International Journal of Computer Networking, Wireless and Mobile Communications (IJCNWMC)
ISSN 2250-1568
Vol. 3, Issue 3, Aug 2013, 73-78

OUTLIER DETECTION ALGORITHMS FOR PREDEPLOYMENT SYMPTOM MINING DEBUGGING OF WIRELESS SENSOR NETWORK APPLICATION: A SURVEY

SREEDEVI T. R & MARY PRIYA SEBASTIAN

Department of Computer Science, Rajagiri School of Engineering & Technology, Kochi, Kerala, India

ABSTRACT

As advances in networking technology help to connect the distant and unapproachable corners of the globe, the importance of wireless sensor networks is increasing day by day. Wireless Sensor Network (WSN) applications are typically event-driven. While the source codes of these applications may look simple, they are executed with a complicated concurrency model, which frequently introduces software bugs, in particular, transient bugs. Such buggy logics may only be triggered by some occasionally interleaved events that bear implicit dependency, but can lead to fatal system failures. These deeply-hidden bugs or their symptoms can be identified by an effective outlier detection algorithm. In this paper, we provide a comprehensive survey of the different aspects of the current state of the art outlier detection systems. We also discuss the optimum features of an anomaly detection algorithm applicable for the Predeployment symptom mining debugging of a wireless sensor network application.

KEYWORDS: Wireless Sensor Network, Debugging, Outlier Detection, Symptom Mining, Transient Bugs

INTRODUCTION

Wireless Sensor Networks (WSNs) have been advocated as a promising tool for environmental data collection and monitoring [1]. Recent publications however report that existing WSN applications frequently encounter failures due to various software bugs [2], posing a major barrier to their extensive deployments. In fact, potential industrial customers have ranked software reliability as the most critical concern toward adopting WSNs. A key reason is that the simple codes are in fact executed with a complicated concurrency model. As an energy-aware embedded device, a sensor generally works in an *event-driven* mode. Specific event-handling logic (*i.e.*, *event procedure*) is activated by its corresponding event (*i.e.*, a hardware interrupt) [3]. For example, when receiving a packet, the wireless interface chip will issue an interrupt, activating its corresponding event procedure to perform such actions as retrieving the packet content. During system runtime, events may occur randomly, and instances of event procedures may therefore start at any time and even interleave with each other. Predeployment WSN debuggers based on symptom mining introduces the notion of event-handling interval to systematically anatomize the long-term execution history of an event-driven WSN system into groups of intervals. It then applies a customized outlier detection algorithm to quickly identify and rank abnormal intervals. Its design is based on a key observation that transient bugs make the behaviours of a WSN system deviate from the normal, and thus outliers (i.e., abnormal behaviours) are good indicators of potential bugs [4].

In references[5][6] long-term execution history of an event-driven WSN system is carefully anatomized into groups of intervals, during which the same event type is being handled. Such a semantic partition can exploit the similarity of system behaviours when the same event procedure runs. A customised outlier detection algorithm is used to quickly identify and rank abnormal intervals. In order to apply an outlier detection algorithm we need to identify the features for such event procedure instances. There are many straightforward candidates for featuring the samples. Examples include memory usage, number of calls to a specific function and number of packets transferred. However, most of these attributes

are only suitable for specific applications. Based on these considerations, the attributes should be adequately chosen so that they are easy to quantify and suitable for generic WSN applications.

Anomaly detection refers to the problem of finding patterns in data that do not conform to expected behavior[7] These nonconforming patterns are often referred to as anomalies, outliers, discordant observations, exceptions, aberrations, surprises, peculiarities, or contaminants in different application domains. Of these, anomalies and outliers are two terms used most commonly in the context of anomaly detection; sometimes interchangeably. Anomaly detection finds extensive use in a wide variety of applications such as fraud detection for credit cards, insurance, or health care, intrusion detection for cyber-security, fault detection in safety critical systems, and military surveillance for enemy activities. Another prominent application area for outlier detection algorithm is in symptom mining debugging of WSN application. These algorithms will seek to find portions of a simulated WSN application run time data that are somehow different from the rest of the data set .The potential bugs of the application code can thus be identified in the pre-deployment stage itself.

The ability to detect anomalies in the emulated program run time data of a WSN application is important for at least two reasons. First, detecting anomalies in pre-deployment time can be helpful in making an efficient application. Second, it will be very difficult to debug and change the embedded software after deploying it in the real area to be monitored .This article mainly focuses on identification of ideal features of an outlier detection algorithm for pre-deployment symptom mining debugging of WSN application code. An analysis on the different aspects of an anomaly detection problem is done in section II. Based on this analysis a discussion on the ideal features of an outlier detection algorithm for discovering the hidden transient bugs in a Tiny OS application is done in section III. A table which summarizes this discussion is also listed in the above section. Section IV concludes the article.

DIFFERENT ASPECTS OF AN ANOMALY DETECTION PROBLEM

This section identifies and discusses the different aspects of anomaly detection algorithm. The selection of an outlier detection algorithm is determined by several different factors such as the nature of the input data, the availability or unavailability of labels as well as the constraints and requirements induced by the application domain. The features are as listed below.

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 (also referred as object, record, point, vector, pattern, event, case, sample, observation, or entity). Each data instance can be described using a set of attributes (also referred to as variable, characteristic, feature, field, or dimension). The attributes can be of different types such as binary, categorical, or continuous. Each data instance might consist of only one attribute (univariate) or multiple attributes (multivariate). In the case of multivariate data instances, all attributes might be of same type or might be a mixture of different data types. Input data can also be categorised based on the relationship present among data instances [7]. 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 neighbouring instances, for example,vehicular traffic data, and ecological data. When the spatial data has a temporal (sequential) component it is referred to as spatio-temporal data, for example, climate 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 labelled data that is accurate as well as representative of all types of behaviours, is often prohibitively expensive. Labelling is often done manually by a human expert and hence substantial effort is required to obtain the labelled training data set. Typically, getting a labelled set of anomalous data instances that covers all possible type of anomalous behavior is more difficult than getting labels for normal behavior. Moreover, the anomalous behavior is often dynamic in nature; for example, new types of anomalies might arise, for which there is no labelled training data. In certain cases, such as air traffic safety, anomalous instances would translate to catastrophic events, and hence are very rare. Based on the extent to which the labels are available, anomaly detection techniques can operate in one of the following three modes:

- Supervised Anomaly Detection
- Un Supervised Anomaly Detection
- SemiSupervised Anomaly Detection

Supervised Anomaly Detection techniques trained in supervised mode assume the availability of a training data set that has labelled 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 semisupervised mode, assume that the training data has labelled 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 analyze the top few anomalies or use a cut off threshold to select the anomalies. Labelling 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.

DISCUSSIONS

Outliers might be introduced in the data for a variety of reasons like malicious activity, instrumentation error, change in the environment and human error. Outliers resulting from a malicious activity may be due to factors such as

insurance or credit card or telecom fraud, a cyber intrusion, a terrorist activity etc. Instrumentation error has its sources such as defects in components of machines or wear and tear or defects in the software. The outliers generated in the symptom mining of WSN application code can be classified into this category. Outliers can also be generated due to change in the environment such as a climate change, a new buying pattern among consumers, mutation in genes etc. All of the reasons have a common characteristic that they are *interesting* to the analyst. The interestingness" or real life relevance of outliers is a key Feature of outlier detection and distinguishes it from noise removal.

In our context we have a data set of event procedure instances corresponding to a particular event and each instance is featured with attribute, instruction counter which is a vector of N elements, where N is the total number of instructions of the program's corresponding machine codes. The i^{th} element of the vector denotes the execution number of the ^{ith} instruction during the interval type .The instruction counter is not the sole option to feature event procedure instances. The straightforward candidates for featuring the samples include memory usage, number of calls to a specific function, sequence of function calls, and number of packets transferred. However, most of these attributes are only suitable for specific applications. Another relevant approach is to feature the event procedure instances with number of function invocations. It is been observed that TinyOS applications are generally not designed to perform complex data computation with many looping and branching control flows due to hardware limitation [8]. Hence, showing the involved function invocations would be enough to indicate the control flows of the instance, and thus well captures its behaviours .This is also noticed by some existing approaches (e.g., [9]).

Table 1: Features and Applicability

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

This article has identified the relevant features of an outlier detection algorithm which will be applicable for pre deployment debugging of wireless sensor network application. The event procedure instances of the WSN application code to be debugged does not relate to each other and can be considered as a point data. Now based on the availability of labels we have to choose an unsupervised algorithm where we will be under the assumption that majority of the training instances will be normal. Regarding the reporting of the anomalies a labelling technique has to be opted since careful manual inspection has to be performed on the top ranked anomalies .An efficient unsupervised outlier detection algorithm can score all samples conveniently according to their distances to the boundary it finds. The ranking can instantly show how suspicious a sample is in a comparative way. So the top *k* suspicious samples has to be taken for careful manual inspection, where *k* can be flexibly chosen and its selection depends on the efforts we plan to put in manual inspections of the WSN

application. The following table summarizes the analysis of various features of outlier detection algorithms and their applicability in the context of pre-deployment debugging of wireless sensor network application.

CONCLUSIONS

The version of WSN applications is fault-prone. The existing software testing techniques cannot be applied for testing WSN systems because many WSN bugs are subtly caused by random interleaving executions of event procedures. The WSN application bugs are difficult to debug since their symptoms are transient in nature, which rarely occurs in tremendous system runtime data. It demands tremendous manual effort , to examine whether a system behaves correctly or not. Symptom mining debugging is an efficient way for testing WSN systems. Such debuggers for finding out the potential transient bugs of WSN application code divide the long-term system runtime data into event-handling intervals. It captures the system behaviours of each interval with various features like instruction counter, function call sequence etc. The features of the current state of the art outlier detection algorithms have been analysed and an ideal outlier detection algorithm features have been identified . The symptoms of potential bugs are thus exposed for human inspections. So selection of an efficient outlier detection algorithm will lead to an efficient Predeployment symptom mining debugger which will expose relevant potential bugs of the with a minimum amount of time,cpu utilization and memory overhead.

REFERENCES

1. J. Kahn, R. Katz, and K. Pister, “Next century challenges:Mobile networking for “smart dust”,” in *Proc. of the ACM MOBICOM*, Seattle, Washington, Aug. 1999, pp. 271–278.
2. K. Langendoen and A. B. O. Visser, “Murphy loves potatoes:Experiences from a pilot sensor network deployment in precision agriculture,” in Proc. of the International Workshop on Parallel and Distributed Real-Time Systems, Apr. 2006.
3. G. Werner-Allen, K. Lorincz, J. Johnson, J. Lees, and M. Welsh, “Fidelity and yield in a volcano monitoring sensor network,” in Proc. of the USENIX OSDI, Seattle, USA, Nov.2006, pp. 381–396.
4. A. Zeller, Why Programs Fail: A Guide to Systematic Debugging,2nd ed. Elsevier Science, 2009.
5. Y. Zhou, X. Chen, M. Lyu, and J. Liu. “Sentomist:Unveiling transient sensor network bugs via symptom mining”. In Proc. of the IEEE ICDCS, pages 784–794,2010.
6. Yangfan Zhou, Xinyu Chen, Michael R. Lyu, Jiangchuan Liu,”T-Morph: Revealing Buggy Behaviors of TinyOS Applications via Rule Mining and Visualization”, in Proc. of the 20th ACM International Symposium on Foundations of Software Engineering,2012
7. V. Chandola, A. Banerjee, and V. Kumar, “Anomaly detection:A survey,” ACM Computing Surveys, vol. 41, no. 3, pp. 15:1–15:58, Jul. 2009.
8. TinyOS Home Page. http://www.tinyos.net.
9. M. M. H. Khan, H. K. Le, H. Ahmadi, T. F.Abdelzaher, and J. Han. Dustminer: Troubleshooting interactive complexity bugs in sensor networks. In Proc. of the ACM SenSys, pages 99–112, 2008.