Towards Practical Debugging of Wireless Sensor Networks

Matthew Bradbury
Tim Law
Ivan Leong
Dan Robertson
Amit Shah
Joe Yarnall

CS407: Fourth Year Project

10–11AM Tuesday 7th May 2013

Introduction

What is a Wireless Sensor Network

A wireless sensor network (WSN) is a collection of computing devices called motes, they have:

- a short range wireless radio
- an array of sensors such as light, heat and humidity
- a simple low powered CPU
- a battery with limited power supply

Motes communicate with each other to form a WSN. WSNs perform data gathering tasks such as environment monitoring.

The Problem of Debugging Distributed Systems

- Multiple tasks running simultaneously leads to non–deterministic interactions
- Traditional debugging tools are unsuited
- Timing and synchronisation issues

Complications to the Problem

- Motes are energy constrained
- Sending messages is the most expensive task
- Receiving messages is the next most expensive task [Shnayder et al., 2004]
- Motes have low computing power and a small memory
- WSNs deployed in hard to reach areas — physical access after deployment is difficult [Herbert et al., 2007]

Related Work

- Global Predicate Detection [Garg and Waldecker, 1996]
- H–SEND [Herbert et al., 2007]
- NodeMD [Krunic et al., 2007]
- TinyOS [Levis et al., 2005], Contiki

Project Aims

- Develop tools to aid in debugging distributed programs running on WSNs.
- Implement libraries that check predicates, with a focus on correctly evaluating these predicates.
- Investigate if there are places in the network where evaluation is more efficient.
- Visualise some of the state of the network, as part of a tool to inform system users what state the network is in.

Project Management

- Matthew — Group Leader
- Amit — Technical Manager
- Everyone was involved with development and research
- Bitbucket.org Git repository

Implemented Libraries

Implemented Libraries — Containers

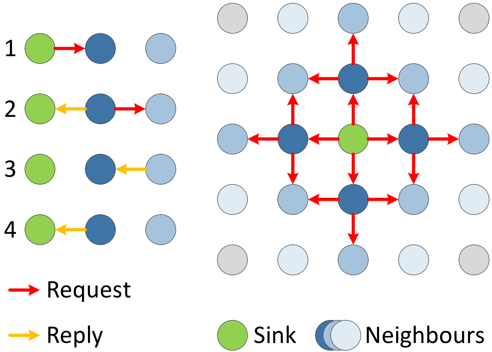
- Linked List
  - Our's: Standard linked list
  - Contiki's: Intrusive linked list
- Array List
- Unique Array
- Map

Benefits:

- Abstraction
- Reduced code duplication
- Simplified memory management

Implemented Libraries — N–Hop Request

- Used by predicate evaluation
- Floods request N–Hops away from sink
- Asks for motes current state
- Returned along the chain created by the flooding stage



Implemented Libraries — N–Hop Flood

- Floods a given packet N hops
- Surprised that Contiki did not provide this as a library
  - Had to implement ourselves using TTLs in packet headers

Implemented Libraries — Event Update

- Used by predicate evaluation
- Periodically checks if node's data has changed
- If it has floods the new data the required number of hops

Depends On:

- N–Hop Flood

Implemented Libraries — Multi–Packet Unicast

- Contiki packet size: 128 bytes
- This is too small for some of our data
- Alternative APIs Contiki implements are convoluted
    - Targeted towards sending file chunks
- We split packet up, send pieces and then reassemble

Implemented Libraries — Tree Aggregation

1. Leaf node generates data, forwards to parent
2. Parent waits for a period, aggregating received data
3. The node adds its own data to the aggregate
4. The node then forwards the message to its parent
5. Repeat until the aggregated message reaches the base station

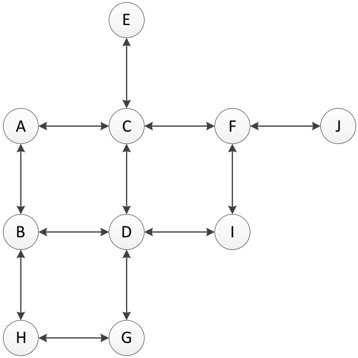
Again surprised that Contiki didn't have an implementation

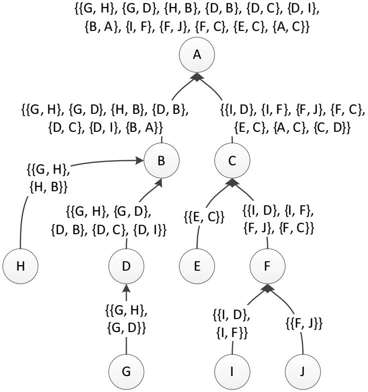
Depends On:

- Multi–Packet

Implemented Libraries — Neighbour Detection

- To debug a WSN we need to know the network topology
- Uses Tree Aggregation to send neighbours to sink





Predicate Evaluation

Predicate Evaluation

1. Disseminate predicate to network
2. Evaluate predicate
3. Return response to base station

Consider:

- Where to evaluate: Local vs. Global
- When to propagate mote data: Periodic vs. Event
- How to respond to a failure or success

Predicate Evaluation — Libraries I

	Periodic	Event
Local	PELP • Evaluated in–network • Data is requested when needed to evaluate predicate • Previous round's data is forgotten after a round completes	PELE • Evaluated in–network • Data is sent by data sources, when it changes • Data is never forgotten, simply updated

Predicate Evaluation — Libraries II

	Periodic	Event
Global	PEGP • Similar to PELP, except data is aggregated to the base station	PEGE • Similar to PELE, except data is aggregated to the base station

Predicate Evaluation — Libraries Used

	Periodic	Event
Local	PELP • N–Hop Request	PELE • Event Update • N–Hop Flood
Global	PEGP • Neighbour Detect • Tree Aggregation	PEGE • Neighbour Detect • Tree Aggregation

Predicate Evaluation — Response

Failure

- Only predicate failures are reported to the base station
- Cannot say much about the network state, either:
  - we have not been informed of a failure
  - we have been informed
- We chose this one due to the reduced traffic

Failure and Success

- Both failures and successes are reported
- Supports detecting the network is in the following states:
  - Unknown
  - Failed
  - Succeeded
  - Failed and then later succeeded
- Uses more energy

Note: Global PE may as well take advantage of "Failure and Success" messages, as the target of them is the node the predicates are evaluated on

Predicate Evaluation — Evaluation

- Implemented a virtual machine in C to evaluate predicates on the nodes
  - Optimised for low memory environment
  - Opcodes for high–level operations to reduce program size
- Implemented a DSL with a JavaCC parser and a Java compiler and assembler
  - Functional language
  - Expects a boolean output

Predicate Evaluation — DSL

$$\forall n \in \mathsf{Nodes}\cdot$$
$$\forall n' \in \mathsf{Neighbours}(n, 2)\cdot$$
$$\mathsf{slot}(n) \neq \mathsf{slot}(n')$$

```
[all]
function 1 as slot returning int in
    using Neighbours(2) as twohopn in
        @(x : twohopn ~
            slot(x) != slot(this)
        )
```

Predicate Evaluation — DSL

$$
\begin{aligned}
&\forall n \in \text{Nodes}\cdot\\
&\quad \forall n' \in \text{Neighbours}(n,1) \cup \{n\}\cdot\\
&\quad\quad \forall n'' \in \text{Neighbours}(n,1) \cup \{n\}\cdot\\
&\quad\quad\quad \text{addr}(n') \neq \text{addr}(n'')\\
&\quad\quad\quad\quad \implies \text{slot}(n') \neq \text{slot}(n'')
\end{aligned}
$$

```
[all]
function 0 as addr returning int in
function 1 as slot returning int in
    using Neighbours(1) as ohn in
        @(a : ohn ~
            @(b : ohn ~ addr(a) != addr(b)
                => slot(a) != slot(b))
            & slot(a) != slot(this)
        )
```

Predicate Evaluation — DSL

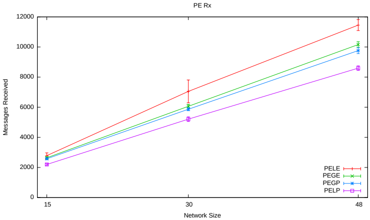
```
//TARGETING all
//FUNC 0 AS addr
//FUNC 1 AS slot
//STORING 1 IN ohn
IVAR 1
IPUSH 1
IPUSH 0
ISTORE 1
start1: ALEN 255
INEQ
JZ end1
IVAR 2
IPUSH 1
IPUSH 0
ISTORE 2
start2: ALEN 255
INEQ
JZ end2
//addr(a[*1])
VIFAFC 1 255 0
//addr(b[*2])
VIFAFC 2 255 0
INEQ
//slot(a[*1])
VIFAFC 1 255 1
//slot(b[*2])
VIFAFC 2 255 1
INEQ
IMPLIES
AND
VIINC 2
JMP start2
//slot(a[*1])
end2: VIFAFC 1 255 1
//slot(this)
THISC 1
INEQ
AND
AND
VIINC 1
JMP start1
end1: HALT
//VD 1 = 255
```

Results

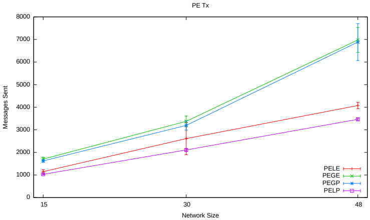
Results Methodology

- Run and measure energy usage of TDMA algorithm
- Measure energy cost of predicate evaluation algorithm
  - Checking for slot collisions
  - Vary predicate distance (1–hop and 2–hop)
  - Vary predicate evaluation algorithm
- Network was laid out as a grid
- N, S, E, W communication possible
- 5 minutes setup time for PE, start TDMA
- 35 minutes total runtime

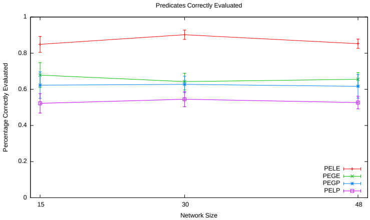
Results when period=4.0 minutes using a 1–hop predicate



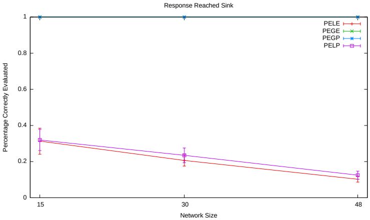
(a) Rx



(b) Tx

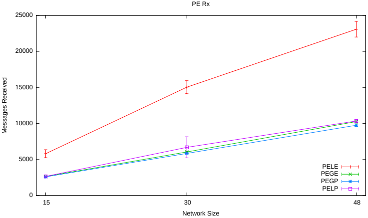


(c) Percentage of predicates correctly evaluated

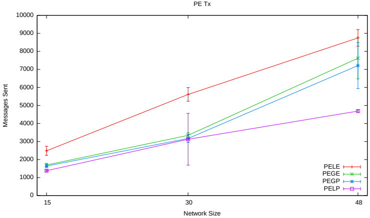


(d) Percentage of responses received

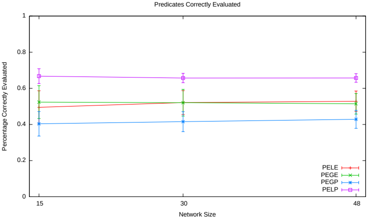
Results when period=4.0 minutes using a 2–hop predicate



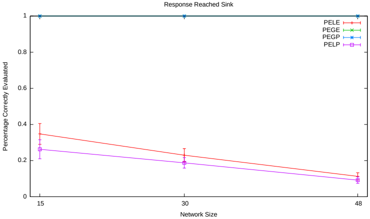
(e) Rx



(f) Tx



(g) Percentage of predicates correctly evaluated



(h) Percentage of responses received

Demo

Visualisation Tool and Network Interface

Features:

- Creating and compiling predicates to monitor
- Deploying predicates to the WSN
- Recording history of evaluation results
- Use of `serialdump-linux` to interface with sink mote

Views:

- Predicate view
- Network view

Conclusions

Future Work

- Improve memory management
- Improve C containers developed
- Stateful predicates
- Handle mote mobility
- Improve failure response message deliver ratio

Summary

Developed:

- Libraries for use in Contiki (Container and Network)
- Predicate Evaluation Libraries (Global and Local)
- GUI tool to interface with network

Found:

- In–network, event–based evaluation suitable for “small” predicates
- Global, periodic evaluation more suitable for “large” predicates

References I

Garg, V. and Waldecker, B. (1996).
Detection of strong unstable predicates in distributed programs.
Parallel and Distributed Systems, IEEE Transactions on, 7(12):1323–1333.

Herbert, D., Sundaram, V., Lu, Y.-H., Bagchi, S., and Li, Z. (2007).
Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking.
ACM Transactions on Autonomous and Adaptive Systems (TAAS), 2(3):8.

References II

Krunic, V., Trumpler, E., and Han, R. (2007).
Nodemd: diagnosing node-level faults in remote wireless sensor systems.
In *Proceedings of the 5th international conference on Mobile systems, applications and services*, MobiSys '07, pages 43–56, New York, NY, USA. ACM.

Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D. (2005).
Tinyos: An operating system for sensor networks.
In Weber, W., Rabaey, J., and Aarts, E., editors, *Ambient Intelligence*, pages 115–148. Springer Berlin Heidelberg.

References III

Shnayder, V., Hempstead, M., rong Chen, B., Allen, G. W., and Welsh, M. (2004).
Simulating the power consumption of large-scale sensor network applications.
In *In Sensys*, pages 188–200. ACM Press.

The End

Any Questions?