The 34th IEEE Real-Time Systems Symposium (RTSS 2013)
Vancouver, Canada

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

Wei Dong[1], Chun Chen[1], Jiajun Bu[1],

Xue Liu[2], Yunhao Liu[3]

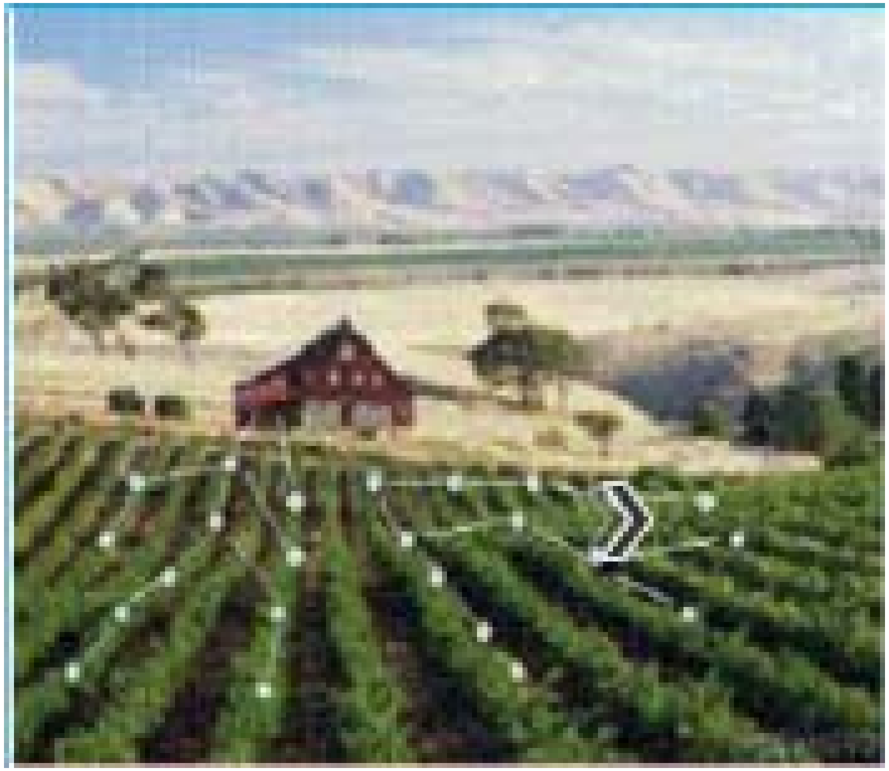
1. Zhejiang University; 2. McGill University;

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 requires considerable time to fix***

GreenOrbs SVN repository

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

Related work

- There are numerous existing works
  - Node-level debugging, tracing and logging

    Clairvoyant **[SenSys'07]**, NodeMD **[MobiSys'07]**, DT **[SenSys'10]**, Aveksha **[SenSys'11]**, T-Morph **[FSE'12]**

    …
  - Network-level diagnosis

    Sympathy **[SenSys'05]**, PAD **[SenSys'08]**, AD **[INFOCOM'11]**, LD2 [INFOCOM'12] …

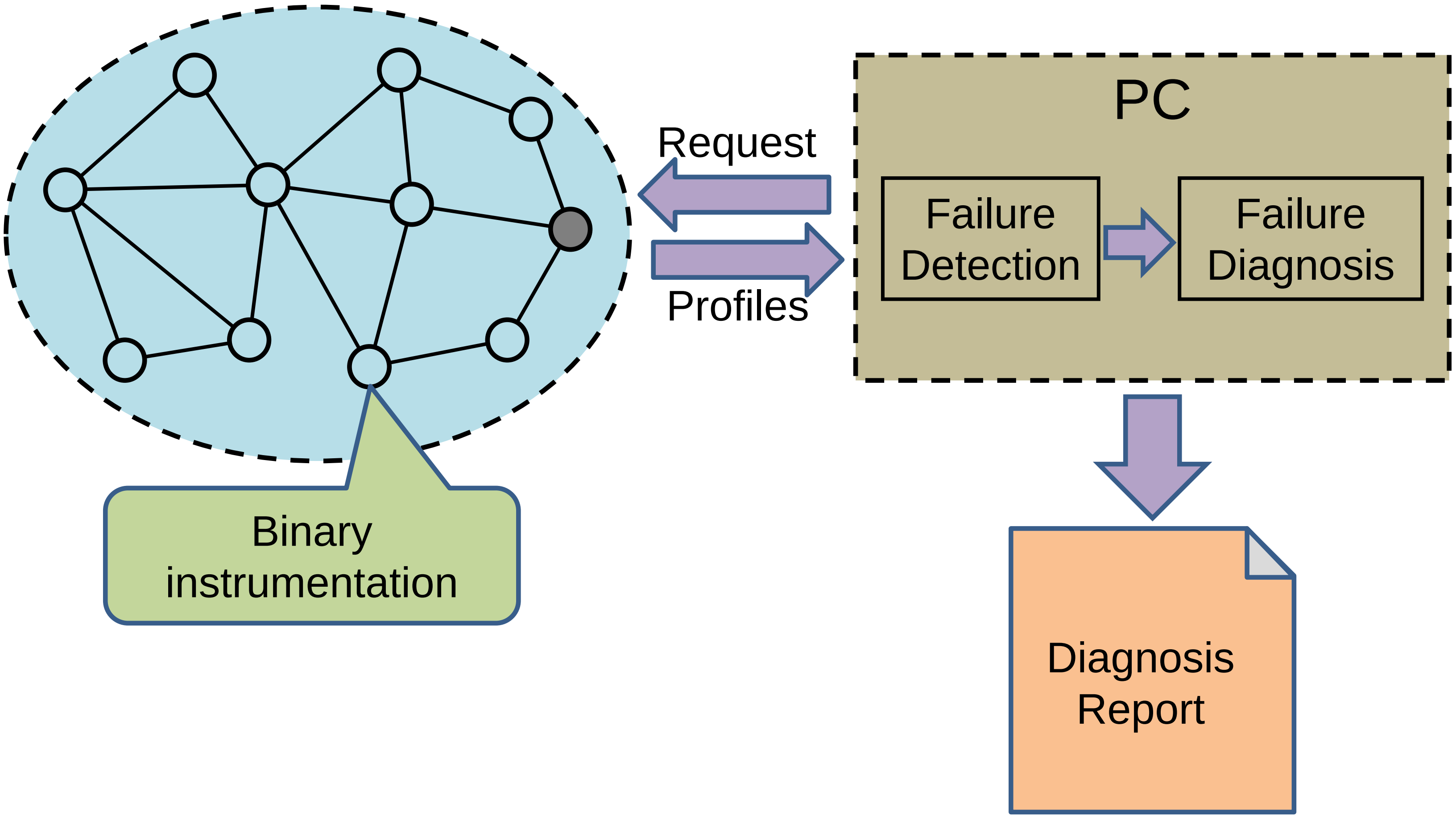
Motivation

- 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*.

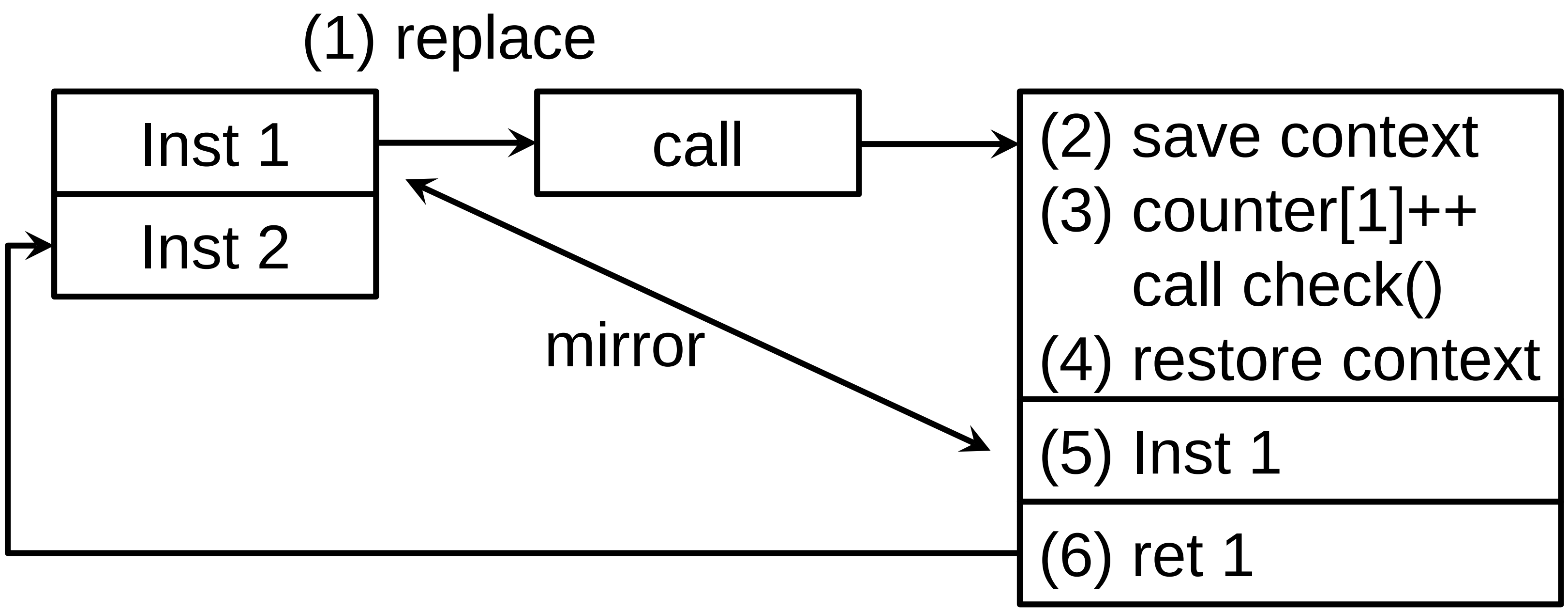
D2's overview



Binary instrumentation

- 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



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

$$\begin{bmatrix} 20 \\ 30 \\ \cdots \\ 20 \\ 30 \end{bmatrix} \quad \begin{bmatrix} 10 \\ 40 \\ \cdots \\ 20 \\ 30 \end{bmatrix} \quad \begin{bmatrix} 20 \\ 30 \\ \cdots \\ 20 \\ 30 \end{bmatrix} \quad \begin{matrix} \text{Send()} \\ \text{Receive()} \\ \\ \text{sense()} \\ \text{blink()} \end{matrix}$$

Anomaly detection

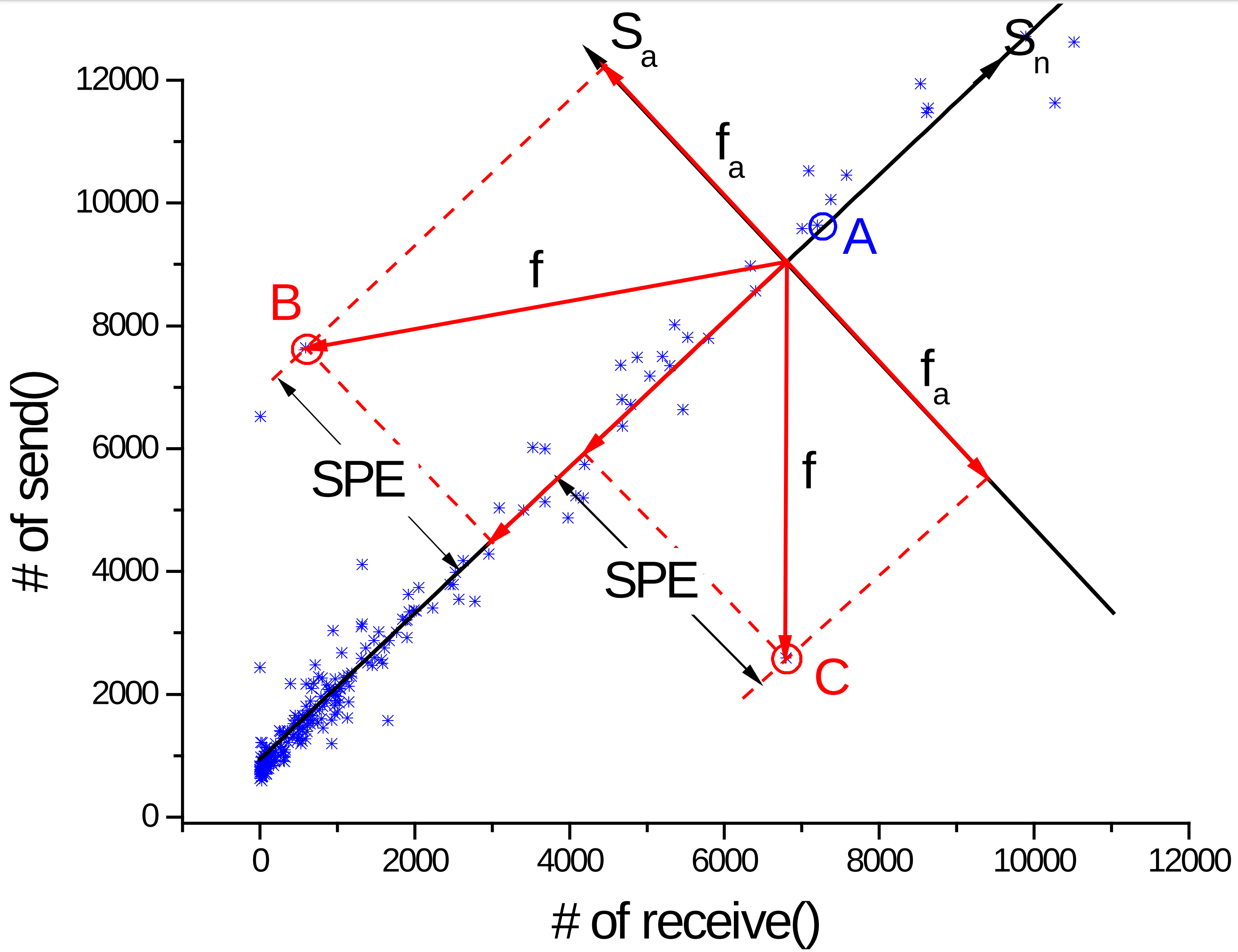
- Key assumptions:
  - During normal executions the relative frequency of two function counts in a time window usually stays the same.
  - For example, the ratio between functions send() and receive() in the CTP component is usually very stable
  - The actual count does not matter (as it depends on workloads),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

$$\begin{bmatrix} 10 \\ 40 \\ \cdots \\ 20 \\ 30 \end{bmatrix}$$

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

$$t = \frac{\overline{X_1} - \overline{X_2}}{S_{X_1 X_2}\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}$$

$$S_{X_1 X_2} = \sqrt{\frac{(n_1 - 1)S_{X_1}^2 + (n_2 - 1)S_{X_1}^2}{n_1 + n_2 - 2}}$$

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.

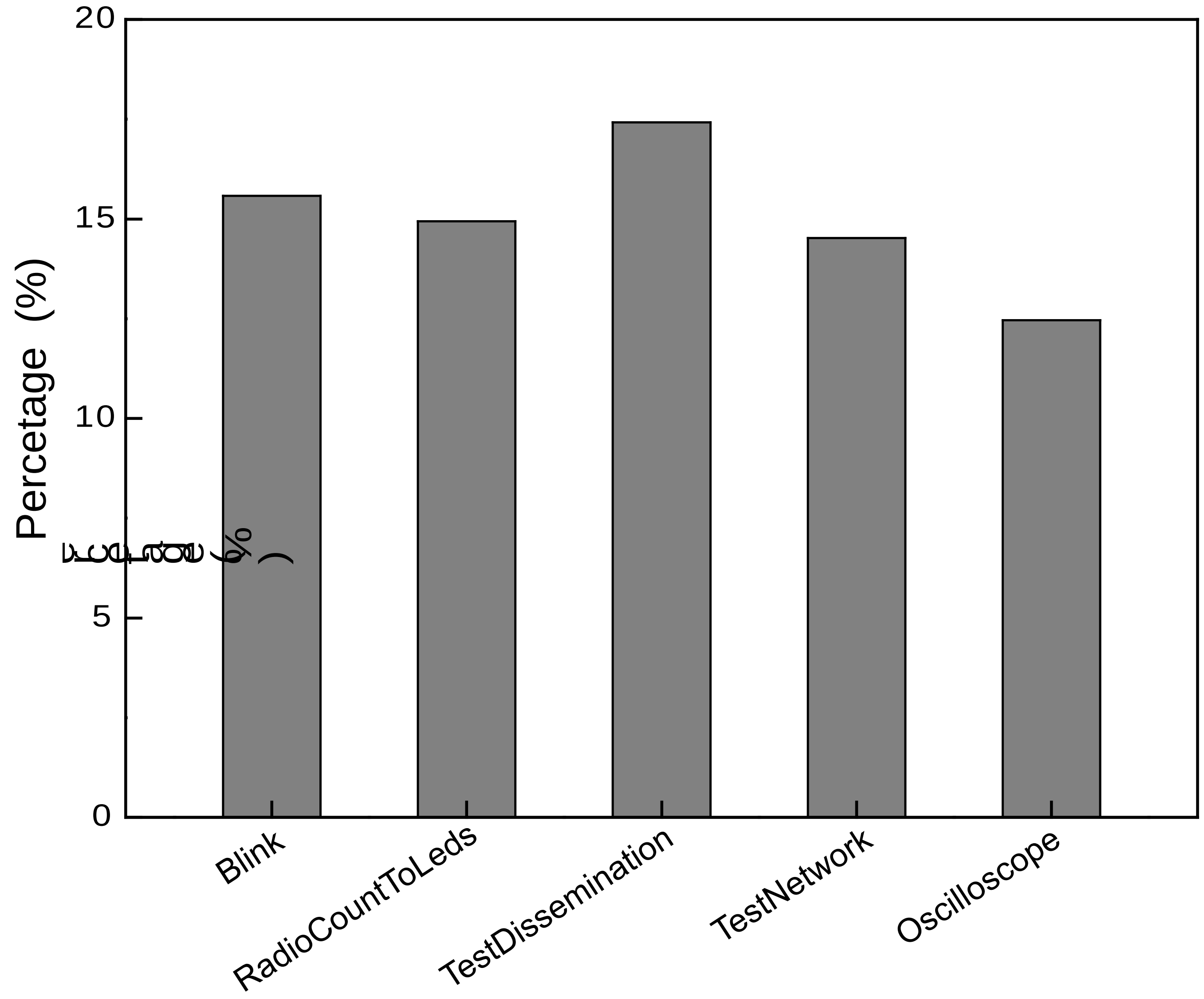
Evaluations

- 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)

Benchmark	Without D2	With D2
Blink	**48**	**196**
RadioCountToLeds	**70**	**284**
TestDissemination	**85**	**344**
TestNetwork	**157**	**632**
Oscilloscope	**101**	**408**

Program flash overhead



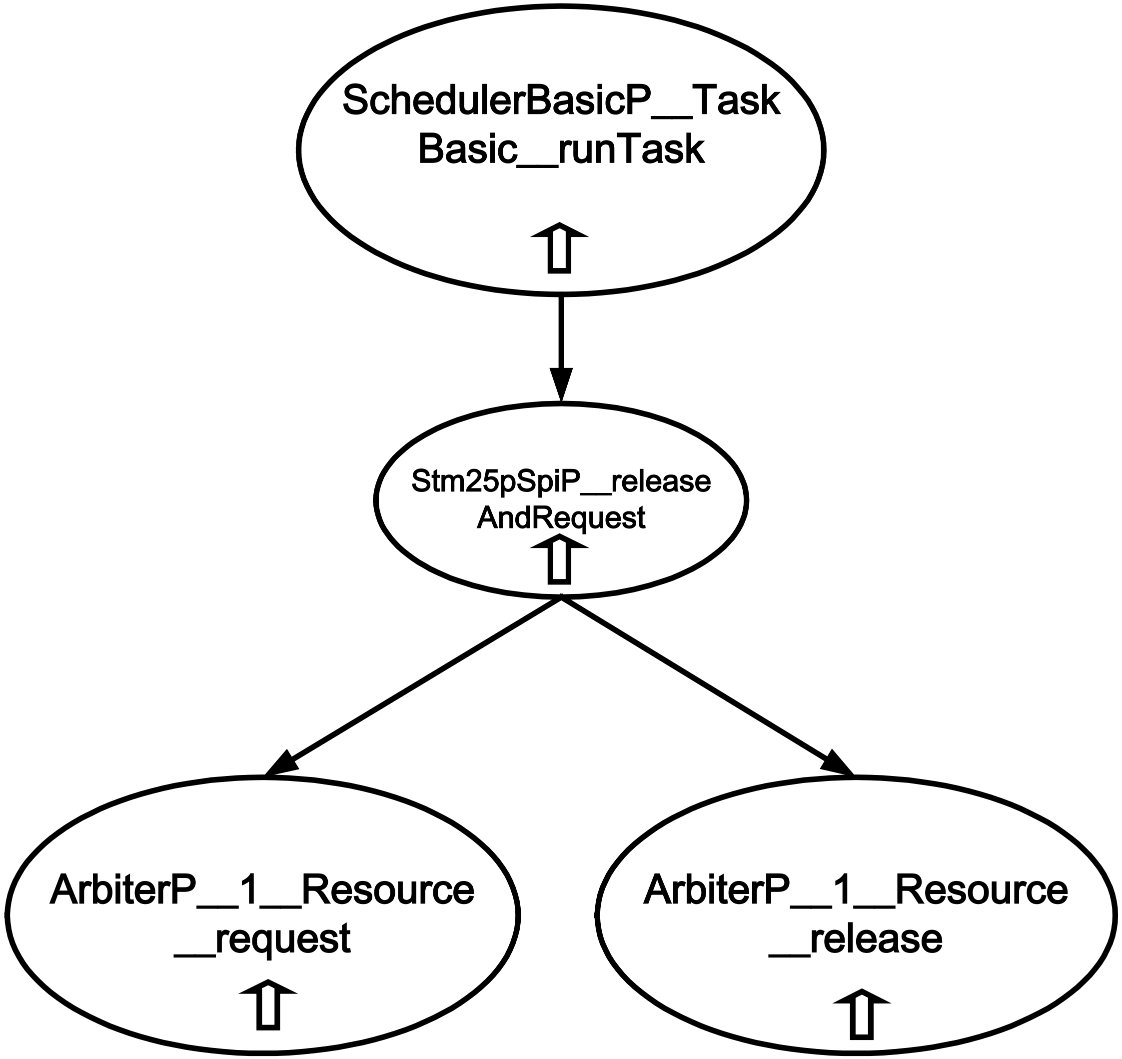
CPU overhead

Benchmark	Without D2	With D2
Blink	**1.38%**	**1.59%**
RadioCountToLeds	**1.22%**	**1.45%**
TestDissemination	**1.40%**	**1.60%**
TestNetwork	**2.16%**	**2.50%**
Oscilloscope	**4.82%**	**5.57%**

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



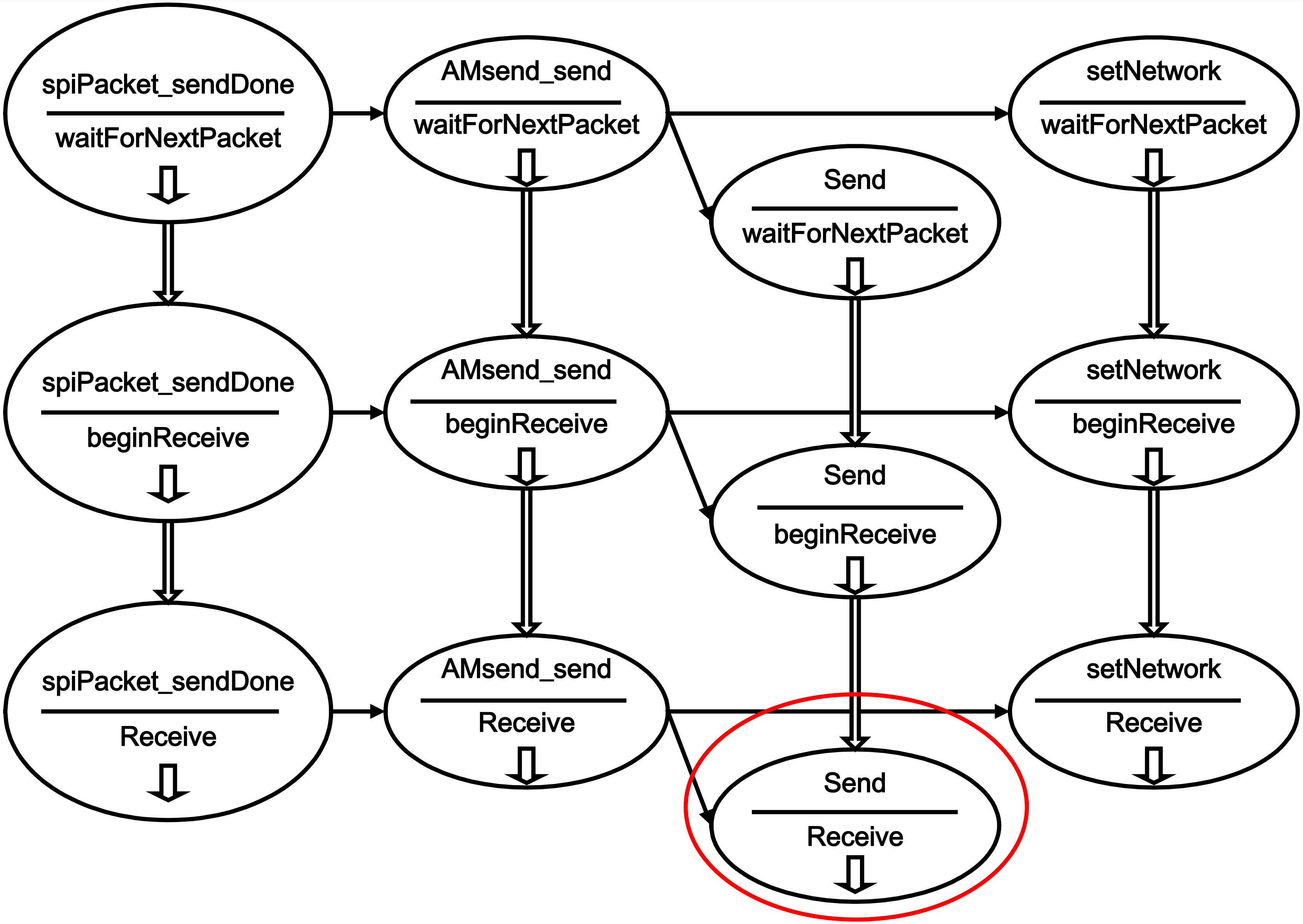
Case 1: fixing the problem

- 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.

Q & A

Thank you!

dongw@zju.edu.cn