

# Towards Debugging of Wireless Sensor Networks

## Abstract

Debugging tools are vital for developers to produce reliable software, however traditional tools are less useful when developing software for new system paradigms such as wireless sensor networks. As wireless sensor networks become increasing prevalent in our lives it will become ever more important that the software they are running works reliably and to do this debugging tools will be required. This project investigates how predicates can be specified and checked on wireless sensor node and how errors can be reported to a base station.

## Background

Wireless Sensor Networks (WSNs) are a collection of small computing devices equipped with a radio (for wireless communication), sensors (to get information on the environment) and a battery to power the device.

As these devices are not attached to the mains power we need to make sure that we do as little energy intensive work (eg. Receiving/Transmitting data) as possible to lengthen the lifetime and usefulness of the network.

As energy is a limiting factor, the software developed for WSNs typically trade reliability for decreased energy usage. That means that very special care must be taken to develop programs that behave well and continue to behave well under real world conditions.

## Project Management

For this project we allocated roles as shown in the table to the right. We met weekly to discuss our progress as well as allocate tasks to each team member for the week.

Name	Role
Matthew Bradbury	Group Leader
Tim Law	Developer and Researcher
Ivan Leong	Developer and Tester
Daniel Robertson	Project Manager
Amit Shah	Technical leader
Joe Yarnall	Developer and Tester

We liaised with other academics and PhD students to assist in our project, and allocate time when we could have access to the wireless sensor nodes.

## Predicate Specification

One of the requirements of our project is that we wished to specify predicates to be checked, not just at compile-time, but also at run-time. The main reason is that the needs of a system will change over time.

We explored a variety of methods of specifying predicates for evaluation, with an aim to enable an expressive syntax. However, we found that existing solutions – such as eLua (Embedded Lua) [CITE] and Wren [CITE] – were not suitable due to their high resource requirements or lack of desired features.

Thus, we implemented our own predicate-based scripting language that provided the features we wanted, and the ability to run on the limited resources available. We designed a syntax that enabled us to target nodes and utilise built in functions to iterate over sets of nodes when evaluating first-order predicates.

In order to realise this, we have implemented a parser, assembler and virtual machine. The base application will compile new predicates and transmit the assembled bytecode to nodes for execution.

As an example of a predicate specified in our language, the following snippet will check no nodes within a two-hop neighbourhood have clashing slots. This is a property that should hold in systems using the TDMA (time division multiple access) MAC protocol.

```
using Neighbours(2) as twohopn in    ∀x ∈ Neighbours(2) ·
  @(x : twohopn ~                    slot(x) ≠ slot(this)
    slot(x) != slot(this))
```

Listing 1. A predicate in our custom language and the same predicate in first order logic

## Visualisation and Interface Tool

We are developing a GUI to control our project, which will run on a computer and interface with the base station. It's main features will be predicate creation and deployment, and network visualisation.

Monitoring a new predicate will simply involve writing a predicate in our custom scripting language, and submitting it using the GUI for evaluation in the network.

In order to visualise the WSN and the state of the attached predicates, we have two main views in mind. The first is simply a list of predicates, colour coded by their current status, enabling the user to see at a glance any problems that arise.

The second will be a network graph, showing the spatial relationships between nodes, associated clustering hierarchies, and generally provide a more visual representation of the WSN and attached predicates.

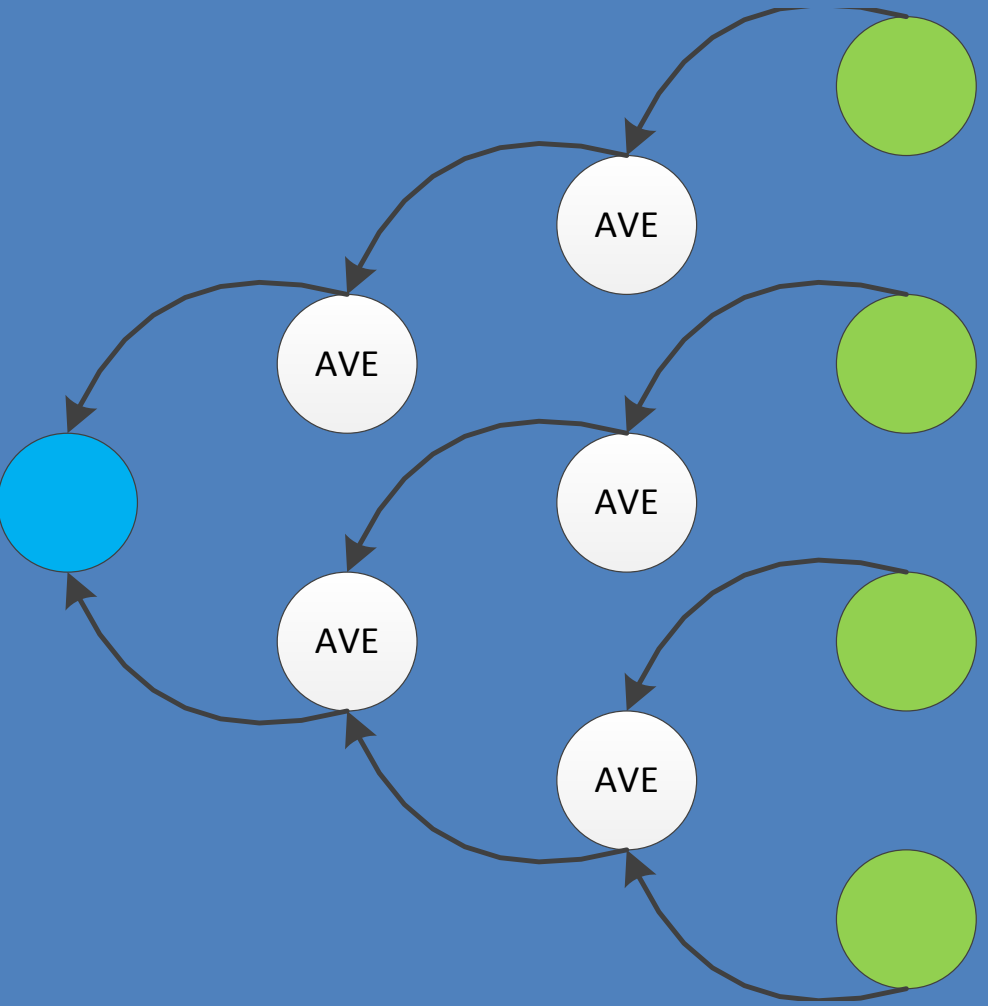


## Algorithms

When developing software for WSNs, there are several algorithms that are commonly used to perform certain tasks. So, in order to test that our solution is applicable to real world code, we have developed some of these algorithms for us to specify and check predicates for.

### Tree Aggregation

A common task is to have every node in a network send data to the sink, whilst having some aggregation function performed on that data. To minimise the number of retransmissions the network is organised as a tree. In this tree a node will collect child node's data and perform some function over that data, once all children's information is received the aggregated result will be sent to the parent of that node. Eventually the sink will receive and aggregate information from its children. Common applications for this algorithm is to calculate the average temperature over the area monitored by the sensor network.

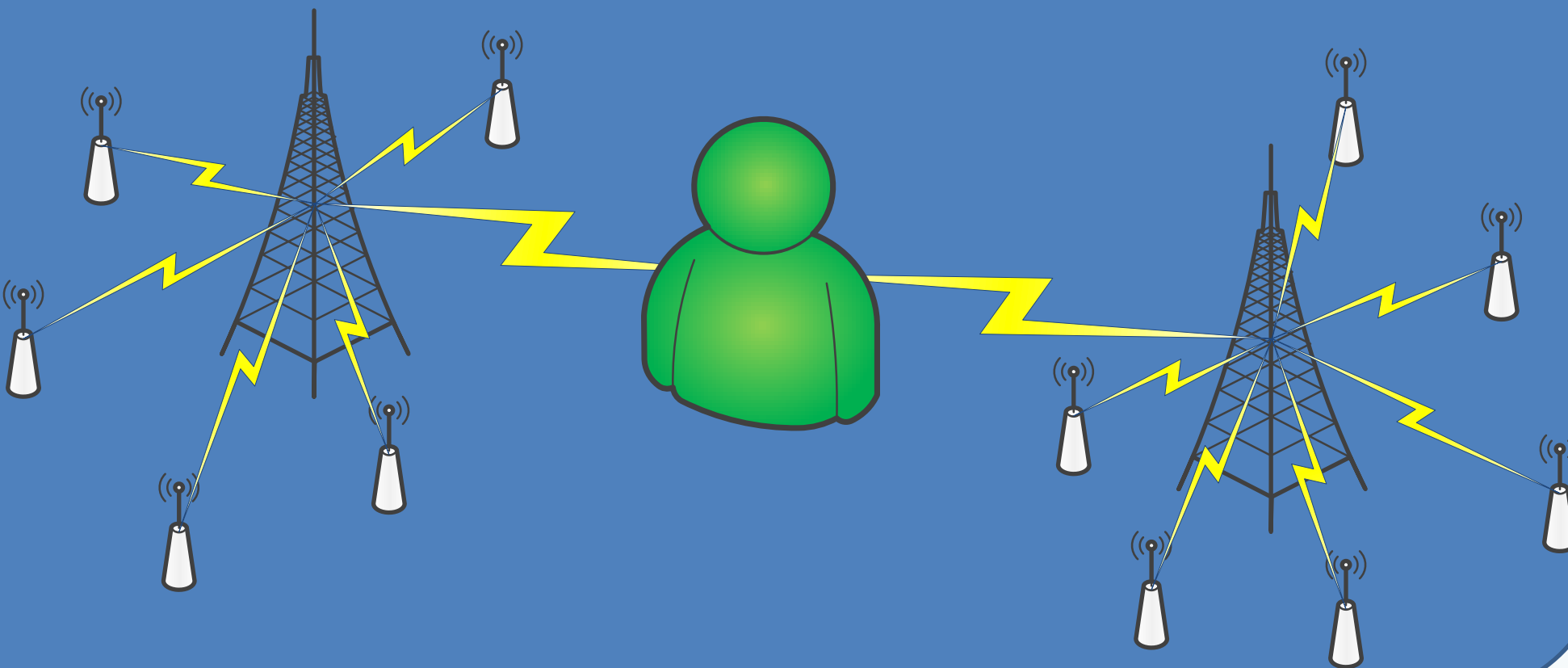


### Clustering

As two of the key constraints in a WSN are energy consumption and bandwidth, we made it a priority to minimise the number of messages sent through the network; clustering is perhaps the most widely-used strategy for achieving this. Using features suggested by the Low-Energy Adaptive Clustering Hierarchy (LEACH), we devised two clustering algorithms; a basic cluster for small networks (cf. the diagram below), and a hierarchical cluster (with variable depth) which can be scaled up arbitrarily.

The basic clustering algorithm simply elects all nodes within radio range of the sink as clusterheads (CHs); all other nodes send data to their closest CH which application-dependent processing before forwarding them on to the sink. Hierarchical clustering extends this by specifying some depth D such that, if the shortest discovered route to a node's CH is D hops, that node becomes a clusterhead itself.

In both cases, nodes will either send sensor data to their clusterhead, where aggregates and application predicates will be evaluated (e.g. average temperature over a cluster's area), or send the information requested by a predicate message (for example, checking that no node has two CHs). In accordance with the desire for energy efficiency, we also plan to test energy usage on the clustered network in Term 2.



## HSEND

H-SEND is a framework for detecting faults in WSNs, designed to minimise energy consumption. It differs from related algorithms by being capable of handling very large WSNs.

Our implementation allows the sending of a predicate message from a base station to a target node, who will then begin the evaluation process. This can involve several aspects, such as local predicate checking, or the evaluation of predicates the require information from multiple nodes.

With our implementation, we initially focused on evaluating fixed, compile time predicates and reporting the results back to the base station. This was further expanded to allow checking of compile-time predicates that require information from neighbouring nodes. Further abstractions of the framework will allow us to specify run-time predicates, and have the information relayed using the same underlying network stack.

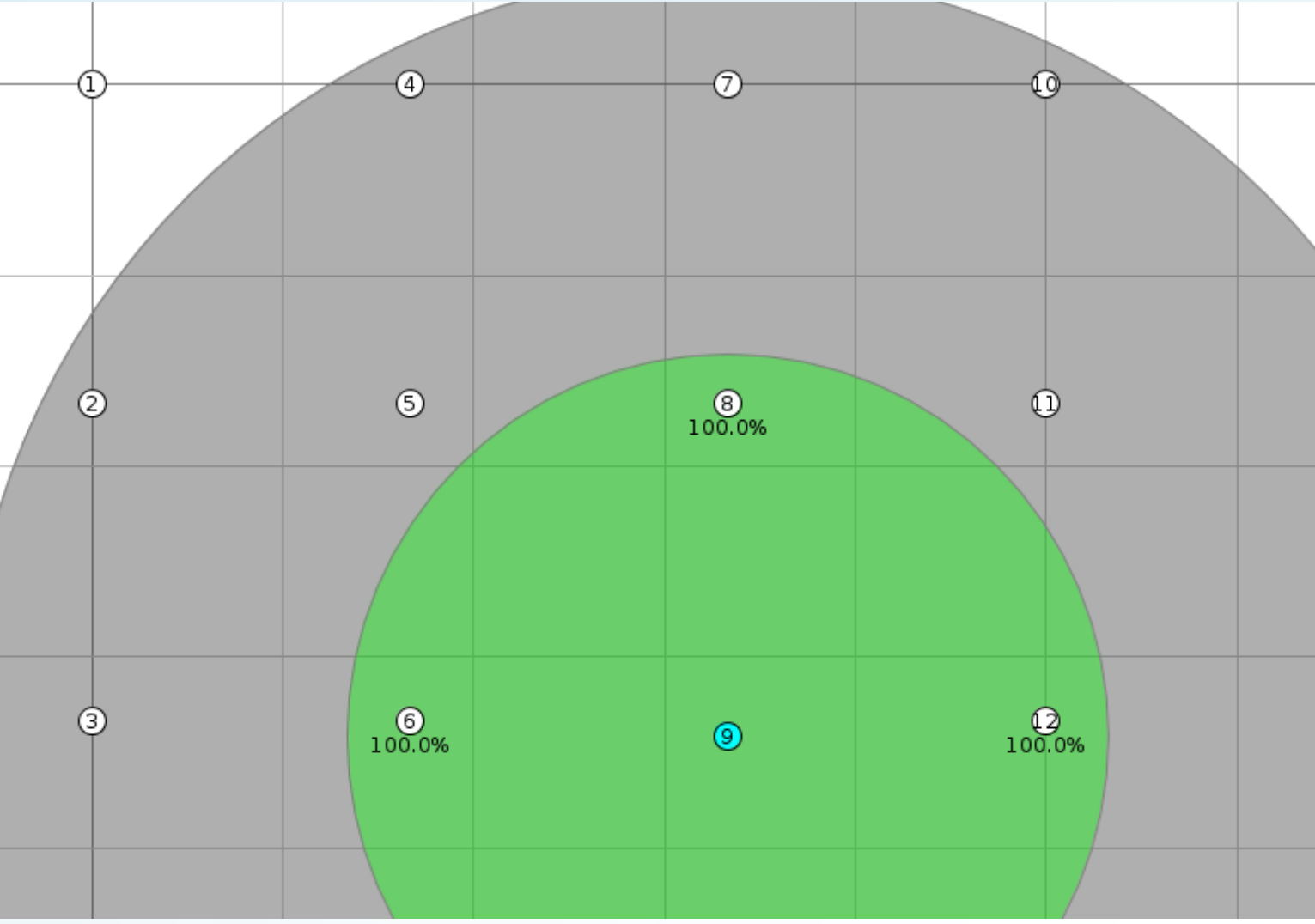
Further work will involve integrating the message sending with neighbour detection algorithms and clustering to allow for much more efficient message sending. This will involve researching where it is best to evaluate predicates in a network in order to minimise energy and time usage.

## Neighbour Discovery

A key requirement for a wireless sensor networks is reliably knowing who your neighbours are, this can be useful for a great many applications including among others clustering and predicate evaluation.

Contiki contains a built in module for neighbour discovery but it only handles half of the problem, communication. In order for each node to maintain a reliable list of it's neighbours, we decided to implement a wrapper module around the core Contiki module. Our module maintains a list of neighbour nodes and checks periodically to make sure that no new nodes have joined the network or that old nodes have left.

This module will be used as part of our underlying network stack with applications in n-hop predicate evaluation and network visualisation.



## Issues Encountered

We have not encountered too many unforeseen issues with the project so far. However those we have encountered were more problematic than we originally thought they would be:

1. Learning to work in a different development environment (different language, unfamiliar APIs)
2. Difficulty thinking and develop in a distributed non-error free environment (where messages may be lost)

## Second Term Plan

1. Predicate Specification
  - i. Improve run-time definition and evaluation
  - ii. Reduce energy usage
2. Visualisation Tool
  - i. Provide more information
  - ii. Better predicate checking integration
3. Performance
  - i. Investigate where the best place to evaluate a predicate is
  - ii. Perform performance testing using physical nodes

## References