Corl8: A System for Analyzing Diagnostic Measures in Wireless Sensor Networks

Loren Klingman and Jason O. Hallstrom
School of Computing, Clemson University
Clemson, SC 29634, USA
{lklingm, jasonoh}@clemson.edu

Abstract

Researchers are deploying wireless sensor networks more than ever before. These networks comprise a large number of sensors integrated with small, low-power, wireless transceivers. They are often deployed in harsh, volatile locations, which increases their rate of packet loss and device failure. Using diagnostic metrics to debug wireless sensor networks allows for transmitting only a small amount of extra information, while still enabling performance analyses to be performed.

We present *Corl8*, a system for analyzing diagnostic traces in wireless sensor networks. Our approach relies on diagnostic data that is periodically transmitted to a network sink as part of the standard sensor payload. Corl8 provides an interactive environment for exploring correlated changes in diagnostic measures within an individual node or on a batch basis to flag interesting correlations. The system's flexibility makes it applicable for use in any wireless sensor network that transmits diagnostic information. The analysis methods are user-configurable, but we suggest settings and analyze their performance with data from five real-world deployments.

keywords: wireless sensor networks, debugging, diagnostics, correlation

1 Introduction

Researchers use wireless sensor networks to monitor the physical world, and to gather information that would otherwise be unavailable. These networks use a large number of sensors to gather data, often in harsh environments, over a large area. Each node is equipped with a small, low-power, wireless radio used to transmit data for processing and storage. These networks can be applied in a wide range of applications, including habitat monitoring [16], microclimate monitoring [17], sow monitoring [3], pipeline monitoring [9], target detection [18], coal mine structural monitoring [10], early warning systems for floods [2], and hospital patient monitoring [4]. Fault tolerance is critical in these networks because they are failure-prone, resulting from link quality variation, packet corruption, and wireless congestion. These factors are often exacerbated due to multi-hop communication between nodes and sinks [11]. The devices themselves are also subject to failure, stemming from the harsh environments in which they are deployed.

Sensor nodes present unique requirements and hardware constraints. They must be able to endure the environment, require minimal maintenance, and function with limited power in confined spaces. Equally important, they must be able to react to and recover from failures, even though they function with less storage and computational capability than typical desktop computers or even smartphones. These limitations present challenges for debugging failures because there is insufficient space on a node to store a full debugging log, such as those commonly used on desktop computers. Further, energy limitations prohibit transmission of a complete log to a central node.

In light of these limitations, it is more common for sensing systems to report a variety of simple diagnostic measures as part of their standard operation. These measures are analyzed when nodes do not report as expected. However, it is often difficult to understand what these numbers mean individually. Understanding the *relationships* among the measures is critical. For example, a large number of TCP disconnect errors could indicate any number of problems with the initiating node's hardware or software, the target system's hardware or software, or external environmental factors. Understanding that the TCP disconnect errors occur in step with low signal strength reduces the exploration space considerably.

To this end, we present an integrated system to support the analysis of diagnostic data produced by wireless sensor nodes. The system generates user-friendly graphs of diagnostic data and supports basic mining of error rate correlations, resulting in graphs

of highly correlated data. The analysis mechanism is configurable, allowing users to exclude repeated data points (e.g., no diagnostic errors) to prevent inflation of the correlation statistic. The user may also request that the data be analyzed in any number of equal width segments to support the discovery of error rates that may only be correlated during certain "data windows". We apply this system to five deployed wireless sensor networks that are part of the Intelligent River® project [7] and evaluate the system in terms of its ability to assist in reasoning about system errors.

The remainder of the paper is organized as follows. Section 2 surveys some of the most closely related work in debugging wireless sensor network applications. Section 3 presents the design and implementation of Corl8. Section 4 presents several use-case scenarios. Section 5 presents results identified using Corl8 and discusses its effectiveness. Finally, Section 6 presents a summary of our work and conclusions.

2 Related Work

Systems approaches to debugging wireless sensor networks have received considerable attention in the literature. Here we consider some of the most relevant related work.

Ramanathan et al. [13, 14] present *Sympathy*, which is designed to be deployed in tree-based, multi-hop topologies. It assigns each failure a localized source (i.e. root cause), indicating where the failure most likely originated and where the developer should begin when debugging the problem. These sources can be *self*, *path*, or *sink*. Sympathy works by monitoring network traffic and extracting flow and node metrics from messages received at the sink. This information is used to compute other diagnostic support data, such as routing tables and neighbor lists. Sympathy introduces overhead less than or equal to 31% of data traffic. Its purpose is to identify the node causing packet or data loss in a network. Our system focuses on isolating failures to specific components of a sensor node by identifying the interactions of those components through correlation of diagnostic measures. Our system's flexibility allows it to analyze any diagnostic measures periodically transmitted by a sensor network.

Khan et al. [8] present *Dustminer*, a tool for identifying bugs associated with complex component interactions in networked sensing applications. These bugs are not localized to one faulty component, but occur due to unexpected interactions. The interactions may not be repeatable, making it difficult to reconstruct the anomalous situations and localize the bugs. Dustminer examines sequences of events across nodes to identify the root cause. Since events at devices that have not yet communicated are logically independent and could have occurred in any order, the system uses non-determinism to help build sufficiently diverse training examples to recognize relevant correlations for faulty system behaviors. A front-end framework collects event logs; the back-end processes these logs using Apriori [1] to mine for frequent discriminative patterns. The success of this approach depends on logging the appropriate events. In their tests, the logs took 100-400 KB of flash memory and were not transmitted directly to the server. Corl8 relies on a count of events, rather than full sequence information. This improves efficiency, but comes at a cost; event causality is lost.

Girod et al. [5] present *EmStar*, a software environment for developing and deploying wireless sensor networks. EmStar favors ease of use and modularity over efficiency; it must be run on a multi-process "microserver" node [6], not a resource-constrained device. The *EmView* component for network visualization shows which network nodes have failed or are functioning properly. *EmView* must explicitly request updates from individual nodes, which could be disadvantageous in providing regular updates since it would require many data requests. While the approach is more efficient when the system is not being monitored, if the system crashes in such a state, there would be no data to help diagnose the cause of the failure. While robust and modular, it is not applicable to systems with limited on-board memory or processor resources. It relies on stored logs and concurrently running processes to be able to report failures after they have occurred. Our system focuses on resource-constrained devices, providing enough information prior to failure or degradation that the cause can be inferred by the programmer.

3 Design

In this section, we present the design and implementation of the Corl8 analysis system. It is designed to focus on "interesting" data, which is automatically presented as a set of scatterplots that depict how the various diagnostic measures in question vary. The system can be used to analyze a single node or all network nodes. First, we present an overview of the system, then describe each of the individual components.

3.1 Overview

An overview of the flow of diagnostic data through the Corl8 system is shown in Figure 1. Data originates at the *Sensor Nodes* and flows into the *Sensor Diagnostic*

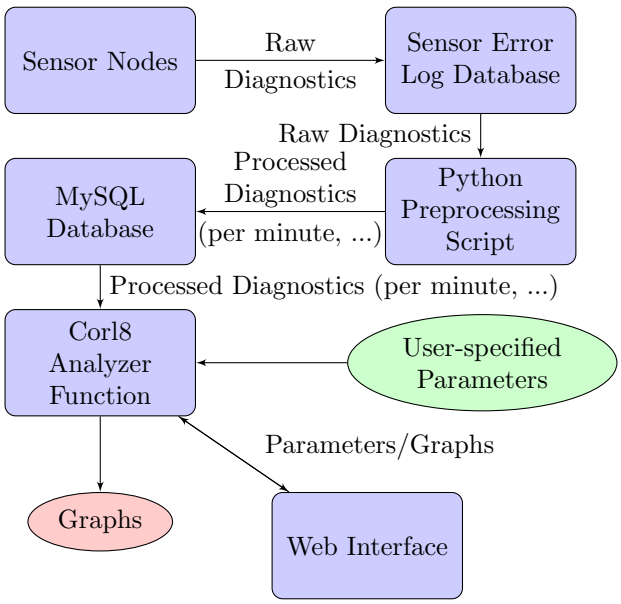

Figure 1: Diagnostic Data Flow

Log Database (in whatever format is used by the target network[1]). Next, a *Python Preprocessing Script* parses the raw data into per minute, per sensor attempt, and per transmission attempt metrics to allow analysis to be done based on other influencing factors, such as transmission attempts for radio board diagnostics or processing time. As the data is processed, the Python script inserts the results into a *MySQL Database*. The data may also be preprocessed to reduce the number of points, or to round the measures, allowing the system to flag more points as duplicates.

The central component of our framework is the *R Analyzer Function*, which reads information from the *MySQL Database* and accepts *User-specified Parameters*. The function parameters tailor various aspects of the data analysis and control how the resulting *graph sets* should be output. The output for each unique combination of (i) node, (ii) independent diagnostic measure, and (iii) dependent diagnostic measure is a *graph set*. Each graph set contains one or more graphs. The first graph represents the entire data set. If the parameters are configured such that the data is also analyzed based on divisions (equal width segments), a graph will be created for each division. The graph set is then saved as a single image. The final result of the function is a collection of graph sets.

We discuss the details of representative parameters in Section 3.4. The *R Analyzer Function* can also accept input from the *Web Interface*. In this case, the graph set resulting from the Analyzer is sent back to the web interface for the user to view. The web interface is also

[1]Our target system uses a MongoDB database.

Name	***Type***
id	int(11) - Primary Key
diagnostic	varchar(100)
device	varchar(100)
pmin	decimal(18,9)
pradio	decimal(18,9)
psample	decimal(18,9)
raw	decimal(18,9)
time	datetime

Table 1: MySQL Table Structure

written in R and is hosted using Shiny Server [15]. The interface supports many of the same parameters, but the inputs are specified in an interactive manner.

3.2 MySQL Database

The database is designed to import diagnostic measures from any number of nodes, and from any number of diagnostic categories. The structure is shown in Table 1; it contains eight columns. The `id` column is the (generated) primary key. The `diagnostic` column captures the name of a diagnostic measure. In our sensor system, we use names such as *appDiagnostics-sampleAttempts* and *gm862Diagnostics-wakeErrors*. The `device` column records the name of the device being observed (e.g., *srb_1*, *srb_2*, *bar_1*, ...). The data columns (`pmin`, `pradio`, `psample`, `raw`) contain the values of the diagnostic measures in several formats: count per minute, count per radio attempt, count per sample attempt, and the raw count reported in the diagnostic message. The number of data columns can vary among applications. In some cases, it may not make sense to use per minute, transmission, or sample error rates; the raw value may be more appropriate. For example, signal strength and battery voltage will only make sense when interpreted using the raw values. A user might also wish to create a data column containing errors per event detected. The `time` column records the data timestamp.

3.3 Populating MySQL

We assume sensor nodes send cumulative diagnostics since the last reboot. Given that the counts are set to zero after each reboot, and then increase monotonically, the system detects reboots. It also excludes counts from prior reports before dividing the diagnostic measures in the current report to compute counts per minute, per sensor reading attempt, per transmission attempt, etc. The system inserts one row for each diagnostic reported by each node. After all reports for a device have been processed, the system advances to the next device. When processing is complete, the database

Parameter	*Description*
device	The device to consider. If `all`, all devices will be used.
DB Parameters	dbName, dbHost, dbUser, dbPassword, dbTable
diagnostics	The vector of diagnostics to analyze, or NULL to use all.
diagCol1, diagCol2	The column to use for the first/ second diagnostic parameter.
threshold	The minimum number of points required to run the analysis.
round	The number of decimal places, or NULL to use data as-is.
maxDuplicates	The maximum number of duplicates of a single point allowed in the test.
minRS	The minimum r-squared value required to consider a graph as correlated.
minDate, maxDate	The start/end date in the form (YYYY-MM-DD HH:MM:SS).
divisions	Number of segments to divide the data into.
divisionThreshold	The minimum number of points in a division to run the analysis.
requireSubCor	Require a division be correlated to output the graph set.
Image Output Parameters	folder, nCol, graphWidth, graphHeight

Table 2: Analyzer Parameters

contains all the diagnostic data reported by the sensor network.

3.4 Analyzer Function

The Analyzer Function is responsible for determining which data to display to the user for further analysis. It is written in R [12] and controlled by the parameters provided by the user. It acts on the processed data in the MySQL table. A change in the parameters can make a significant difference in the false positive/negative rate, as well as the volume of data output for further analysis. As a result, it is important to understand the various parameters. A subset of the available parameters is summarized in Table 2.

4 Use-case Scenarios

We now present two use-cases to demonstrate the utility of Corl8. The first illustrates the use of batch-based analysis to discover possible errors. The second illustrates Corl8's ability to support the investigation of potential flaws using the web interface.

4.1 Batch Analysis

Corl8's batch analysis mode allows users to analyze all of the available diagnostic data to search for correlated error rates. Running the system in this manner results in the production of a collection of graph set images requiring further analysis by the user. In our example, we use data from the Intelligent River® project. In total, we use over 3 million diagnostic measures from 36 nodes.

To remove duplicate data points, we use the default rounding setting to round our data to the nearest hundredth. The running time of Corl8 in batch mode was approximately 9 hours. The machine was equipped with 8 GB of RAM and an Intel Core2 Quad CPU running at 2.66 GHz. However, because Corl8 uses a single thread, it is unable to take advantage of the 4 cores. By using multiple processes and instructing each instance of Corl8 to process a single device, this time could be significantly reduced. After the analysis completed, Corl8 flagged approximately 590 graph sets for further investigation.

Some of these graphs show expected correlations (e.g., up-time versus sampling attempts, see Section 5). Other graphs represent sources of concern (see Section 5). The default analysis parameters met our goal of providing a reasonable number of useful graphs as output. However, one common change is to increase or decrease the minimum R-squared value used to identify a correlated graph, which can further balance the number of graph sets with the false positive rate.

4.2 Interactive Analysis

Corl8's interactive analysis mode allows for faster results because the system produces only one graph set at a time in response to a user's request. If a specific interaction is suspected, developers can quickly view a graph of the diagnostic measures in question. They can then modify the search parameters as needed to investigate whether the issue started or stopped at a specific time, and to see if viewing the data based on per minute, per radio attempt, or per sample attempt yields better information. For example, a developer might want to compare log errors per sample attempt versus log errors per radio attempt to see if one is more related than the other. In general, the web interface allows user's to interactively control every salient parameter of the Analyzer Function. Adjusting values interactively provides a quick way to increase or decrease the various thresholds and allowances, and to immediately see how a specific graph is affected. This helps developers

determine the optimal settings to use when running batch analysis.

5 Results

Using Corl8 to analyze our Intelligent River® deployments throughout South Carolina, we were able to identify several interesting data trends, which helped us to understand performance issues. In the following sections, we present three of the most interesting results.

The graph set shown in Figure 2 represents diagnostic data from a node deployed at the Baruch Institute in Georgetown, SC. It shows that TCP disconnect errors from our cellular board are correlated with escape errors from the same board. This correlation suggests an error in our sensor nodes. As background, to send data to the Intelligent River® server, the cellular modem must first create a connection to the server. After the connection is established, the modem automatically switches from "command mode" to "data mode." In command mode, each byte sent to the modem is interpreted as a command. In data mode, the bytes are instead relayed over the connection, which sends the data to the server. Once the connection is established, and the node has sent its data, it needs to disconnect and shutdown. To complete this task, it must first escape back into command mode. This correlation data points to an issue where the cellular modem emits a connection response, but the connection is not fully established. As a result, the connection is immediately dropped, causing the device to fall back into command mode. From there, the escape command fails because the cellular modem is already in command mode, and the disconnect command also fails because it is no longer connected. This pattern of failure increases the up-time of the cellular chip, increasing energy consumption, and decreasing device longevity. This correlation pattern helps to point to this error chain.

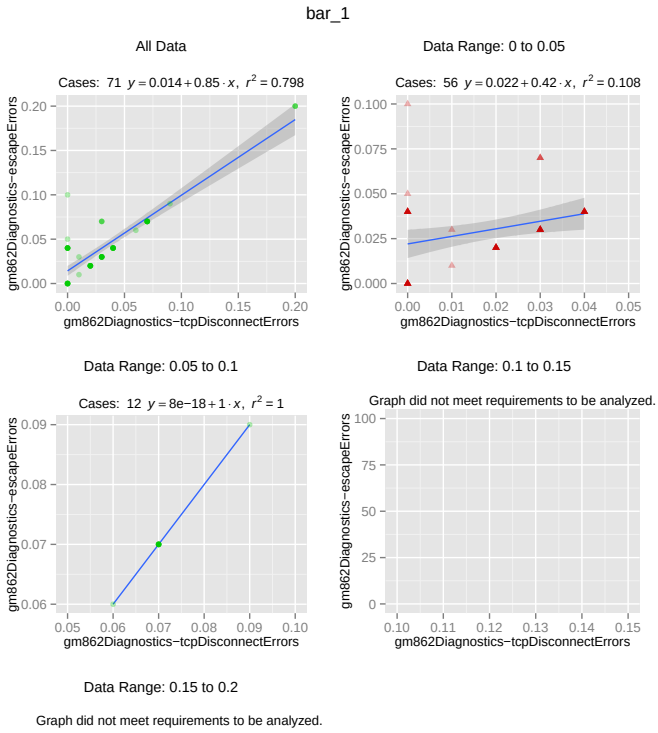

Figure 2: TCP Disconnect Errors vs Escape Errors

The graph set in Figure 3 shows the number of sample attempts performed versus the up-time of the sampling board in a device used at a conference demonstration. This graph set shows that the node was working properly. Looking closely at the graph, one outlier shows where 2 sampling attempts were made, and the ADS sampling board was up for 8.25 seconds, instead of the expected 8 seconds. If there were more outliers in the data set, it would be a cause for concern, but since there is only one outlier, it does not warrant concern.

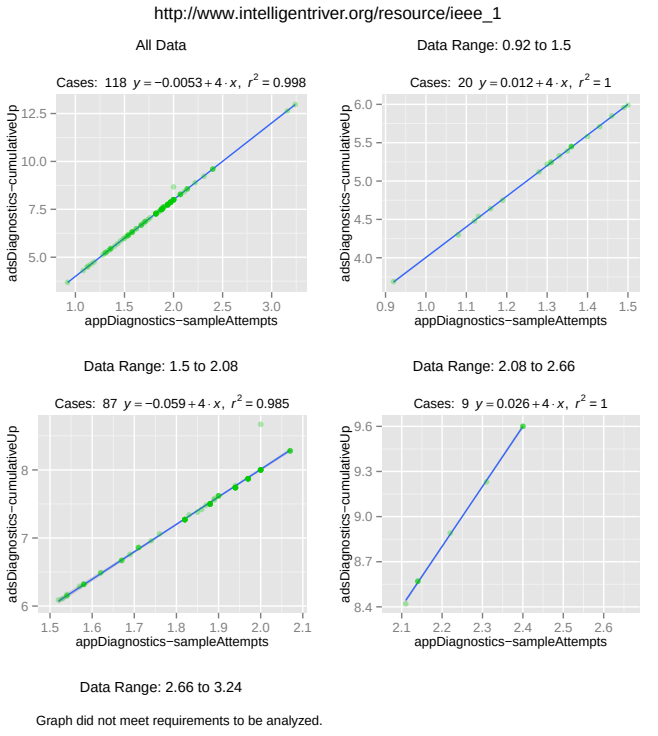

Figure 3: Sample Attempts vs Sampling Up-time

While monitoring the nodes in an on-campus deployment, we noticed that two nodes were occasionally unresponsive for long periods. Looking at the application diagnostics, we observed that a large number of sample

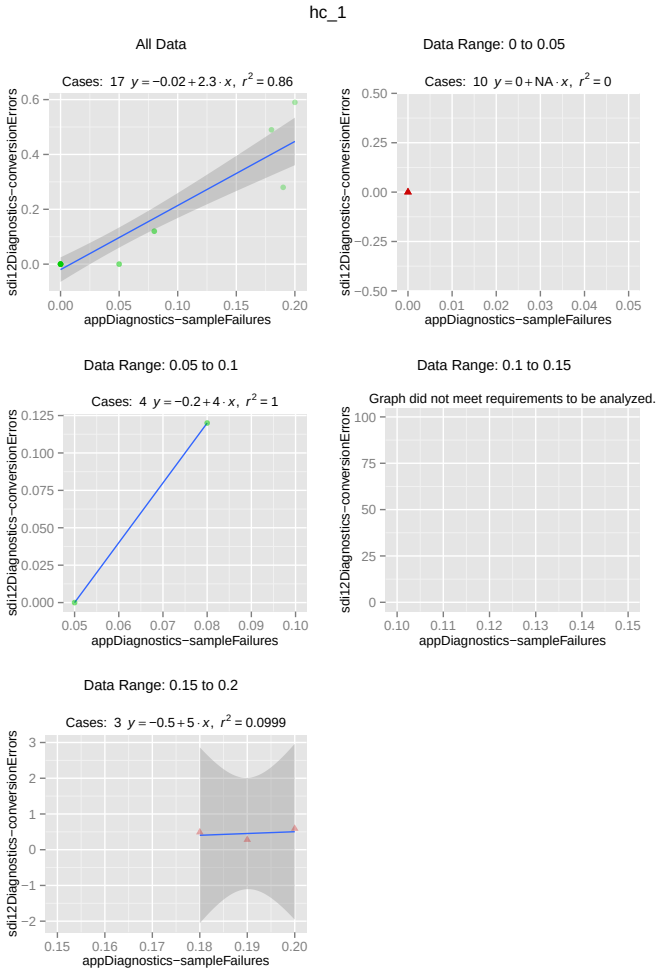

Figure 4: Sample Failures vs SDI-12 Conversion Errors

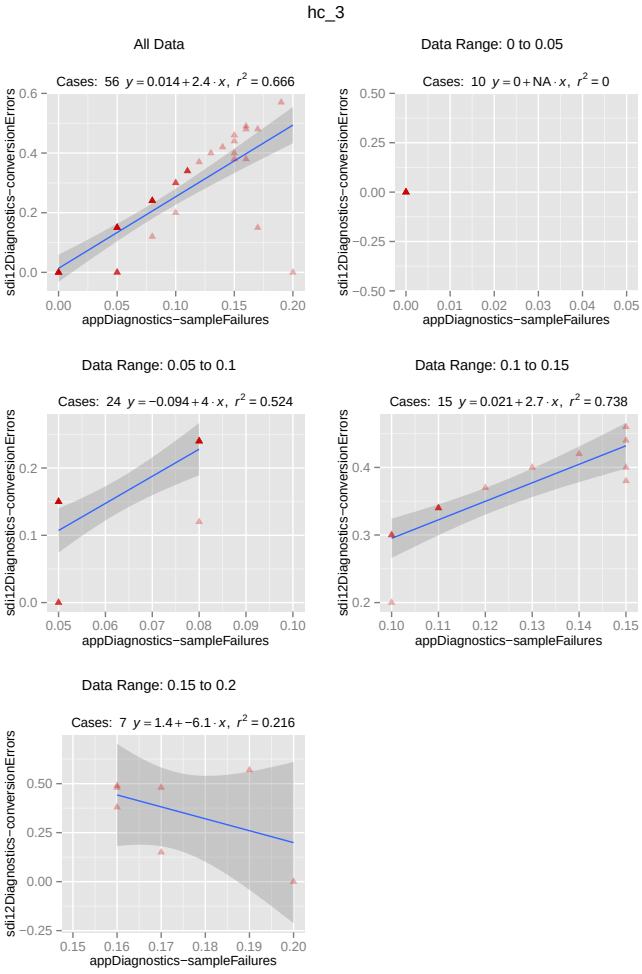

Figure 5: Sample Failures vs SDI-12 Conversion Errors

failures were occurring during these time periods. Since these sample failures were unexpected, we employed Corl8 to determine the cause of the failures. The observed failures began on February 24, 2014, so we ran a batch test using a start date (`minDate`) of `2014-02-24 00:00:00`. We also reduced the threshold, the number of points required to analyze a dataset, to 10, and the division threshold, the number of points required in any segment of the data to be analyzed, to 3 because we had only a few reports showing these issues. Corl8 automatically flagged three graphs for one of the nodes. These graphs showed that sample failures were correlated to radio attempts, conversion errors, and up-time on the transmission board.

We were able to eliminate two of the graphs (radio attempts and up-time) because they showed that when a large number of sample failures occurred, the number of radio attempts (and thus up-time) was reduced. This is an expected correlation since the nodes are configured to transmit only when they have three successful samples. The conversion errors from the SDI-12 sampling board provided more information. The associated graph set produced by Corl8 is shown in Figure 4.

It was also interesting that Corl8 flagged only two graphs for another node in the same deployment — radio attempts and up-time on the transmission board. Thus, we turned to the web interface to explore sample failures versus conversion errors. The graph set is shown in Figure 5. From this graph, we can observe that the general trend is still present, but the presence of several outliers caused Corl8 to not automatically flag the data set. The underlying cause of this error appears to be hardware damage.

We were able to identify errors through Corl8's batch mode, and further investigated the errors using the interactive mode. The time spent investigating the graphs was only a few hours to obtain useful information for addressing errors and verifying that other components were working as expected. Corl8 can significantly decrease the time required to debug

wireless sensor network systems that report diagnostic measures.

6 Conclusion

Researchers are deploying wireless sensor networks more than ever before. Large numbers of sensors integrated with small, low-power, wireless transceivers compose these networks. The harsh, volatile locations in which these devices are deployed increase their failure rate.

Corl8 helps developers determine causes of failure by identifying correlated diagnostic measures. Since hardware resources such as memory and power are often scarce, our work seeks to minimally impact the amount of data that must be stored and transmitted. We also seek to avoid add-in network protocols, which consume additional resources and may be difficult to add to existing wireless sensor networks. The flexibility of our system allows researchers with sensor networks already collecting some amount of diagnostic data to apply our system with no changes to their network.

Corl8 allows for batch mode analysis to help identify unknown faults in a system, and interactive analysis for investigating suspected faults. In our tests, Corl8 successfully found the correlated diagnostic measures to explain why nodes were seeing an unusually large number of cellular disconnect errors and assisted an exploration into why some nodes were experiencing an increased number of sampling errors. We believe Corl8 will help researchers more quickly and easily diagnose performance anomalies.

It is our hope that the reach of the Corl8 system can be expanded to include observation data. We believe this will lead to more robust diagnostic analyses, particularly when environmental factors are suspected as a potential cause for failure.

Acknowledgments

This work was supported in part by NSF grants CNS-1126344 and CNS-0745846. The authors gratefully acknowledge the NSF for its continued support.

References

[1] Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules in large databases. VLDB '94, pages 487–499, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers Inc.

[2] Elizabeth A. Basha, Sai Ravela, and Daniela Rus. Model-based monitoring for early warning flood detection. SenSys '08, pages 295–308, New York, NY, USA, 2008. ACM.

[3] P. Bonnet, M. Leopold, and K. Madsen. Hogthrob: Towards a sensor network infrastructure for sow monitoring (wireless sensor network special day). DATE '06, pages 1109–1109, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[4] Octav Chipara, Chenyang Lu, Thomas C. Bailey, and Gruia-Catalin Roman. Reliable clinical monitoring using wireless sensor networks: Experiences in a step-down hospital unit. SenSys '10, pages 155–168, New York, NY, USA, 2010. ACM.

[5] Lewis Girod, Jeremy Elson, Alberto Cerpa, Thanos Stathopoulos, Nithya Ramanathan, and Deborah Estrin. Emstar: A software environment for developing and deploying wireless sensor networks. In *USENIX 2004 Annual Technical Conference*, pages 283–296, 2004.

[6] Lewis Girod, Nithya Ramanathan, Jeremy Elson, Thanos Stathopoulos, Martin Lukac, and Deborah Estrin. Emstar: A software environment for developing and deploying heterogeneous sensor-actuator networks. *ACM Trans. Sen. Netw.*, 3(3), August 2007.

[7] Intelligent River Team. Intelligent river project, http://www.intelligentriver.org/, May 26, 2014 (*last access*).

[8] Mohammad Maifi Hasan Khan, Hieu Khac Le, Hossein Ahmadi, Tarek F. Abdelzaher, and Jiawei Han. Dustminer: Troubleshooting interactive complexity bugs in sensor networks. SenSys '08, pages 99–112, New York, NY, USA, 2008. ACM.

[9] Ted Tsung-Te Lai, Wei-Ju Chen, Kuei-Han Li, Polly Huang, and Hao-Hua Chu. Triopusnet: Automating wireless sensor network deployment and replacement in pipeline monitoring. IPSN '12, pages 61–72, New York, NY, USA, 2012. ACM.

[10] Mo Li and Yunhao Liu. Underground coal mine monitoring with wireless sensor networks. *ACM Trans. Sen. Netw.*, 5(2):10:1–10:29, April 2009.

[11] Lilia Paradis and Qi Han. A survey of fault management in wireless sensor networks. *Journal of Network and Systems Management*, 15(2):171–190, 2007.

[12] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria, http://www.R-project.org/, 2013.

[13] Nithya Ramanathan, Kevin Chang, Rahul Kapur, Lewis Girod, Eddie Kohler, and Deborah Estrin. Sympathy for the sensor network debugger. SenSys '05, pages 255–267, New York, NY, USA, 2005. ACM.

[14] Nithya Ramanathan, Eddie Kohler, and Deborah Estrin. Towards a debugging system for sensor networks. *Int. J. Netw. Manag.*, 15(4):223–234, July 2005.

[15] RStudio, Inc. *shiny: Web Application Framework for R*. http://CRAN.R-project.org/package=shiny, 2013. R package version 0.8.0.

[16] Robert Szewczyk, Eric Osterweil, Joseph Polastre, Michael Hamilton, Alan Mainwaring, and Deborah Estrin. Habitat monitoring with sensor networks. *Commun. ACM*, 47(6):34–40, June 2004.

[17] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A macroscope in the redwoods. SenSys '05, pages 51–63, New York, NY, USA, 2005. ACM.

[18] Wei Wang, Vikram Srinivasan, Kee-Chaing Chua, and Bang Wang. Energy-efficient coverage for target detection in wireless sensor networks. IPSN '07, pages 313–322, New York, NY, USA, 2007. ACM.