSN applications, system
traces need to be buffered in RAM for later processing.
However, RAM is a precious resource on most kinds of
WSN nodes. For example, a widely used WSN platform,
TelosB, only has 10 K RAM [17]. For this reason, the size of
training data should be limited. On the other hand, mining
system runtime pattern on limited training data is also a
challenging task. The less the training data, the more difficult
the obtaining of accurate system runtime patterns.

3.3. Lightweight Compression Algorithm. As mentioned in
Section 2, several techniques can compress system trace
effectively, but most of them either require tremendous
calculation at runtime or need extra RAM to store system
model. These additional calculation and storage overhead
would impact the operation of system. For some timing
sensitive applications such as wireless communication and
time synchronization, the delay caused by trace compression
would introduce additional bugs into system.

4. Design

In this section, we present the design of TOC. Section 4.1
presents an overview of TOC. Section 4.2 describes how TOC
obtains event log at runtime. Section 4.3 describes how TOC
discovers the periodical pattern of system behaviors by min-
ing event log. Section 4.4 demonstrates how TOC reassigns
token for each event based on their frequency in the log.

4.1. Architecture Overview of TOC. Figure 1 shows an over-
view of TOC: First, TOC collects the system event trace.
A unique token is assigned to each event in advance and
the executing event’s token is recorded to node RAM. Then,
TOC employs FFT to capture the periodical pattern of the
event sequence. Finally, TOC estimate the event frequency
accurately by calculating the occurrence of each event in
the periodical pattern. Based on these different frequencies,
TOC assigns each event a new token with variable length to
efficiently perform tracing at runtime. The assigned tokens
are stored in the token table for event recovering at PC side.

It is worth to note some of the most important features in
TOC:

(i) TOC is an online trace compression method. TOC
can adapt to various WSN applications based on their
system behaviors.

(ii) TOC only need a small set of training data to capture
the period of system behaviors.

(iii) TOC only incurs limited computational overhead
and memory consumption when mining periodic
pattern of event sequences and assigning each event
a variable-length token.

(iv) TOC do not consume additional calculation resource
at runtime once the token assignment is completed.

4.2. System Event Sequence Collection. The first step in
the TOC workflow is to obtain system event sequence.
Typically, the execution of WSN application on node is
composed of two phases: initialization phase and function-
ality phase. In most cases, the system behaviors of these two
phases are different from each other. During the initializa-
tion phase, WSN application configures relevant hardware
components such as Timer, Sensor, and Radio, as well as the
initialization of scheduler and task queue of the operating
system. Once the initialization phase is finished and the
functionality phase begins, the application will never reenter
the initialization phase again unless the WSN system reset
itself due to some particular conditions, for example, after the
reprogramming process. When WSN application works in
the functionality phase, system performs different behaviors
accordingly. For example, Blink toggles LEDs every time the
timer interrupt is fired. For another example, TestNetwork
sends its local packets or forwards other nodes’ packets to
sink node based on CTP protocol.

In most cases, the abnormal system behaviors occur dur-
ing functionality phase. Therefore, in resource-constrained
WSN node, it is of higher value to monitor the system events execution in functionality phase compared to that in initialization phase. Note that the behavior of WSN application is highly repeated during the functionality phase. In each repeated period, the event frequency can be calculated exactly and does not evolve much over time. Based on these repeated patterns of WSN application’s behavior, TOC only needs to take a few periods of event traces into consideration to perform further analysis.

In the initialization phase, TOC assigns a unique token to each system event. When the application enters the functionality phase, TOC monitors the system event trace and buffers the event sequence into node RAM. Once the collected event sequence is long enough, TOC stops event sequence collection and performs periodical pattern mining.

4.3. Periodical Pattern Mining. One of our main observations is that the invoked system event sequences of each WSN application’s execution have obvious periodicity. Based on this observation, we can estimate the event frequency accurately, which is essential to efficient token assignment. The naïve approach to estimate the event frequency is to compute the ratio of specific events count and the number of totally buffered events. This estimation can be incorrect since the limited event trace may end in an incomplete period. Thus, to find out the actual periodical pattern is of great importance. TOC employs Fourier Transform method as described below to obtain the period more precisely in order to ensure the accuracy of event frequency estimation.

Fourier Transform is a powerful tool in capturing periodical pattern of complex signal sequence and has been applied in many areas. Fourier Transform can transfer signal in time domain to frequency domain. In more specific terms, Fourier Transform can decompose a periodical time domain signal into several sine waves with different amplitudes, frequencies, and phase positions. In other words, the original signal can be got by simply combining this set of sine waves. So, the frequency of the sine wave with relatively higher amplitude can reflect the periodical pattern of the original signal. Additionally, Fourier Transform is robust enough in managing signal sequences with noise which are normal situations in WSN event sequences, such as misplacing or duplicating of some elements or subsequences.

Note that Fourier Transform is designed to analyze the frequency feature of signal in time domain. The frequency with higher amplitude of signal wave in time domain is closer to the real pattern frequency in the process of Fourier analysis. However, the original event tokens make Fourier Transform hardly capture the real periodical feature in event sequence since they indicate nothing except distinguishing themselves among others in the event token sequence. To apply the characteristic of Fourier Transform which can extract frequency features of signal in time domain efficiently for further analysis, we adopt preprocessing on the token of each event.

Algorithm 1 shows the pseudocode for event token preprocessing. The key idea is to reassign each event a new token value based on the first appearance of each event in the sequence. An example is shown in Figure 2. The figures on the left are the original event token sequence and the result of Fourier Transform on this sequence, respectively. The figures on the right are the event token sequence after the preprocessing of event ID and the result of Fourier Transform on such sequence. It is easy to find that the original event token sequence is relatively disordered compared to the processed one. What is more, the frequency scope information can hardly be obtained from Fourier Transform on the original event token sequence. On the contrary, the frequency scope information can easily be extracted from the result of Fourier Transform on the processed event token sequence. Compared to the original event token assignment, Algorithm 1 can transform event token sequence into ramp-like shape, which can be used to extract the periodical pattern more accurately by Fourier Transform. For example, the dominated frequency of this event sequence is 0.025, which means the period of this sequence is 40.

After preprocessing the event token in collected sequence, we can calculate the sequence period. Algorithm 2 shows the pseudocode for obtaining the period of event sequence by performing FFT. There is a detail worth mentioning here. Each sine wave that is decomposed from the original signal in time domain is regarded as a component. The components which have frequency that is integral multiple of real sequence period interfere with the analysis of real periodicity. To get the most likely bias-free result, we take the longest period as the system period among those components which have high ranking of amplitude.

4.4. Event Trace Encoding. Based on the event frequency estimated in the previous phase, TOC assigns each event a new token with variable length to perform efficient tracing at runtime. The basic idea is to assign shorter bits of token to more frequent event. Thus, the problem is essentially a traditional problem of compression. There are two influential entropy coding methods, Huffman coding and arithmetic coding, that can be applied to perform data compression on a given sequence.

Huffman coding is a prefix coding scheme which achieves lossless data compression. It can derive code with variable length for each event based on the estimated probability or frequency. As in other entropy encoding methods, the
**Algorithm 2:** Pseudocode for obtaining the period of event sequence.

**Input:** event token sequence after preprocessing: $S$

**Output:** period of event sequence: $P$

1. $A = \text{fft}(S)$
2. $\text{candidate} = []$
3. for $i = 1$ to $\text{length}(A) - 1$
4.   if $A[i-1] < A[i]$ and $A[i+1] < A[i]$
5.     $\text{candidate} = \text{candidate} + (A[i], i)$
6. end if
7. end for
8. $\text{candidate} = \text{top3}(\text{candidate})$
9. $P = 1/(\text{minValue}(\text{candidate})/\text{length}(A))$
10. procedure $\text{minValue}(\text{List})$
11.   return the minimum value of second element in List
12. end procedure
13. procedure $\text{top3}(\text{List})$
14.   return 3 tuples in List with the highest 3 value of first element
15. end procedure

**Figure 2:** Event sequence and result of FFT before and after event token preprocessing.
more frequent the event appears, the fewer the token bits are assigned. What is more, Huffman’s method can be efficiently implemented, finding a code in linear time corresponding to the number of input weights if these weights are sorted. However, although being optimal among methods encoding symbols separately, Huffman coding is not always optimal among all compression methods.

Another entropy coding method, arithmetic coding, can achieve the optimal result in theory, which means it can reduce more memory consumption of trace compared to Huffman coding. However, arithmetic coding introduces float calculation, which is more complicated and is more costly compared to integer calculation in Huffman coding. Additionally, arithmetic coding does not assign each event an identical token and needs calculation during the whole functional phase. Such computational overhead is not bearable in WSN applications. Thus, TOC employs Huffman coding to perform trace compression.

To make full use of the buffered data to obtain an accurate result, we calculate frequency of each event in the maximum integral period in the buffer. Then we perform Huffman coding to calculate the variable-length code of each event.

There is one detail worth mentioning here. The buffered event sequence may not necessarily contain all of the events in system. In such case, we assume that these events only occur during the initialization phase and rarely appear in the future. Hence, we assign such event a low weight in the Huffman coding procedure to obtain a better coding solution while missing no event.

We allocate 3 bytes to represent each event token and one extra flag byte to indicate whether the encoding step is performed. When entering the encoding phase, the flag is set and two of the 3 bytes hold the Huffman code of each event while the remaining one byte indicates the code length. Note that 16 bits are big enough to indicate each event because the number of system events in WSN applications is only hundreds.

When an event is invoked by WSN applications, the system first checks off the flag and then traces corresponding code based on the system state (before or after Huffman encoding).

5. Evaluation

In this section, we present an evaluation of TOC. Section 5.1 introduces the evaluation setup as well as the benchmarks we consider. Section 5.2 evaluates TOC’s overhead on the TelosB node including RAM consumption, program flash consumption, and computation time needed by periodical pattern mining and event token encoding. Section 5.3 evaluates the compression ratio of TOC compared to straightforward tracing method and the state-of-the-art method.

5.1. Benchmarks. In order to evaluate the overhead of TOC on real sensor nodes, we investigate four typical benchmarks on TelosB node with 8 MHz MSP430f1611 processor, 128 KB program size, 10 KB memory size, 1 MB external flash size, and 250 Kbps CC2420 radio.

| Benchmarks       | # of events |
|------------------|-------------|
| RadioCountToLeds | 69          |
| TestDip          | 70          |
| TestDissemination| 74          |
| TestNetwork      | 112         |

| Benchmarks     | Period |
|----------------|--------|
| RadioCountToLeds| 43     |
| TestDip        | 57     |
| TestDissemination| 46     |
| TestNetwork    | 81     |

All four benchmarks are investigated in the TinyOS 2.1.2 distribution:

(i) RadioCountToLeds: it is a microbenchmark to test the Radio and Timer component.
(ii) TestDip: TestDip employs the basic dissemination protocol DIP.
(iii) TestDissemination: TestDissemination executes the basic dissemination protocol Drip.
(iv) TestNetwork: TestNetwork uses the basic networking layers, CTP (collection), and Drip (dissemination).

We instrument the main component, LEDs, Radio or Network layer component, and Timer components in these benchmarks to generate system trace at runtime. Table 1 shows the event numbers for these four benchmarks.

5.2. Overhead

5.2.1. RAM Overhead. TOC requires storing a string of event sequence on RAM to perform periodical pattern mining. In addition, TOC needs extra RAM to perform Huffman coding and store corresponding Huffman code value of each event. We first study the event trace of the four benchmarks offline. We find that the periods of event trace at the normal level of instrumentation are in the range of 40 to 100. The detailed period information is shown in Table 2.

To capture the periodical pattern of event trace accurately while incurring RAM consumption as little as possible, we choose to buffer 256 events into RAM for further pattern mining. The RAM consumption at periodical pattern mining step is 2 K bytes due to the FFT algorithm’s input and output which are complex type (float variable for the real and imaginary component). For the sake of the FFT’s effectiveness, we also build a sine/cosine table to look up later, which consumes additional 256 bytes.

As shown in Table 1, the RAM consumption at Huffman coding step depends on the number of identical events which needs to be encoding. Because Huffman coding is a light weight entropy coding algorithm and does not introduce float calculation, it only takes a few hundred bytes in this step.
Table 3: Program flash size (bytes).

| Benchmarks          | Without TOC | With TOC |
|---------------------|-------------|----------|
| RadioCountToLeds     | 11346       | 16024    |
| TestDip             | 18792       | 23470    |
| TestDissemination   | 13190       | 17866    |
| TestNetwork         | 27098       | 31770    |

5.2.2. Program Flash Memory Consumption. TOC’s program flash memory consumption is mainly attributed to the FFT analysis, coding process, and event trace buffering. We evaluate the original and the increasing program flash memory occupation of the four selected benchmarks to figure out how TOC impacts the program flash size.

Table 3 shows the program size in flash memory of the four benchmarks before and after adding TOC module. We can easily find that TOC introduces program flash size increase of 4.7 K bytes on average. Consider the program flash of TelosB is 48 K bytes; the overhead of TOC is acceptable.

5.2.3. CPU Consumption. We evaluate the CPU consumption at TelosB node. As shown in Figure 3, the time consumption differences between the benchmarks are very low. Consider the mechanism of TOC, the main CPU consumption is attributed to periodical pattern mining for FFT analysis and the event trace encoding for Huffman coding algorithms.

Due to the fixed length of the buffered event sequence, which is 256 points, the time consumption of four benchmarks at periodical pattern mining step is approximately equivalent: 1348 ms on average. Time consumption at this step is relatively high. The reason is that FFT analysis introduces a large number of float point operations which are costly in TelosB platform. However, TOC performs periodical pattern mining only once during the whole lifetime of WSN applications. Thus, the time consumption at this step is acceptable.

The slight time consumption difference among these four benchmarks is caused by the Huffman coding. At this step, TOC performs coding algorithm on different numbers of events, which causes different computational overhead. As shown in Figure 3, time consumption of this step falls in a range of 60 ms to 78 ms.

5.3. Compression Ratio. In this section, we evaluate the performance of TOC in terms of compression ratio.

We implement four kinds of approaches to compare with TOC:

(i) Naïve tracing approach: assign each event a unique token.

(ii) LIS: utilize local namespace to reduce trace size.

(iii) TOC: adopt our method on trace which is obtained by naïve tracing approach.

(iv) TOC on local token of LIS: perform TOC method based on local token of LIS, in which every token with identical local number will be treated as the same token.

We take the naïve approach as the base line to evaluate the compression ratio of TOC compared with LIS. To evaluate the performance of LIS, we divide the events traced into two categories: global and local. The global token denotes the start of a function, which is instrumented before the first statement of a function body. Another extra trace point needs to be used for indicating the end of each function. It is inserted before each return statement. So far, the global token and the end point together define a function scope. Each trace point inside a function scope can be marked as local type and reuse the number of the local token space.

Figure 4 shows that LIS can reduce the trace size by 20% on average. This number is relatively small compared with the evaluation in [18]. This is because the sum of all trace points and the number of local type trace points are less in our evaluation cases. In such cases, the trace size can hardly get benefit from LIS mechanism. Compared to LIS, TOC can reduce the trace size aggressively. TOC can achieve an aggressive compression ratio from 35% to 55%. When we implement TOC on local tokens of LIS, TOC reduces the trace size further to 40% on average.
6. Conclusion and Future Work

We exploit the fact that the behaviors of WSN application are highly repetitive and do not evolve much over time to propose a novel online trace compression technique, called TOC. TOC combines periodical pattern mining and efficient token assignment to effectively reduce trace size while incurring acceptable runtime computational overhead. Compared with existing methods, TOC can adapt to various WSN applications online based on their system behaviors. To perform the pattern mining of WSN applications, TOC only needs a small set of training data and incurs very low calculation and memory consumption. What is more, TOC do not incur additional calculation overhead at runtime once the token assignment step is completed.

We further implement TOC based on TinyOS 2.1.2 and evaluate its effectiveness by case studies in sensor network applications. Results show that TOC can reduce the size of event trace to 39.9% and 47.8% on average compared with straightforward tracing method and the state-of-the-art methods.

TOC can effectively reduce the size of event trace while incurring acceptable system overhead. However, there are still several aspects that can be considered further to improve the performance and scalability of TOC:

(i) We assume that the behaviors of WSN applications do not vary much during the whole application lifetime, so TOC calculates the system periodical pattern and assigns each event a new token only once. But the system pattern will change when WSN applications suffer from anomaly. In this case, the compression ratio will drop down obviously because some infrequent events may be frequently executed under the abnormal state of WSNs node. To address this problem, a monitor can be added into TOC system. When the compression ratio decreases below a threshold because of the changed system behavior, TOC can do another round of system pattern mining and event token assignment according to the current system behavior.

(ii) Another issue that should be taken into consideration is how to utilize the frequent sequence information to further optimize trace size. As mentioned above, TOC can find the periodical pattern of system events and assign each event an efficient token based on their frequency. However, this method neglects the correlation between adjacent events which are considered by FCM and LZW approach. Our future research will combine the periodicity and event correlation to further optimize online trace compression.

(iii) To improve the scalability of TOC, we will explore how to apply TOC on more generic traces rather than only on event trace. Such traces include not only event token, but also the variable value and timestamp and so forth.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

Acknowledgments

This work is supported by the National Science Foundation of China under Grants nos. 61472360 and 61202402, the Fundamental Research Funds for the Central Universities, and the Research Fund for the Doctoral Program of Higher Education of China (20120101120179).

References

[1] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, “Fidelity and yield in a volcano monitoring sensor network,” in Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI’06), pp. 381–396, USENIX Association, 2006.

[2] M. Ceriotti, L. Mottola, G. P. Picco et al., “Monitoring heritage buildings with wireless sensor networks: the torre aquila deployment,” in Proceedings of the International Conference on Information Processing in Sensor Networks (IPSN ’09), pp. 277–288, April 2009.

[3] L. Mo, Y. He, Y. Liu et al., “Canopy closure estimates with GreenOrbs: sustainable sensing in the forest,” in Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems (SenSys ’09), pp. 99–112, ACM, Berkeley, Calif, USA, November 2009.

[4] W. Du, Z. Xing, M. Li, B. He, L. Chua, and H. Miao, “Optimal sensor placement and measurement of wind for water quality studies in urban reservoirs,” in Proceedings of the 13th...
[5] G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, "Fidelity and yield in a volcano monitoring sensor network," in Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI ’06), pp. 381–396, Seattle, Wash, USA, November 2006.

[6] A. Hasler, I. Talzi, J. Beutel, C. Tschudin, and S. Gruber, "Wireless sensor networks in Permafrost research—concept, requirements, implementation and challenges," in Proceedings of the 9th International Conference on Permafrost (NICOP ’08), June 2008.

[7] M. M. H. Khan, H. K. Le, H. Ahmadi, T. F. Abdelzaher, and J. Han, “Dustminer: troubleshooting interactive complexity bugs in sensor networks,” in Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems (SenSys ’08), pp. 99–112, November 2008.

[8] L. Luo, T. Hei, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic, "Achieving repeatability of asynchronous events in wireless sensor networks with EnviroLog," in Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM ’06), April 2006.

[9] V. Krunic, E. Trumpler, and R. Han, "NodeMD: diagnosing node-level faults in remote wireless sensor systems," in Proceedings of the 5th International Conference on Mobile Systems, Applications and Services (MobiSys ’07), pp. 43–56, ACM, 2007.

[10] L. Luo, T. Hei, G. Zhou, L. Gu, T. F. Abdelzaher, and J. A. Stankovic, "Achieving repeatability of asynchronous events in wireless sensor networks with EnviroLog," in Proceedings of the 25th IEEE International Conference on Computer Communications (INFOCOM ’06), April 2006.

[11] N. Tsiftes, A. Dunkels, and T. Voigt, "Efficient sensor network reprogramming through compression of executable modules," in Proceedings of the 5th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON ’08), pp. 359–367, June 2008.

[12] B. Goeman, H. Vandierendonck, and K. de Bosschere, “Differential FCM: increasing value prediction accuracy by improving table usage efficiency," in Proceedings of the 7th International Symposium on High-Performance Computer Architecture (HPCA ’01), pp. 207–216, IEEE, Monterrey, Mexico, October 2001.

[13] T. A. Welch, "A Technique for high-performance data compression," Computer, vol. 17, no. 6, pp. 8–19, 1984.

[14] V. Sundaram, P. Eugster, and X. Zhang, "Prius: Generic hybrid trace compression for wireless sensor networks," in Proceedings of the 10th ACM Conference on Embedded Networked Sensor Systems (SenSys ’12), pp. 183–196, November 2012.

[15] P. Levis, S. Madden, J. Polastre et al., "TinyOS: an operating system for sensor networks," in Ambient Intelligence, pp. 115–148, Springer, Berlin, Germany, 2005.

[16] C. M. Sadler and M. Martonosi, "Data compression algorithms for energy-constrained devices in delay tolerant networks," in Proceedings of the 4th International Conference on Embedded Networked Sensor Systems (SenSys ’06), pp. 265–278, ACM, 2006.

[17] Msp430f1611 Datasheet, http://focus.ti.com/docs/prod/folders/print/msp430f1611.html.

[18] R. Shea, M. B. Srivastava, and Y. Cho, "Optimizing bandwidth of call traces for wireless embedded systems," IEEE Embedded Systems Letters, vol. 1, no. 1, pp. 28–32, 2009.
