D2: Anomaly Detection and Diagnosis in Networked Embedded Systems by Program Profiling and Symptom Mining

Wei Dong¹, Chun Chen¹, Jiajun Bu¹,
Xue Liu², Yunhao Liu³
1. Zhejiang University; 2. McGill University;
3. Tsinghua University
2013-12-5
Introduction

- Detecting and diagnosing anomalies in networked embedded systems is difficult.
- Case 1: LOFA-argo

  - Low data rate due to malfunction in TMAC
  - Causes are left unclear
Introduction

- Case 2: GreenOrbs [SenSys’09, INFOCOM’11]

- Bugs in TinyOS low level drivers require considerable time to fix
<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>V1</td>
<td>08-10-2010</td>
<td>The initial version</td>
</tr>
<tr>
<td>V40</td>
<td>09-03-2010</td>
<td>Fix the log bug to log onto the flash</td>
</tr>
<tr>
<td>V48</td>
<td>09-06-2010</td>
<td>Fix PAD mark bug</td>
</tr>
<tr>
<td>V72</td>
<td>09-10-2010</td>
<td>Fix c3 packet size bug</td>
</tr>
<tr>
<td>V75</td>
<td>09-12-2010</td>
<td>Fix dissemination bug by employing multi-collection sender</td>
</tr>
<tr>
<td>V87</td>
<td>09-22-2010</td>
<td>Fix testing bug by adding log, LPL</td>
</tr>
<tr>
<td>V101</td>
<td>09-26-2010</td>
<td>Fix loop counter</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
<td>......</td>
</tr>
<tr>
<td>V193</td>
<td>01-07-2011</td>
<td>Fix sync bug to preserve metadata</td>
</tr>
<tr>
<td>V215</td>
<td>01-10-2011</td>
<td>Fix boot bug by providing Splitcontrol</td>
</tr>
<tr>
<td>V227</td>
<td>01-11-2011</td>
<td>Fix logging bug by start radio after log booting</td>
</tr>
<tr>
<td>V288</td>
<td>05-12-2011</td>
<td>Fix sink_loss bug in sink node</td>
</tr>
<tr>
<td>V357</td>
<td>09-15-2012</td>
<td>Fix a time sync bug</td>
</tr>
<tr>
<td>V366</td>
<td>10-20-2012</td>
<td>Add CC2420/receive to the repository</td>
</tr>
</tbody>
</table>
Related work

- There are numerous existing works
  - Node-level debugging, tracing and logging
    - Clairvoyant \([\text{SenSys'07}]\), NodeMD \([\text{MobiSys'07}]\), DT \([\text{SenSys'10}]\), Aveksha \([\text{SenSys'11}]\), T-Morph \([\text{FSE'12}]\)
    - ... 
  - Network-level diagnosis
    - Sympathy \([\text{SenSys'05}]\), PAD \([\text{SenSys'08}]\), AD \([\text{INFOCOM'11}]\), LD2 \([\text{INFOCOM'12}]\) ...
Motivation

- Node-level debugging tools vs. Network-level diagnosis tools
- A simple combination of the two will cause large overhead. Moreover, some errors may not be reproducible
- To close the gap, we propose D2, a new anomaly detection and diagnosis method by combining program profiling and symptom mining
D2’s main idea

- We employ *binary instrumentation* to perform *lightweight* function count profiling. Our method treats the program as a black box, thus is *scalable* for a wide range of applications.

- Based on the *fine-grained* statistics, we employ *PCA* (*Principal Component Analysis*) *based approach* for automatically detecting network problems.

- D2 is able to point programmers closer to the most likely causes by a novel approach combining *statistical tests and program call graph analysis*. 
D2’s overview

Binary instrumentation

Request
Profiles

PC

Failure Detection
Failure Diagnosis

Diagnosis Report
Binary instrumentation

- The D2 module (at the sensor node) finds the start of each function.
- The D2 module uses the *trampoline* technique to track the count of each function’s execution.
- The D2 module allocates free RAM space and dynamically updates the function counters (i.e., profile).
- The D2 module *adaptively takes snapshots* of the function counters and sends the profile to the external flash (for later analysis).
The trampoline technique

1. replace

Inst 1 → call → Inst 2

(1) save context
(2) counter[1]++
(3) call check()
(4) restore context
(5) Inst 1
(6) ret 1

mirror
Adaptively taking snapshots

- **Why?**
  - Time variations in the long-term execution can be captured

- **How?**
  - Native approach: take snapshots every fixed interval, e.g., 10 minutes
  - Problem: extra overhead if no activities happened
  - Our approach: take snapshots when the total function count in a period reaches a threshold, e.g., 5000.
Problem detection

- What we have? snapshots of function counters
- What we want to do? Which snapshots are anomalies

```
\begin{array}{lll}
20 & 10 & 20 \\
30 & 40 & 30 \\
\vdots & \vdots & \vdots \\
20 & 20 & 20 \\
30 & 30 & 30 \\
\end{array}
```

Send()
Receive()
sense()
blink()
Anomaly detection

- Key assumptions:
  - During normal executions the relative frequency of two function counts in a time window usually stays the same.
  - For example, the ratio between functions send() and receive() in the CTP component is usually very stable.
  - The actual count does not matter (as it depends on workloads), but the ratio among different function counts matters.
PCA (Principal Component Analysis)

- PCA captures patterns in high-dimensional data by automatically choosing a set of principal components (i.e., coordinates).

- PCA is able to capture the essence of correlation in the data.
PCA (Principal Component Analysis)
Problem diagnosis

- What we know?
  - Which snapshots are anomalies
  - E.g., node 2 in the first day exhibits abnormal behavior

- What we do not know?
  - Which functions are wrong?
t-tests

- We use *t-tests* to compare data points in the normal space with those in the abnormal space.

- We compare $n$ function counts and $n^2$ function count ratios.

\[
\begin{align*}
t &= \frac{\bar{X}_1 - \bar{X}_2}{S_{X_1X_2} \sqrt{\frac{1}{n_1} + \frac{1}{n_2}}} \\
S_{X_1X_2} &= \sqrt{\frac{(n_1 - 1)S_{X_1}^2 + (n_2 - 1)S_{X_1}^2}{n_1 + n_2 - 2}}
\end{align*}
\]
Generating diagnosis report

- What we have now?
  - a list of suspicious functions or ratios between two functions ranked by their statistical significance

- The result can further be refined by considering the call/post relationship between functions.

- D2 obtains the call/post relationships by program analysis

- D2 generates diagnosis report showing both statistical difference as well as the call/post relationships.
Evaluations

- We implement D2 on TelosB/TinyOS

- We evaluate D2’s overhead in terms of
  - Memory overhead
  - CPU overhead

- We evaluate D2’s efficacy using cases from real-world sensor systems
## RAM overhead (bytes)

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Without D2</th>
<th>With D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blink</td>
<td>48</td>
<td>196</td>
</tr>
<tr>
<td>RadioCountToLeds</td>
<td>70</td>
<td>284</td>
</tr>
<tr>
<td>TestDissemination</td>
<td>85</td>
<td>344</td>
</tr>
<tr>
<td>TestNetwork</td>
<td>157</td>
<td>632</td>
</tr>
<tr>
<td>Oscilloscope</td>
<td>101</td>
<td>408</td>
</tr>
</tbody>
</table>
Program flash overhead

![Bar chart showing program flash overhead for different tasks. The chart displays percentages for tasks such as Blink, RadioCountToLeds, TestDissemination, TestNetwork, and Oscilloscope. The percentages range from 0% to 20%.](image-url)
## CPU overhead

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Without D2</th>
<th>With D2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blink</td>
<td>1.38%</td>
<td>1.59%</td>
</tr>
<tr>
<td>RadioCountToLeds</td>
<td>1.22%</td>
<td>1.45%</td>
</tr>
<tr>
<td>TestDissemination</td>
<td>1.40%</td>
<td>1.60%</td>
</tr>
<tr>
<td>TestNetwork</td>
<td>2.16%</td>
<td>2.50%</td>
</tr>
<tr>
<td>Oscilloscope</td>
<td>4.82%</td>
<td>5.57%</td>
</tr>
</tbody>
</table>
Case 1: flash broken

- **Symptom**: nodes with broken external flash have CPU utilization (~90%) much higher than normal nodes (<5%)

- **Without D2**, we do not know how to fix the code

- **With D2**
  - Automatically detect the abnormal nodes
  - Help us diagnosis, i.e., point us closer to the buggy function
Case 1: diagnosis report
Case 1: fixing the problem

- Looking into the suspicious functions, we could easily guess the causes of the bug
  - when the code powers up the external flash, it does not check the status of the hardware. Therefore, if the external flash is broken, the code would repeatedly make requests to acquire the resource
- We finally fix the bug in Spi.powerUp() function.
- This bug still exists in the latest TinyOS
Case 2: CTP queue overflow

- **Symptom**: nodes near the sink are more likely to experience heavier losses

- **Without D2**
  - we do not know how improve the design

- **With D2**
  - Help us diagnosis, i.e., point us closer to the suspicious function
Case 2: diagnosis report
Case 2: fixing the problem

- The diagnosis report indicates that the ratio between receive() and send() decreases in a few snapshots.

- Looking into the code, we indeed find that the default CTP implementation does not turn on the congestion control mechanism.

- We implement a simple congestion control mechanism which address the problem fairly well.
Conclusion

- We propose D2, a novel method combining program profiling and symptom mining for detecting and diagnosing anomalies in networked embedded systems.
- We propose a novel approach combining statistical tests and program call graph analysis to point programmers closer to the most likely causes.
- We implement our method and demonstrate its effectiveness using case studies from real sensor network applications.
Thank you!

dongw@zju.edu.cn