IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

Comparative Analysis of Outlier Detection Algorithms for Predeployment Symptom Mining Debugging of WSN Applications

Mrs. Sreedevi T.R

Assistant Professor, Computer Science Engineering, Presidency University, Bangalore, Karnataka, India

Abstract: A WSN (Wireless Sensor Network) is an ad hoc network of sensor nodes to monitor natural or man-made conditions and to cooperatively pass their data through the network to a main location. The thorough debugging of WSN applications before actual deployment can save a lot of time and manual effort. The existing WSN applications look short and simple, but they frequently encounter failures due to various software bugs. The main reason is that they are executed with a complicated event-driven concurrency model. WSN debuggers based on symptom mining are one of the efficient methods for identifying potential transient bugs in WSN applications. The design of such debuggers is based on a key observation that outliers in the normal behavioral patterns of a WSN system indicate transient bugs. The symptom mining debuggers applies a customized outlier detection algorithm to quickly identify and rank abnormal intervals. The scope of this paper is to implement the design of a symptom mining debugger and to conduct a comparative study of the various outlier detection algorithms in finding the potential transient bugs of a few WSN applications. The paper has identified the optimum features of an outlier detection algorithm to be used in a symptom mining debugger. OCSVM and Orca were selected because of their unsupervised nature and ability to deal with homogeneous multivariate continuous attributes. The merits and demerits of the selected outlier detection algorithms were also identified after performing an execution time and memory overhead analysis.

Keywords: WSN, OCSVM, Sentomist

INTRODUCTION

WSN is a distributed self-configurable network of sensors, which is derived from its parent network of MANETs(Mobile Ad hoc Network). In this network, sensor nodes are placed to monitor a multitude of environmental phenomena, i.e., habitant monitoring, wildlife monitoring, patient monitoring, industrial process monitoring, battle field monitoring and home automation, to name but a few. WSN is an effective tool for environmental data collection and monitoring [1]. But it is been noted that existing WSN applications frequently encounter failures due to various software bugs [2, 3]. Moreover software reliability is considered as the critical concern toward adopting WSNs as far as the potential industrial customers are concerned. With TinyOS [4],one of the popular WSN operating system, the sensing and forwarding functionalities of a sensor node can be realized with less than 100 lines of codes. But it will inject bugs because of its complicated concurrency model of execution.WSN applications are designed to be driven by events. The source codes of these applications may look simple and small, but they are executed with a concurrency model,which frequently introduces software bugs. These bugs will be particularly transient in nature. Such buggy logics will only be triggered by some interleaved events that bear implicit dependency, but can lead to fatal system failures. Because of its energy saving policy, a sensor generally works in an event-driven mode. An event procedure is the specifc application logic for handling an event. It is activated by its corresponding event which can be a hardware interrupt [5, 6]. For example, wireless interface chip will issue an interrupt upon receiving a packet, activating its corresponding event procedure to perform such actions as retrieving the packet content. Sensor data can be regularly sampled, activated upon by hardware timer interrupts.

During system runtime, events may occur randomly, and instances of event procedures may therefore start at any time and even interleave with each other. Software bugs in WSN applications become inevitable because of such interleaved executions. The interactions among multiple sensor nodes and multi-tasking in the latest WSN operating systems, will make such bugs unpredictable by the original system designers. These buggy logics will only be be triggered by some occasionally interleaved event-procedures that bear implicit dependency. Such bugs in WSNs can lead to fatal system failures. They are also difficult to be identified by state-of-the-art software testing tools for commercial software. One famous example is a bug in the widely used CTP(Collection Tree Protocol) in the TinyOS distribution [7]. The bug, once triggered, makes a WSN stop data reporting. Unfortunately, due to their momentary nature, the symptoms of these transient bugs are deeply hidden. So it requires intensive manual effort to identify the buggy symptoms and to correct the bug. These deeply-hidden bugs or even their symptoms can be identified only through emulation with precise timing. The identification of anomalies from the collected samples is done by an outlier detection algorithm. This paper performs a comparative analysis of the various current state of art outlier detection algorithms in the above context.

The aim of the paper is to perform a comparative analysis of the various outlier detection algorithms used in the pre-deployment symptom mining debuggers of WSNs.The scope of this work is associated with the debugging techniques of WSNs.Initially, a general survey on debugging techniques of WSNs presented. With this background work, one of the pre-

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

deployment debuggers of WSNs utilizing symptom mining technique is identified for implementation. An integrated platform was developed where various WSN applications are debugged using such debugging technique. In the implemented symptom mining debugger the performance of various outlier detection algorithms are analysed and compared in terms of execution speed and memory overhead.An integrated platform where wireless sensor applications can be efectively debugged in an emulated environment is developed. The monitor function of a WSN emulator called Avrora is overridden to provide program execution trace of a wireless sensor application. This is fed as the input of a symptom mining debugger for anatomizing into fine event intervals. The optimum features of outlier detection algorithms suitable for pre-deployment symptom mining debugging of WSN applications is explored. The chosen outlier detection algorithms, OCSVM(One Class Support Vector Machine) and Orca are implemented and they are compared in terms of their execution speed and memory overhead.

DEBUGGERS OF WSNS

WSNs employs a large number of autonomous sensor nodes known as motes which are capable of sensing and cooperatively send environmental conditions to a sink node that acts as an interface to the outside world. The development of WSNs is highly appreciated due to its monitoring capabilities in the remote areas and in human-unfriendly locations where a conventional wired infrastructure is found to have less use. The purpose of a WSN debugger is to iteratively detect and figure out the root causes of the failures which are difficult to identify with limited resources and visibility of the WSN network. The debugging tools that aid in fault detection and diagnosis systems for WSNs have been emerging out enormously out of the dissatisfaction with the current state of art. The debuggers of WSN can be classified based on their usage in application life cycle into 3 groups as shown below.

a) Pre-deployment debuggers
b) Deployment-time debuggers
c) Post-deployment debuggers

Pre-deployment debuggers are used in the concept development, initial testing and evaluation stage of an application life cycle. The pre-deployment tools that are generally used for debugging WSNs are Simulators, Emulators, Testbuds etc. Post-deployment debuggers are used for evaluation and parameter tuning in the field. The ongoing maintenance of the WSN can also be done with post-deployment tools. The various techniques used for post-deployment debuggers are record and replay, passive network analysis, active network analysis etc. Deployment-time tools are the tools which can perform the health monitoring when the network is set up for the first time in the real application field. The various techniques used are deployment time validation, deployment support networks etc. A survey[8] was made on the current state of the art debuggers of WSNs and they were classified based on their usage in the application life cycle as well as on the implementation strategy. Most of the troubleshooting support for sensor networks does not favor transient bugs, and is generally geared towards finding local programming errors. These bugs can be an incorrectly written line of code, an erroneous pointer reference, or an infinite loop. Existing tools revolve around testing, measurements, or stepping through instruction execution.

Sentomist [9] is a novel tool for identifying potential transient bugs using symptom mining technique. The Sentomist design is based on a key observation that outliers in the normal behavioural patterns of a WSN system indicate transient bugs. When this technique is used in the pre-deployment phase, the application software will be emulated and such bugs may be triggered by some occasionally interleaved events and can lead to fatal system failures. The simulated program run trace will be systematically anatomized into groups of event handling intervals by the symptom mining debugger. It then applies a customized outlier detection algorithm to quickly identify and rank abnormal intervals. This dramatically reduces the human efforts of inspection and thus greatly speeds up debugging. But the automatic localization of such bugs is not handled by Sentomist. It still needs tedious manual efforts by understanding the execution process represented in the instructions which can be an assembly or machine code.

SYSTEM ARCHITECTURE

This section covers the architecture of the entire system and it follows the design
of one of the symptom mining debugger in which the chosen outlier detection algorithms can be compared. The system design consists of four phases. In the first phase, the WSN application code to be checked for potential transient bugs will be compiled in TinyOS operating system. In the second phase an emulator with precise timing called Avrora[11] will emulate the assembly code of the compiled binary produced in the frst phase. The third phase has the Sentomist monitoring and anatomizing the program run time data produced by Avrora into event procedure instances. In the last phase an outlier detection algorithm will rank these event procedure instances based on how anomalous they are compared to normal samples. The overall design view can be shown as Figure 4.1 This phase deals with compiling of WSN application code using TinyOS. TinyOS is a free software and open source software component-based operating system and platform targeting WSNs. TinyOS is an embedded operating system written in the nesC [10] programming language as a set of cooperating tasks and processes. TinyOS programs are built out of software components, some of which present hardware abstractions. Components are connected to each other using interfaces. TinyOS provides interfaces and components for common abstractions such as packet communication, routing, sensing, actuation and storage. Nesc programming model has two components: Confguration File and Module File. Confguration fle connects interfaces used by components to interfaces provided by others. Module represents the application logic. Interface is a bi-directional multi-function interaction channel between two components.

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

TinyOS is completely non-blocking and it has only one stack. Therefore, all I/O operations that last longer than a few hundred microseconds are asynchronous and have a call back.

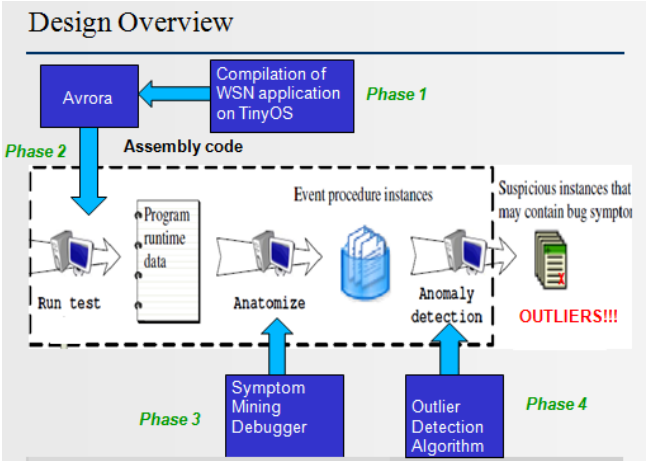

Figure 4.1: Overview of Sentomist

PHASE I : COMPILATION OF TINYOS APPLICATION

To support larger computations, TinyOS provides tasks, which are similar to a deferred procedure call. A tinyOS component can post a task, which the operating system will schedule to run later. Tasks are non pre-emptive and run in FIFO(First-In-First-Out)order. TinyOS code is statically linked with program code and is compiled into a small binary.

PHASE II : EMULATION USING AVRORA

Avrora is a cycle-accurate instruction level sensor network simulator which scales to networks of up to 10,000 nodes and performs as much as 20 times faster than previous simulators with equivalent accuracy, handling as many as 25 nodes in real-time.Simulation is an important step in the development cycle of embedded systems, allowing more detailed inspection of the dynamic execution of microcontroller programs and diagnosis of software problems before the software is deployed onto the target hardware. Avrora is a clean and open implementation motivated by this need. Avrora also provides a framework for program analysis, allowing static checking of embedded software and an infrastructure for future program analysis research. Avrora is exible, providing a java API for developing analyses and removes the need to build a large support structure to investigate program analysis. Avrora allows programs that are running in the simulation to be instrumented so that information can be collected as they execute and statistics reported after they complete. Unlike other simulators that offer a fixed set of statistics or output options that report on a program's execution, Avrora allows users of the simulator to add instrumentation to programs as they run, without modifying either the program source, program binary, or the simulator. Many instrumentation tasks can be accomplished by adding instrumentation code at particular points in the program that is triggered when that part of the program executes.

A probe is a piece of instrumentation that can be added to any instruction in the program. For example, if we are interested in recording all calls to a particular function, we can add a probe to the frst instruction in the function. The probe will fire when this instruction executes and can produce output, record the time, or collect statistics about the call that we are interested in. The simulator exposes an interface that allows users to write probe code that they can insert at various locations in their program.

PHASE III : DATA ANATOMIZATION USING SYMPTOM MINING TECHNIQUE

The input for phase III will be the monitored information from instrumented codeof Avrora. One of the WSN debugger has to anatomize this simulated run time data for finding out potential transient bugs of the application code. Based on the literature survey presented earlier in this paper, pre-deployment automated symptom mining debuggers are needed in this context. For pre-deployment symptom mining debuggers the input program run trace will be obtained by emulating the WSN application code which has to be debugged. Such debuggers will systematically anatomize the long-term execution history of an event-driven WSN system into groups of intervals called event-handling intervals. These event-handling intervals will be handling the same event type. Such a semantic partition can produce the normal behavioral patterns when the same event procedure runs. A customized outlier detection algorithm is then applied to quickly identify and rank abnormal intervals, where the system misbehaves. As a result, WSN application developers and testers can save a lot of time and manual effort by inspecting the top ranked intervals, instead of the whole execution trace of a WSN system. After an interrupt is triggered, the MCU(Micro Controller Unit) will automatically call its corresponding interrupt handler. Event procedure is the specific application logic for handling the event and it may be straightforward to implement the entire event procedure. However, an event procedure may involve extensive computational efforts and thus need to be executed for a long time. Hence, to avoid the monopolization of MCU resource and to minimize the usage of function-call stack, WSN operating systems (e.g.,TinyOS) typically implement an event procedure as two separated parts: an asynchronous interrupt handler and some deferred synchronous procedure calls, namely, tasks.

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

A task is posted by an interrupt handler or another task. The operating system maintains one single task queue, where a task is posted and executed in a FIFO manner. Moreover, tasks are executed only when there is no running interrupt handler. Finally, when there is a new interrupt arriving during the execution of a task, the task can be pre-empted by the interrupt handler. Otherwise, it will run till completion. Three rules of the concurrency model of WSN applications can be summarized as follows:

a) An interrupt handler is triggered only by its corresponding hardware interrupt
b) Interrupt handlers and tasks all run to completion unless pre-empted by other interrupt handlers
c) Tasks are posted by interrupt handlers or other tasks and executed in an FIFO

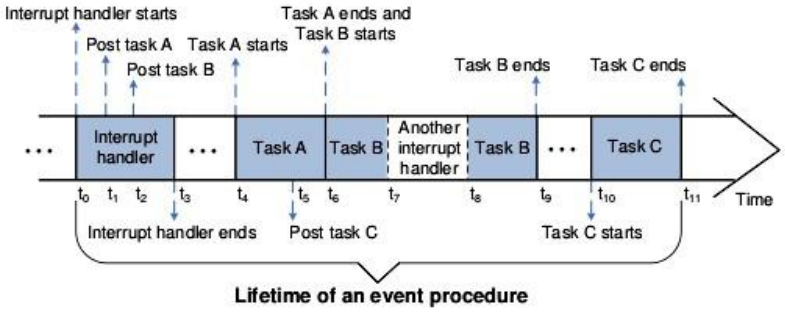

Figure 4.2: Lifetime of an Event Procedure[9]

Figure 4.2 demonstrates an example of how a typical event procedure instance runs in a sensor node. As explained before, this event procedure instance has an asynchronous interrupt handler and some deferred synchronous procedure calls: A, B, and C. The interrupt handler is activated at time t0 by its corresponding hardware interrupt event. Before it exits at t3 , it posts two tasks A and B at t1 and t2 , respectively. Later, after the MCU finishes performing the tasks posted previously by other event procedures, task A will be executed at t4. It defers some application logic by posting another task C at t5. After task A ends at t6, task B runs, during which a new interrupt handler preempts it at t7. After the preempting interrupt handler exits at t8, task B continues till completion. Task C is then the last task to be executed. Hence, the event procedure instance starts at time t0 and ends at time t11, during which its logic is executed in the time intervals denoted by the shaded areas in the figure. Note that different event procedure instances for the same event type may not run in the same pattern. For example, task B in another instance may not be interrupted. The tasks of an event procedure instance include those posted by its corresponding interrupt handler and those posted by other tasks of the same event procedure instance Sentomist, one of the WSN pre-deployment symptom mining debugger, detects potential transient bugs of WSN application code in 3 steps.

a)Decompose the program runtime into a set of time intervals
b) Selection of good attributes to feature each interval
c)Selection of generic outlier detection algorithm

Decomposition of Program Runtime Trace

Although an event procedure instance always starts with an interrupt handler, when it terminates is not straightforward to identify. It is hard to tell how many tasks an event procedure instance may post, when a task is posted, and when it is executed. Moreover, the complicated interleaving executions of event procedure instances make it more difficult to track task posting and executions. A generic method has to determine when an event procedure ends. An event-handling interval is the lifetime of an event procedure instance. It starts at the entry of its corresponding interrupt handler. It ends when its last task has been executed if the interrupt handler posts tasks; otherwise, it ends when the interrupt handler exits. For the example shown in Figure 4.2, the corresponding event-handling interval is that from t0 to t11. So the WSN application concurrency model causes complicated interleaving executions of event procedures.

Featuring Event Procedure Instances

Given an event-handling interval when an event procedure instance runs, a set of attributes to feature the behaviour of the WSN application during this interval has to be determined. The behaviour of the WSN application during such an interval can be called as a sample. This quantifcation of the samples will make the system more tractable for automatic bug symptom mining. Hence, one instant requirement is that normal samples should still be distinguishable from abnormal samples where bugs are triggered via this abstraction. In other words, the abstraction must not eliminate the differences between normal samples and abnormal ones. Sentomist adopts instruction counter as a metric to feature a sample or an event procedure instance. Instruction is a sentence of machine codes denoting a single operation of the MCU. An instruction counter is a feature of a sample which represents a vector of N elements, where N is the total number of instructions of the programs execution trace. The ith element of the vector denotes the execution number of the ith instruction during the interval. The selection of the outlier detection algorithm is dealt in phase IV.

Phase IV : Outlier Detection

In the final phase a customized outlier detection algorithm is applied to quickly identify and rank abnormal event procedure instances. The different aspects of an anomaly detection problem is analyzed and have identified the features required for an

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

optimum algorithm for finding the anomalous event procedure instances. These optimum features are explained below and summarized in Figure 4.3.

Feature	Classification	Applicability
Number of Attributes	Univariate	No
	Homogeneous Multivariate	Yes
	Heterogeneous Multivariate	No
Type of Attribute	Binary	No
	Categorical	No
	Continuous	Yes
Relationship present among data	Point Data	Yes
	Sequence data	No
	Spatial data	No
	Graph data	No
Type of Anomaly	Point	Yes
	Contextual Anomalies	No
	Collective Anomalies	No
Availability of Labels	Supervised	No
	Semi-supervised	No
	Unsupervised	Yes
Output of Anomaly detection	Score	Yes
	Label	No

Figure 4.3: Optimum Features of Outlier Detection Algorithms[18]

Nature of the input data

A key aspect of any anomaly detection technique is the nature of the input data. Input is generally a collection of data instances. Each data instance can be described using a set of attributes which are the characteristics common in all of the event procedure instances. The attributes can be of diferent types such as binary, categorical, or continuous. A data instance is called univariate if each of the data instance consist of only one attribute. If there are multiple attributes to represent each instance it is known as multivariate. In the case of multivariate data instances, all attributes might be of same type or might be a mixture of different data types. The former is marked as homogeneous and latter as heterogeneous in Figure 4.3.Input data can also be categorised based on the relationship present among data instances. Most of the existing anomaly detection techniques deal with record data or point data, in which no relationship is assumed among the data instances. In general, data instances can be related to each other. Some examples are sequence data, spatial data, and graph data. In sequence data, the data instances are linearly ordered, for example, time-series data, genome sequences, and protein sequences. In spatial data, each data instance is related to its neighboring instances, for example, vehicular traffic data, and ecological data. When the spatial data has a temporal component it is referred to as spatio-temporal data. Climate data is an example for spatio-temporal data. In graph data, data instances are represented as vertices in a graph and are connected to other vertices with edges.

Availability of Labels

The next feature of an outlier detection algorithm is availability of labels. The labels associated with a data instance denote whether that instance is normal or anomalous. It should be noted that obtaining labeled data that is accurate as well as representative of all types of behaviors, is often prohibitively expensive. Labeling is often done manually by a human expert and hence substantial effort is required to obtain the labeled training data set. Typically, getting a labeled set of anomalous data instances that covers all possible type of anomalous behavior is more difficult than getting labels for normal behavior. Based on the extent to which the labels are available, anomaly detection techniques can operate in three modes: Supervised, Semi supervised and Unsupervised. Supervised Anomaly Detection techniques trained in supervised mode assume the availability of a training data set that has labeled instances for normal as well as anomaly classes. A typical approach in such cases is to build a predictive model for normal vs anomaly classes. Any unseen data instance is compared against the model to determine which class it belongs to. There are two major issues that arise in supervised anomaly detection. First, the anomalous instances are far fewer compared to the normal instances in the training data. Techniques that operate in a semi supervised mode, assume that the training data has labeled instances only for the normal class. Since they do not require labels for the anomaly class, they are more widely applicable than supervised techniques. For example, in spacecraft fault detection, an anomaly scenario would signify an accident, which is not easy to model. The typical approach used in such techniques is to build a model for the class corresponding to normal behavior, and use the model to identify anomalies in the test data. Techniques that operate in unsupervised mode do not require training data, and thus are most widely applicable. The techniques in this category make the implicit assumption that normal instances are far more frequent than anomalies in the test data. If this assumption is not true then such techniques suffer from high false alarm rate.

Reporting of Anomalies

Another important aspect for any anomaly detection technique is the manner in which the anomalies are reported. Typically, the outputs produced by anomaly detection techniques are one of the following two types: Scores and Labels. Scoring techniques assign an anomaly score to each instance in the test data depending on the degree to which that instance is considered an anomaly. Thus the output of such techniques is a ranked list of anomalies. An analyst may choose to either

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

analyze the top few anomalies or use a cut of threshold to select the anomalies. Labeling Techniques in this category assign a label (normal or anomalous) to each test instance. Scoring-based anomaly detection techniques allow the analyst to use a domain specific threshold to select the most relevant anomalies. Techniques that provide binary labels to the test instances do not directly allow the analysts to make such a choice, though this can be controlled indirectly through parameter choices within each technique.

In our problem scenario the input data is the collection of event procedure instances and the feature is instruction counter. The instruction counter can be considered as a collection of counter values of each instruction which is homogeneous. The algorithms identified for finding out anomalous event procedure instances are listed below.

a) OCSVM using RBF kernel
b) OCSVM using Linear kernel
c) Orca

Implementation Details

Phase I: Compilation of TinyOS Application

This phase deals with compiling of two WSN application codes using TinyOS which are explained below. One of the application is a WSN data collection application where several sensor nodes monitor temperature and report the readings to a data sink in a single hop manner. Each sensor node will collect the temperature every D seconds. When a reading is ready upon request, an ADC(Analog-to-Digital Converter) interrupt will be issued so that the program can get the reading via the ADC event procedure. After collecting three sensor readings, a node will post a task to send the three readings in a data packet to the sink. As explained in the previous chapter,the nesC application code contains a confguration file and a module file. The component having the application logic module is shown in Figure 5.1

```
event void Read.readDone(error_t result, uint16_t data) {
  if(result != SUCCESS) {
    data = 0xffff;
  }
  data = sensorreading++;
  padcmsg->readings[readingcount] = data;
  readingcount++;
  if(readingcount == NREADINGS) {
    readingcount = 0;
    padcmsg->seq = counter++;
    post prepareAndSendPacket();
  }
}

task void prepareAndSendPacket() {
  if (!busy) {
    if (call AMSend.send(AM_BROADCAST_ADDR, &pkt, sizeof(adcmsg_t)) ==
SUCCESS) {
      busy = TRUE;
    }
  }
}

event void AMSend.sendDone(message_t* msg, error_t err) {
  if (&pkt == msg) {
    busy = FALSE;
  }
}
```

Figure 5.1: Application Module

Now go to the folder where these files are saved and type Make micaz test.exe. The compiled binary form of the application code is saved in test.exe. For user readability this binary code can be converted to equivalent assembly code into fe test.od using the below command.

avr-odjdumb -hD test.exe > test.od

Phase II : Emulation Using Avrora

The front-end data acquisition approach of Sentomist is by running a WSN appli-
cation over Avrora. The input for Avrora is the od file generated from phase I along with the time of simulation, nodeid to be monitored, number of nodes etc. The debugger adds its own monitoring function to the simulation by using Simulator. Probe interface which provides Avrora probe functionality as shown in Figure 5.2.

```
public class EventMonitor implements Monitor, Simulator.Probe,
        Simulator.InterruptProbe {
    private Simulator sim = null;
    private SentomistMonitor seqRecorder = null;

    @Override
    public void fireBefore(State state, int pc) {
        seqRecorder.recordInstruction(pc);
    }
```

Figure 5.2: Use of Probes in Avrora

Phase III : Data Anatomization by Symptom Mining Debugger

The output of Avrora is fed into a symptom mining debugger and is anatomized into a set of event-handling intervals, each corresponding to an ADC event procedure instance. The pseudocode to identify an event procedure instance is shown in Figure 5.3.Instruction counter is taken as a metric to feature a sample. It is a vector of N elements, where N is the total number

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

of instructions of the programs execution trace. Each element of the vector is having the count value of that particular instruction.

```
S ← The corresponding int-reti string of the int(n) item
loc ← The index in the lifecycle sequence of the
      last reti item in S
Remove the substrings of S which are int-reti strings
P ← S      // P contains only postTask items.
loop
  if P has no postTask items
    output loc
    break loop
  else
    T ← ∅
    for each postTask item p in P
      r ← p's corresponding runTask item
      loc ← r's index in the lifecycle sequence
      Q ← The string between r and the next runTask
      Remove substrings of Q which are int-reti strings
      // Q contains only postTask items.
      T ← TQ // Concatenate T and Q.
    end for
    P ← T
  end if
end loop
```

Figure 5.3: Lifetime of Event Sequence

Phase IV : Anomaly Detection

The symptom mining debugger then feeds the instruction counter vector of anatomized event procedure instances for anomaly detection. As per the design the anomaly detection is carried out using 3 outlier detection algorithms and their performances are compared.

1) OCSVM

SVM is a statistics-based method that infers how two classes of points are different from each other [12]. The input of an SVM is a set of points in a d-dimensional space R, where each point is designated to either one of the two classes. The algorithm finds a hyperplane that best separates these points into the two different classes. The hyperplane is considered as the boundary of the two classes. With such a boundary, any unlabeled sample can then be labeled to a class according to which side of the boundary it locates. In the problem settings, we do not have two sets of labeled samples. On the contrary, what we have is a set of unlabeled ones. One-class SVM therefore assume that all input samples belong to one class, i.e., the normal class, which however contains some misclassified ones. Also consider that there is a virtual outlier class, which naturally contains the origin of R and some samples that are misclassified to the normal class. We can then apply SVM to find a boundary to separate these two classes. Such a variant of SVM is called one-class SVM [13].Consequently, if a sample is on the normal side, the closer it is to the boundary, the more suspicious it is as an outlier. Otherwise, the farther it is away from the boundary, the more certain it is as an outlier. Considering that the distance between a sample to this boundary is positive if the sample is on the normal side, and negative otherwise, we can then take such a distance as a score to rank the samples. The lower the score of a sample is, the more possible that the sample should contain bug symptoms. LIBSVM [14] is a library for Support Vector Machines and can be used for implementing one-class SVM. The LIBSVM package has a Main directory with core C/C++ programs and sample data. It contains fles for implementing training and testing algorithms. LIBSVM also has a tool sub-directory which includes tools for checking data format and for selecting SVM parameters. It also has sub-directories which contain pre-built binary files and interfaces to other languages/software. The steps followed for implementation of OCSVM using LIBSVM is shown below.

Data Preprocessing

LIBSVM accepts input in a particular format and we have to transform our data to that format which is shown below.

Input Format : [label] [index1]:[value1] [index2]:[value2] ... In our context we are assuming that all instances are normal and assign a label of 1. The indexes will be the address of the instructions and value is the counter indicating the number of occurrences of a particular instruction in that instance.
Eg. Data Instance 1 60:1 68:1 896:18 898:18 900:18 902:18 904:18

Scaling on the Data

The main advantage of scaling is to avoid attributes in greater numeric ranges dominating those in smaller numeric ranges. Another advantage is to avoid numerical difficulties during the calculation. Because kernel values usually depend on the inner products of feature vectors, e.g. the linear kernel and the polynomial kernel, large attribute values might cause numerical problems. It is recommended to scale each attribute to the range [-1; +1] or [0; 1]. The same method has to be used to scale both training and testing data.

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

Training

There is a LIBSVM command which trains a Support Vector Machine to learn the data indicated in the training file and produce a model file to save the results of the learning optimization. One-class SVM detector or model is formed from training vector sets. This model can be used later with predict command to find outliers in a test file. The syntax of training command has a parameters like svm type,kernel type etc. The parameter svmtype is given a value of 2 representing one-class SVM. Its kernel type defaults to 2 for RBF kernel. The linear kernel is also tried with 0 parameter value. The parameters for svm-train command for RBF kernel is given below.

String[] argsTrain=f"-s","2","-n",nu,trainingFileName+".scale", strFileName+".model";g

Testing

The data to be tested is also preprocessed to a vector set and one-class SVM detector is applied on this test vector set. Depending on the return value of a decision function in one-class SVM detector anomalies will be detected. With the model fle formed after traing, the testfle can be tested for anomalies using the below command. This function will return the anomaly score of each instance of the testfile. A manual inspection will be done on the top scored anomalous instances and will be compared with the normal instances. The parameters for testing command in LIBSVM is given below.

String[] argsPredict=ftestingFileName+".scale",strFileName+".model",strFileName+".output"g;

2) ORCA

Orca mines distance-based outliers [15]. That is, Orca uses the distance from a given example to its nearest neighbors to determine its unusualness. The intuition is that if there are other examples that are close to the candidate in the feature space, then the example is probably not an outlier. If the nearest examples are substantially different, then the example is likely to be an outlier. Probabilistically, one can view distance-based outliers as identifying candidates that lie at points where the nearest neighbor density estimate is small. The various definitions which can be given for distance-based outliers are given below.

a) Examples for which there are fewer than p other examples within distance d .
b) Top n examples whose distance to the kth nearest neighbor is greatest.
c) Top n examples whose average distance to the k nearest neighbors is greatest .

Orca uses a nearest-neighbor approach to unsupervised anomaly detection. It defines an anomaly to be a point whose nearest neighbors in data space are far away from it. To measure the distance between two points, it uses a weighted average of the Euclidean distance for the numerical variables and the Hamming distance for the discrete variables[16].Orca does not assume that all of the training data are nominal, and can be used to find anomalies in the training data as well as in other data sets. Orca reports the contribution of each variable to the anomaly score, which is defined to be the amount that the overall anomaly score would decrease if a given variable were given a weight of zero in the distance metric. The obvious algorithm for finding distance-based outliers compares every point with every other point, and therefore has quadratic run time. When searching for the n strongest outliers in a data set, Orca keeps track of the distance score for the nth top outlier found so far. Then, when searching for the nearest neighbors of a point, it stops the search as soon as it has found enough neighbors close enough to the point to prove that it is not one of the top n outliers. This technique is known as pruning rule to obtain near-linear-time performance, allowing it to scale to very large data sets. Steps involved in Orca are listed below.

Data Preprocessing

Just like LIBSVM, orca has its own input format and have two input files. One file lists the features along with its type while other file has entry for each data instance listing the feature values. Figure 5.4 represents the format of one of the file, featurevalues.data in which each row corresponds to an event procedure instance. So each row will have the instruction count of each of the instructions since we have chosen instruction count as the feature. If a particular instruction is not present in a sample or event procedure instance its count will be given as zero. The second fle, features.felds is shown in Figure 5.5 and it lists the memory address of each of the instruction along with its datatype. For example, for the first event procedure instance, the count of the instruction having memory address 2282 is 1 whereas the count of instruction with memory address 2284 is 3.

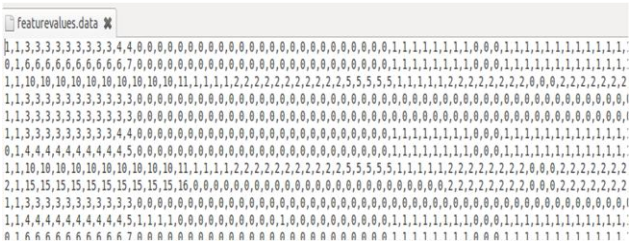

Figure 5.4: Input format of ORCA: Data Instances Figure 5.5: Input format of ORCA: Features of Data Instances

Conversion into Binary using DPrep

DPrep is a utility provided by Orca which converts a data set stored in a comma delimited text fle to the binary format required by Orca. DPrep will scale continuous features to the range [0,1] or normalize them by subtracting the mean and dividing by the standard deviation. DPrep will also randomize the order of the data set with a disk-based shuffling algorithm

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

and will automatically generates a weight file when converting a data set into the proper binary format. DPrep is called as follows
dprep data-fle felds-fle output-fle [options]

The data-fle is the name of a comma delimited text fle storing the data examples. The felds-fle specifes the fle with a description of the attributes which includes information on which fields to use. To run DPrep on our preprocessed input data the below command is used. The output file in binary format is inputorca.bin .dprep featurevalues.data features.felds inputorca.bin

Detection of Outliers

Orca detects outliers using the below command. orca inputorca.bin inputorca.bin weights Orca examines each example in the test-fle and determines if it is an outlier by comparing it with the examples in the reference-file. The weight-file contains a list of the features and their weights. The test-file and reference-file can be the same, in which case, Orca compares each example to all others except itself.

RESULTS OBTAINED

The working of pre-deployment symptom mining debugger with the selected outlier detection algorithms, OCSVM and Orca is shown below. The application which has to be tested under Sentomist was run on Avrora with the help of Avrora probes .Later Sentomist has overridden the monitoring function of Avrora and has anatomized the emulated program run trace of Avrora into event handling intervals. The event handling intervals or samples has been given as input for both the outlier detection algorithms. Detection of anomalies using the two outlier detection algorithms are shown below. By giving different parameters for svm-train command both kernel types of OCSVM is analyzed. The anomalies presented in application under testing will be listed by clicking on Analyse button as shown in Figure 6.2.

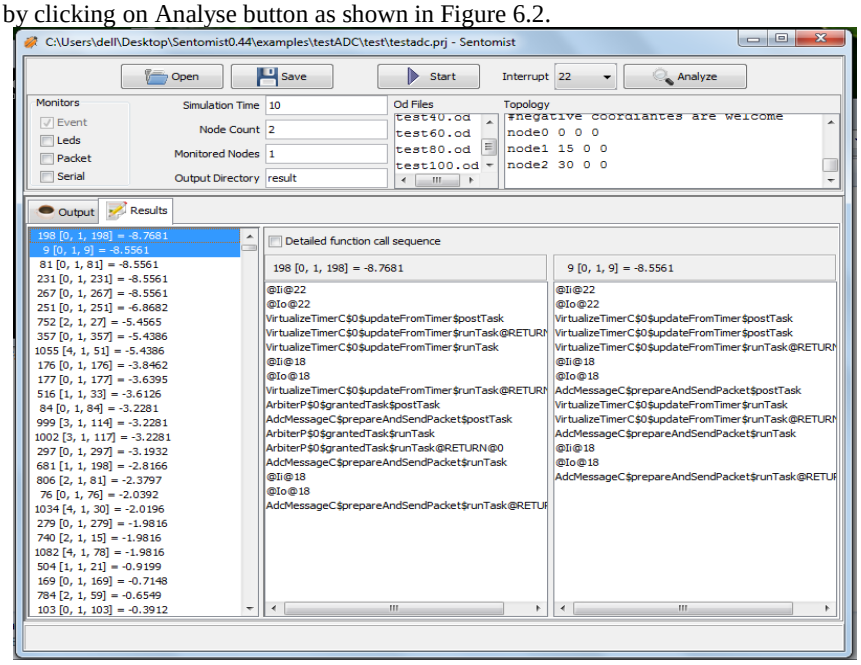

Figure 6.2: Outliers detected by OCSVM

6.1.3.2 Orca

The Orca lists its top outliers as shown in Figure 6.3. These indexes of outliers will be mapped to a function sequence which is shown in Figure 6.4.

Figure 6.3: Outliers detected by Orca

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

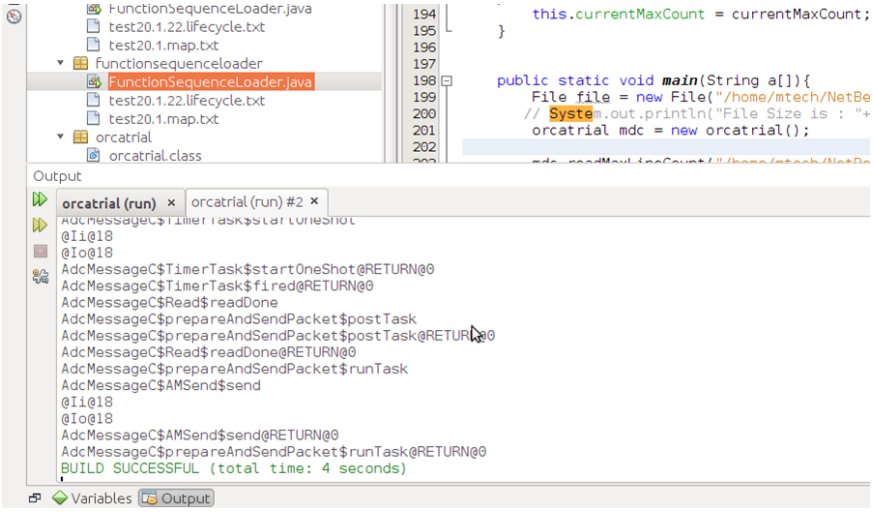

Figure 6.4: Function sequence of Outliers

The result using both outlier detection algorithms shows that the behaviors of top-ranked instances are suspicious to contain bug symptoms compared with the others. It thus directs us to a manual inspection of the system behavior for top ranked instances one by one to check the correctness of the application.

RESULT ANALYSIS AND DISCUSSION

An integrated platform where wireless sensor applications can be efectively de-bugged in an emulated environment is developed. Emulation is a popular way to check the functionality and performance of a WSN application. The system behaviors are collected based on emulation over Avrora by Sentomist. Real systems experiments cannot be relied upon to capture the system runtime behaviors because the transient nature of WSN bugs caused by the random interleaving requires a long-term system execution. It is not a cost-effective way to utilize a real system to explore a variety of system states to hit the trigger condition of a transient bug. Hence, emulation is more efficient for detection of transient bugs. Avrora is chosen since it demonstrates high fidelity to the real world. Avrora can better model interrupt preemptions and network communications with timing accuracy, which are of the most concerns in WSN application development. The debugger divides the long-term program runtime trace into the event-handling intervals also called samples. It captures the system behaviors of each sample with an instruction counter profile. Anomaly is then detected with a plug-in outlier detection algorithm applied to these samples. The symptoms of potential bugs are thus exposed for human inspections. The outlier detection algorithms are then applied to testing transient bugs in several representative real-life WSN applications. The experiments demonstrate that both the outlier detection algorithms can greatly save manual efforts in testing WSN applications. Evaluating the accuracy of unsupervised anomaly detection algorithms when applied to real-world data is a challenge, since it is generally not known how many anomalies exist in the data. In our tests, the chosen outlier detection algorithms successfully detected a potential transient bug, from each of the two WSN application code. The inputs to these algorithms were the emulated program run time data of the WSN application code. The first code to be debugged was of a WSN data collection application where several sensor nodes monitor temperature and report the readings to a data sink in a single hop manner. A potential bug found out from this code was caused by the interleaving of the event procedures triggered by internal events (those indicating the sensor readings are ready). The second WSN application code considered for debugging was a typical multi-hop packet forwarding protocol based on BlinkToRadio distributed with tinyOS. Its bug triggered occasionally by the arrivals of packets from another sensor node which is an external event. The listed outliers detected by the two algorithms may not exactly the same because of their different definition of anomaly. So it can be useful to run multiple anomaly detection algorithms on the same data set. Orca provide some sort of explanation of each anomaly in terms of the variables, but the one-class SVM algorithm does not. If it is known that all of the training data is guaranteed to be nominal, then it may be better to use an algorithm that assumes that all of the training data is nominal. Orca is relatively simple algorithm that are easy to understand, while the one-class SVM algorithm is more sophisticated. The execution time and memory overhead analysis is performed on the selected outlier detection algorithms and the results are discussed below.

Execution Time Analysis

Both the WSN application codes were emulated for different time periods, so that
the number of data instances increases with the time duration of simulation. In the first case study the application was run for 20, 30 and 40 seconds respectively while the second case study was analyzed for a simulation time of 2 ,5 and 10 seconds respectively. In our tests, the chosen outlier detection algorithms successfully detected a potential transient bug, from each of the two WSN application code. The inputs to these algorithms were the emulated program run time data of the WSN application code. The code was emulated for different time periods. The two scenarios where the number of features is greater than and lower than number of data instances respectively are captured. The size of the data set and feature set for the different trial runs for case study I and case study II are shown in Table 7.1. and Table 7.2 respectively.

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

Trial	Features	Data Instances
First	1973	483
Second	1973	1039
Third	1973	1596

Table 7.1: Statistics of Case study 1

Trial	Features	Data Instances
First	1867	2146
Second	1867	9285
Third	1867	21181

Table 7.2: Statistics of Case study 2

The execution time analysis of the two case studies are shown in Figure 7.1 and Figure 7.2 respectively. In the first case study the outlier detection algorithm Orca shows better execution time efficiency than two kernel versions of OCSVM. Among the two kernel versions, RBF kernel OCSVM takes more time for execution. In the second case study however the performance of Orca and Linear kernel OCSVM deteriorates and RBF kernel has the maximum efficiency. When we analyze the size of the data instances the RBF kernel shows poor performance when the number of features are more than number of data instances. If the number of features is large, one may not need to map data to a higher dimensional space. That is, the non-linear mapping does not improve the performance. Using the linear kernel is good enough, and one only searches for the parameter. The second case study when compared to first case study has more data instances than the features. So RBF outperforms linear kernel and Orca.

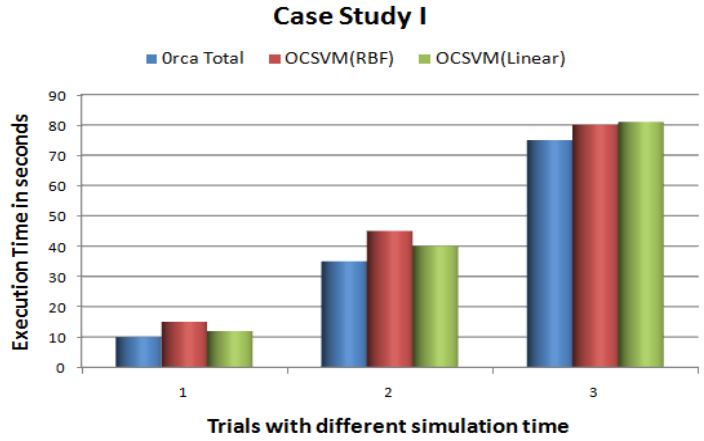

Figure 7.1: Execution Time Analysis : Case study 1

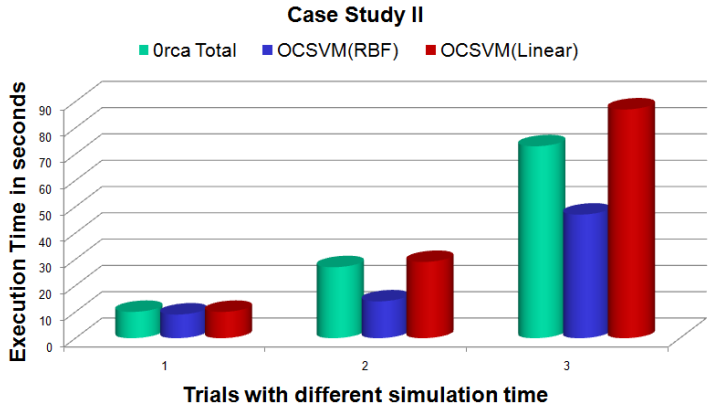

Figure 7.2: Execution Time Analysis : Case study 2

Orca uses NL algorithm [16] with a pre-processed dataset. The main idea in the NL algorithm is that for each data point in our data set, we scan the dataset and keep track of its k closest neighbors. A cut-of threshold, is also maintained that is the distance between the least outlying point discovered thus far and its kth closest neighbor. When a data points kth closest neighbur has a distance that is less than the cut-of threshold, the data point is no longer an outlier. The processing of that data point can be stopped and can proceed with the next data point. As we process more data points, the algorithm finds more extreme outliers, and the cut-of increases giving us improved pruning efficiency. Orca randomizes the dataset in linear time with constant amount of memory using a disk-based shuffling algorithm. This randomization allows the NL algorithm to process non-outlier points, which are the large majority, relatively quickly. It is found that ORCA delivers near-linear scaling behavior only when the cutoff distance can quickly converge to a large value. This can occur only when the dataset has a large number of outlying points. When the dataset consists of a mixture of a few distributions, with not many outlying points, Orcas complexity is near quadratic.

Memory Overhead Analysis

The memory overhead analysis of the chosen outlier detection algorithms is shown in Figure 7.3.

IARJSET
ISSN (Online) 2393-8021
ISSN (Print) 2394-1588
International Advanced Research Journal in Science, Engineering and Technology
ISO 3297:2007 Certified
Vol. 4, Issue 11, November 2017

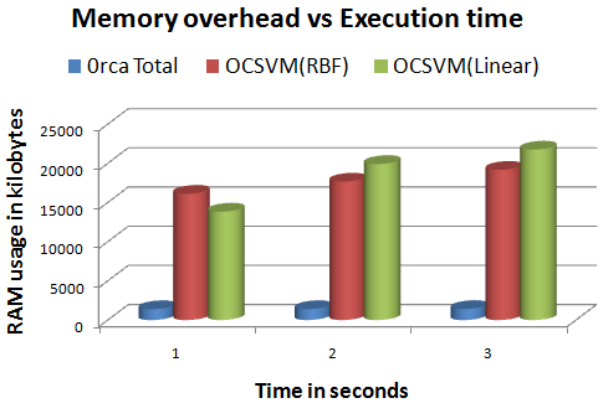

Figure 7.3: Memory overhead analysis

In this Orca has almost constant memory overhead because it only requires a limited amount of main memory to run. It does not require loading the entire database into memory. Among the kernel options of OCSVM the memory overhead of linear kernel is more than RBF kernel for large size of dataset.

CONCLUSION

This paper has its background work as the literature survey of the current-state-of art debuggers of WSNs. This work was followed by the classification of these debuggers based on their usage in application life cycle as well as on their implementation strategies. The pre-deployment debuggers using symptom mining technique were identified to find the potential transient bugs of the WSN applications in the development stage itself. More focus has to be given to pre-deployment debuggers because of their importance in saving time and manual effort by detecting bugs in the development phase itself. The various phases of a symptom mining debugger is analyzed and an optimum experimental set-up was made. Outlier detection algorithms were found to have a critical role in the design of a symptom mining debugger. The optimum features of an outlier detection algorithm which is applicable for symptom mining debuggers is identified after conducting a survey on the current state of art outlier detection algorithms. OCSVM and Orca were selected because of their unsupervised nature and ability to deal with homogeneous multivariate continuous attributes. They list the outliers along with their scores so that the developers can consider the top ranked outliers for manual inspection. The main objective of the paper was realized by performing a comparative analysis of these outlier detection algorithms in testing two WSN application codes. It is seen that pre-deployment automated symptom mining debugger can greatly save manual efforts in testing WSN applications. The merits and demerits of the selected outlier detection algorithms were also identified after performing an execution time and memory overhead analysis.

REFERENCES

[1] Kahn J. , Katz R. , Pister K. , "Next century challenges: Mobile networking for smart dust", In Proceedings of the ACM MOBICOM, Seattle, Washington, Aug. 1999, pp.271278.
[2] Langendoen K., Visser A. B. O., "Murphy loves potatoes: Experiences from a pilot sensor network deployment in precision agriculture",In Proceedings of the International Workshop on Parallel and Distributed Real-Time Systems, Apr. 2006.
[3] Werner-Allen G. , Lorincz K. , Johnson J. , Lees J. , Welsh J. , "Fidelity and yield in a volcano monitoring sensor network", in Proceedings of the USENIX OSDI, Seattle,USA, Nov. 2006, pp. 381396.
[4] TinyOS Community Forum, TinyOS: An open-source OS for the networked sensor regime, http://www.tinyos.net.
[5] Hill J. , Szewczyk R. , Woo A. , Hollar S. , Culler D. , Pister K. , "System architecture directions for networked sensors", in Proceedings of the ACM ASPLOS, 2000.
[6] Regehr J. , "Random testing of interrupt-driven software", in Proceedings of the ACM EMSOFT, Sep. 2005, pp. 290298.
[7] Gnawali O. , Fonseca R., Jamieson K. , Moss D. , Levis P. , "Collection tree protocol", in Proceedings of the ACM SENSYS, Nov. 2009, pp. 114.
[8] Sreedevi T.R., Mary Priya Sebastian, "A Classification of the Debugging Techniques of Wireless Sensor Networks", In Proceedings of International Conference on Advanced Computing and Communications, ICACC 2012, Kochi, India, August 9-11, 2012.IEEE 2012
[9] Zhou Y., Chen X. , Lyu M. , Liu J. "Sentomist: Unveiling transient sensor network bugs via symptom mining." In Proceedings of the IEEE ICDCS, pages 784-794, 2010.
[10] Gay D., Levis P., Von Behren R., Welsh M., Brewer E., Culler D. 2003. "The nesC language:A holistic approach to networked embedded systems". In Proceedings of Programming Language Design and Implementation (PLDI 2003).
[11] Titzer B. , Lee D., J. Palsberg, Avrora: Scalable sensor network simulation with precise timing", in Proceedings of the IEEE IPSN, May 2005.
[12] Vapnik V., "The Nature of Statistical Learning Theory". Springer-Verlag, 1995.
[13] Runarsson R. T., Unnthorsson, R., Johnson, T. M., Model Selection in One Class Nu-SVMS using RBF Kernels, 16th Conference on Condition Monitoring and Diagnostic Engineering Management, 2003.
[14] libsvm userguide, http://www.libsvm.com
[15] Bay S. D, Schwabacher M, "Mining Distance-Based Outliers in Near Linear Time with Randomization and a Simple Pruning Rule." Proceedings of The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.
[16] Knorr E, Nigu R, "Finding intensional knowledge of distance-based outliers." In Proceedings of the international conference on very large data bases
[17] Iverson D. L., Inductive System Health Monitoring, Proceedings of the International Conference on Artificial Intelligence,IC-AI 04,Vol.2
[18] Sreedevi T.R.,Mary Priya Sebastian, "Outlier Detection Algorithms for Pre-deployment Symptom Mining Debugging of Wireless Sensor Network Application: A Survey , International Journal of Computer Networking, Wireless and Mobile Communications (IJCNWMC) ISSN 2250-1568,2013