Searching for the Minimum Failures that Can Cause a Hazard in a Wireless Sensor Network

Iain Bate[1,2]
[1] Mälardalen Real-Time Research Centre
Mälardalen University
Västerås, Sweden
iain.bate@cs.york.ac.uk

Mark Louis Fairbairn[2]
[2] Department of Computer Science
University of York
York, UK
mlf@cs.york.ac.uk

ABSTRACT

Wireless Sensor Networks (WSN) are now being used in a range of applications, many of which are critical systems, e.g. monitoring assisted living facilities or for fire detection systems which is the example used in this paper. For critical systems it is important to be able to determine the minimum number of failures that can cause a hazard to occur. This is normally a manual, human intensive, task. This paper presents a novel application of search to both the WSN and safety domains; searching for combinations of failures that can cause a hazard and then reducing these to the minimum possible using a combination of automated search and manual refinement. Due to the size and complexity of the search problem, a parallel search algorithm is designed that runs on available compute resources with the results being processed using R.

Categories and Subject Descriptors

H.4 [**Performance of Systems**]: Reliability, availability, and serviceability; Performance attributes; Fault tolerance

General Terms

Performance

Keywords

stress-based testing; search-based testing; safety and dependability; wireless sensor networks

1. INTRODUCTION

Wireless Sensor Networks (WSN) are a fast growing area of research and are now being commercially deployed. Systems are normally composed from many simple nodes that autonomously deliver the overall functionality needed for the application. Uses of WSNs include, assisted living facilities to monitor patients, fire detection systems in building or forests, or within oil refineries to monitor stages of production. A typical deployment would consist of tens, if not hundreds, of autonomously-operating nodes. The combination of humans being central to the system, their mission critical nature and the fact functionality is delivered largely by software-based processing make them a classic example of a Cyber Physical System (CPS) where a complex computing system provides essential services to its human operators in order to make the operators more efficient and effective. The essential nature of the services extend beyond merely the functional or real-time properties but also encompass safety and dependability. At the same time there is growing evidence that deployments of WSNs have dependability issues [21, 22, 6].

WSNs use adaptive communications protocols which establish reliable ad-hoc networks, and minimise energy usage by reducing the number of message hops and aggregating information into fewer packets. The result is individual nodes having complex software in terms of how they are written and the algorithms encoded. These algorithms include signal processing from the sensors, data aggregation based on data being fed in from a range of nodes, and systems software including the operating system and network stack. Due to the low resources available, these algorithms tend to be written in a highly optimised device-specific manner. Clearly their behaviour is dependent on their software. However added difficulty is introduced as they are also dependent on, and interact with, other nodes in the WSN, as well as other wireless devices and physical objects in the collective shared environment. To this end, it is important to understand how many node failures can lead to the WSN not delivering the expected functionality. It is also important to know the physical area in which the failures may occur, i.e. there should be no more than M failures within area N. This information allows the designers to decide whether the design is sufficiently dependable, i.e. reliable or available. The designer can then either add more nodes or make the individual nodes more reliable. We only need to know why the individual nodes fail if there is a need to understand the cause of a failure, i.e. to prevent it having effect in the future [11].

There are two key approaches to understand the safety of systems, analysis and testing. Static analysis can work well for more constrained systems in terms of the design principles used and the configurations deployed [12, 2]. However even where these are highly-automated, they still require a model of the system to exist and then the results from this model are validated against the final system. It is our thesis that using search to manipulate the failures within a simulation can find similar results. Simulated models are used

GECCO'13, July 6–10, 2013, Amsterdam, The Netherlands.

as large-scale evaluation is not possible on "real" WSNs and for every trial (i.e. move of the search) all the nodes may need re-programming, and trials would consume much of the already limited power. The simulated models have been shown to be close to reality [5], however any inaccuracies do not matter to the results obtained. The reason is that we aim to determine the minimum number of node failures that can lead to a hazard without trying to understand what might have cause the nodes to fail. Of equal importance to the fact any inaccuracy should have a negligible effect is that the results obtained using a simulated model can easily be validated.

Even though simulation is easier to perform than large-scale trials with real hardware, each simulation can still take minutes, if not hours, [16] making search more difficult. Even though, for a particular system design, determining the minimum number of failures does not need to be determined often, there are still benefits that this is done as quickly as possible. One such benefit is the possible evaluation of many proposed designs. For this reason, a parallel search approach is proposed. This has the secondary advantage that the risk of premature convergence can be reduced by each of the individual searches swapping partial solutions at appropriate points in time [9]. It is assumed that most organisations have a reasonable number of computers that can be used to perform the simulations, e.g. as part of their background processing using frameworks like BIONC (http://boinc.berkeley.edu), or there are plenty of relatively cheap cloud computing facilities, e.g. Amazon EC2 (http://aws.amazon.com/ec2/). However these "commodity" solutions tend to larger variations in completion time due to the need to fairly share the resources across users or machines. This necessitates a parallel search algorithm where the individual search algorithms are not too tightly coupled.

The contributions of this paper include:

1. Presenting a search-based method for WSNs to determine the minimum number of failures that could lead to hazards.
2. Defining the size and complexity of the search problem.
3. Designing a search algorithm that can deal with these complexities despite the evaluation being time intensive.

The structure of the paper is as follows. Firstly, our approach to making WSNs dependable is presented including an example application that is to be used throughout this paper. Next, the search framework is designed showing the choices of the search algorithm itself, the fitness function that guides the search towards the desired solution, and the moves that the search algorithm makes to achieve an efficient and effective solution. Finally there is an evaluation of the approach followed by conclusions.

2. THE USE OF DEPENDABLE WSNS

The purpose of this section is to do the following: provide an overview of how WSNs are used as part of dependable systems; a description of the fire detection system used in this paper; and details of our approach based on the use of Health Monitoring (HM), specifically Safety Policy Monitoring (SPM).

2.1 WSNs for Dependability

An important feature of WSNs to consider is that nodes have highly restricted resource availability, owing to the requirement for small physical size and low unit cost. Limited availability of energy resources are particularly significant. When a node runs out of energy it cannot participate in the WSN. Eventually a state is reached in which there are insufficient active nodes for the distributed application to function correctly; at this stage the entire WSN is effectively useless for the application. WSN designs must therefore be sufficiently optimised to ensure correct operation for at least the specified operational lifetime. One approach that may be taken is to make them reactive. A reactive system will only transmit messages when there is something of interest to communicate. By contrast, a traditional approach taken in hard-wired systems would use many more messages and hence energy to provide similar functionality, due to the sensors continuously reporting back to the application. A quandary for the designer is that a silent network could indicate nothing of interest is happening or that the network has failed, e.g. due to a lack of energy. Therefore appropriate means, e.g. HM, are need to ensure that the network can still function correctly should the need arise that balances resource usage against the need to know if the WSN is still capable of delivering the intended application semantics [21]. The purpose of HM is to periodically check key aspects of the WSN's operation in order to ensure dependability [1]. In this paper, the attributes of dependability of considered are; reliability, availability, maintainability and safety.

One technique to improve the dependability is to use HM. HM approaches are split into centralised systems such as MANNA [14], and distributed systems where typically decisions are made based upon the local neighbourhood, with the global network improving as a more emergent behaviour. The main issue with these approaches is they simply detect failed nodes and do not check the WSN's ability to deliver the expected application semantics. A consequence of this is that maintenance may be ordered before it is needed, for example if sufficient replicas still exist to report information.

Wu et al [21] proposed RTA, which aims to verify the application level semantics of the system by simulating certain behaviours in the WSN. The approach is motivated by WSNs often being reactive which means for long periods the WSN may be quiet indicating there is no sensed data worth reporting, however it could also mean that the WSN has failed. Wu used the example of a fire detection system. Firstly omission of a fire detection is the only hazard considered, and secondly maintenance is only ordered when the WSN can no longer detect a fire. In [4], an improved type of RTA was introduced that specifically targets the mitigation of hazards and hence improving the safety of systems. The approach is referred to here as SPM and the tests that are periodically performed Safety Policy Tests (SPT). SPM is based on the concepts of Safety Policies [15] and Safety Kernels [20]. SPM is used in this paper as it is the only known example of an approach targeting safety of WSNs.

2.2 Fire Detection System

The purpose of this section is to introduce a case study, a firefighting system which is used throughout this paper, and the WSN design that has been produced to deliver dependability.

The firefighting system [21] is intended for use on larger buildings, e.g. skyscrapers, with each room having a number of different sensors, e.g. temperature and humidity. Based

on these sensors a decision is made as to whether there is a fire in the building. Should a fire be detected then a signal is sent to a central monitoring station where information from all the rooms is then used to initiate and control appropriate firefighting measures. As part of the firefighting system it is envisaged the firefighters themselves will receive information on which rooms have a fire and in the case of evacuation what the best route out of the building would be. Other users of the information would be the buildings supervisor who would have responsibility for ensuring an operational fire fighting system, the people who have to maintain the WSN, and the fire chief(s) with responsibility for coordinating the firefighting. The reasons we feel this system is an appropriate example are as follows:

1. The system has clear hazards associated with it, for example not detecting a fire, not warning a firefighter of hazards within a room they are about to enter, or not providing or providing an incorrect egress route.
2. The system is sufficiently complex as it features heterogeneous sensors, data aggregation, communications that will be subject to interference and voids, and multiple categories of users. As such the failure conditions leading to the hazards are not straightforward.

In our WSN-based implementation of a fire detection system, there are five rooms. Each room has four nodes of which a minimum of two are needed to detect a fire. Each node would have a cpu with an application, network stack and a micro-kernel. The software has three distinct purposes. Firstly to sense the environment with some of the available sensors that each node might have, e.g. temperature or humidity, and then process it accordingly, e.g. to reduce the data size. The processed information is then sent to a sink node which collects it together, and then either processes the data or uploads it to remote computers for remote processing. This sink node is assumed to be hard-wired to the operators console, so is less likely to experience failures and is also mains powered. Secondly as most WSNs are multi-hop, a node commonly acts as a routing node. Its software is therefore responsible for receiving information from other nodes and forwards it to the next node en-route towards the sink. Unlike most traditional critical systems, WSNs operate in environments that are not well understood even after deployment and failures are more common place. These failures are more common due to node hardware being unreliable and also the possibility that new interference sources may be introduced during use. To counter these problems WSNs are often designed using adaptive algorithms and protocols which makes verification more challenging [17, 18, 19]. Finally it has system software which is responsible for scheduling, security, HM etc..

A hazard analysis was performed that determined a late fire detection is potentially unsafe, which leads to a DSR being specified. That is, DSR2 - *The fire detection system should detect a fire in a room within Z time units, where Z is dependent on the building's safety case and the need to evacuate the building within a given amount of time.* Therefore due to the reactive nature of WSNs, periodic simulated fire signals are raised at all nodes. These periodic signals are referred to as *Safety Policy Tests* (SPT). They allow the system to determine whether the DSR is in fact met. The test to cover DSR2 is to check whether any SPT message indicating a simulated fire has been delayed by more than Z time units from each and every node in the network. In the fire scenario Z is assumed to be 1 minute, as this is an early indication of either a node failure, node communications being dropped, or messages simply taking too long to reach the operator node. All of these should be handled by maintenance procedures. These failures could be for a large variety of reasons including: environmental noise effecting the chance of successful transmission, the number of hops to traverse the network, or the latency of other messages. The test looking for DSR2 failures is performed on the operator(s) consoles.

If problems are detected then maintenance is planned.

1. *Replacement maintenance* is scheduled if more than P nodes within a room have a permanent fail silent failure or a fail noisy failure indicating that the node is faulty and must be replaced.
2. *Communications maintenance* is scheduled by the presence of a DSR2 failure (transient fail silent failure) being raised on a node without the presence of a DSR1 failure (permanent fail silent failure). Should this be an isolated case then it's most likely a communication hardware failure, and thus the node should be replaced. If many nodes exhibit this behaviour then this form of maintenance does not physically replace the failed nodes, as this would not fix the issue, instead requiring that the network parameters are tuned to decrease the number of collisions, e.g. increasing communications jitter (decreasing the likelihood of a collision occurring during the sending of messages).

Further details on the methods by which the DSRs and SPTs are derived and used are contained in [8].

2.3 Assessing the Search Landscape

Each injected failure is considered to be within a specific node and is transient for a given period and given duration. Durations can be up to and including the length of the period at which point the failure becomes permanent. Each time a failure is triggered, it is assumed that with a given probability it will lead to a total failure of the node, i.e. the node will no longer sense information, communicate its own sensed data or act as a routing node for others. Where this does not happen it is assumed the failure is either not significant to these functions or it is tolerated. It is noted that other (not-injected) failures will occur as part of the simulation, e.g. communication failures due to packet collisions.

Variable	Min	Max	Step
Failure Period (hours)	1	4	1
Failure Duration (mins)	1	Period	1
Failure Offset (hours)	0	Period	0.25
Failure Probability (%)	0	100	5
Maintenance Period (hours)	2	8	2

Table 1: Variables to be Manipulated

The variables, their ranges and the step function used in this paper are described in Table 1. The step function assumes not all values might be chosen, i.e. a variable can take values min_value, min_value+step_value, min_value+2·step_value, ··, max_value. Without a step function, i.e. if all real values were assumed in the range, then the search space would be significantly larger. An important (related) issue is that the search landscape is not smooth. This means that search approaches, e.g. hill climbing, will struggle to find getting good solutions harder. Reducing the search size helps, however it is at the possible ex-

pense of the *best* possible solutions not being found. Initial trials show that too small a *step_value*, including not having a *step_value*, meant that none of the search algorithms could cause the hazard to occur within the time allowed. Too large a *step_value* meant the failure state of the nodes tended to be too regular, e.g. binary between not failed or permanently failed. This would be unrealistic for a WSN. The minimum and maximum values in Table 1 were chosen as they are realistic for the application. The step values were chosen, roughly following a binary search, to best show the differences in the algorithms.

The size of the search space is therefore 20!·307200·5, which equals $3.7\text{x}10^{24}$, based on the following.

1. There are 20! combinations of node failures that can occur if physical position is ignored. If physical position is taken into account then the number of combinations would be much larger, however 20! is sufficiently large to illustrate the significant complexity of the search problem.
2. There are $4 \cdot 240 \cdot 16 \cdot 20$ (=307,200) combinations for each node failure from the period, duration, offset and probability respectively.
3. The maintenance periods give 5 further options.

2.4 Simulation Used for Evaluation

To measure the fitness of a specific solution it was necessary to run a full simulation of the fire detection system using co-simulation between the application simulator and Network Simulator 2 (NS-2). The application was implemented using a custom built application level simulator which defines the behaviour of the nodes and the node failures. NS-2 (www.isi.edu/nsnam/ns/) was used to simulate the transport and lower network levels of the nodes as it is widely used, well tested and allows for high fidelity simulation. NS-2 was configured for a typical WSN application, Ad hoc On-Demand Distance Vector (AODV) protocol [13] for the transport layer, and 802.15.4 for the MAC protocol used in combination with the two-ray ground model for the physical radio model (a typical WSN setup). The failures injected into NS-2 are communications failures, whereby the node simply has its radio turned off, stopping the node communicating its own messages, or relaying messages for others. Further details on the simulation framework are in [8].

```
-PARAMETERS
SimulationTime, MaintenancePeriod
86400, 28800

-FAILURES
Type,NodeID,Probability,Offset,Period,Duration
1,4,0.350000,4560,14400,15840
1,19,0.950000,3840,14400,14880
```

Figure 1: Sample Search Solution

The failures are defined in each search solution, an example of a search solution is shown in figure 1. The figure shown in the figure defines the *SimulationTime*, which is not manipulated by the search algorithm, and the *MaintenancePeriod*. Then the failures are defined, with each row represent one failure. *Type* distinguishes between a comms failure (=0) and a node failure (=1). Node failures then have a *nodeID*, *Probability*, *Offset*, *Period* and *Duration*. Whilst the failure shown is simple, it represents much of the complexity of WSNs that are hard to analyse as discussed within in this paper such as sporadic interference or faulty hardware.

3. DESIGN OF THE SEARCH FRAMEWORK

The purpose of this section is to describe how the parallel search algorithm was implemented to manage the size and complexity of the search problem, how the fitness function was designed to guide the algorithm towards the desired solution, and the moves made by the search algorithm to efficiently and effectively transition the search landscape.

3.1 Parallelising the Search

Two main approaches of conducting the search were considered, genetic algorithms and simulated annealing. Genetic algorithms were considered good at dealing with large complex landscape and is easily parallelisable. Simulated annealing also has these characteristics as it is a local search algorithm. This means once it has got within the proximity of a good result, i.e. caused a hazard, it should be good at then exploring how to minimise the failures needed to cause the hazard. One of the benefits of a meta-heuristic search like simulated annealing is that, other than the definition of the neighbourhood, the heuristics for a good solution are not built into the algorithm but into the cost function. This makes it easy to move between goals such as achieving a load balanced system or minimising differences to an existing solution. One *individual simulated annealing algorithm* in the *parallel simulated annealing algorithm* is described in Figure 2 which is based on a standard algorithm from [10].

$\Gamma = \{\gamma_0, \ldots, \gamma_N\}$ /* Set of parallel simulated algorithms*/
$\omega^*(i) = \{\omega_1^*, \ldots, \omega_N^*\}$ /* Best configuration for each γ_i algorithms*/
$\Omega = \{\omega_0, \ldots, \omega_N\}$ /* Solution space */
$f : \Omega \rightarrow [0, 1]$ /* Cost function */
$\psi \in \Omega$ /* Initial configuration */
$\omega^* = \psi$ /* Best configuration for this algorithm*/
$\omega = \psi$ /* Current configuration */
$t = t_0$ /* Set initial temperature */
$\alpha = c_0$ /* Cooling factor */
$\lambda = T_0$ /* Cooling period */
$\theta = R_0$ /* Re-seed period */
$\phi = S_0$ /* Swap period */
do
  $i = 0$
  do
    if (j mod ϕ) = 0 **then**
      $\omega' = \omega^*(rnd(\Gamma))$
    else
      if (j mod θ) = 0 **then**
        $\omega' = \omega^*$
        $\omega' = modify\_config(\omega)$
      endif
      fork calculate_fitness_function(f(ω'))
      wait for all simulated annealing algorithms
      $\delta = f(\omega') - f(\omega)$
      R = random value $\in [0, 1]$
      if ($R < e^{-\frac{\delta}{t}}$) **then** $\omega = \omega'$ **endif**
      if ($f(\omega) \leq f(\omega^*)$) **then** $\omega^* = \omega$, $j = 0$ **endif**
      $i = i + 1$
      $j = j + 1$
    endif
  until ($i = M$ or (stopping condition))
  if (i mod λ) = 0 **then** $t = \alpha t$ **endif**
until (stopping condition)

Figure 2: Parallel Simulated Annealing

3.1.1 Parallel Simulated Annealing Algorithm

Parallelisation is achieved by executing multiple simulated annealing algorithms, as described in 2. Specifically, each

individual simulated annealing algorithm will **fork** the calculation of the fitness function every move and then it continues when it can **join** every individual simulated annealing algorithm (i.e. they all complete the calculation of the fitness function). The reason for doing this every move is any individual simulated annealing algorithm may wish to swap solutions at a particular move. This join does result in an increase in the elapsed time of the algorithm, however in our work this was not found to be significant and the computational resources were not wasted as they were free for other users to use.

An individual simulated annealing algorithm swaps in a new solution when no improvement has been found its current solution within 50 moves. The new solution is selected by taking the best solution from one of the other individual simulated annealing algorithms chosen at random. This new solution is then used as the current solution and the algorithm continues as normal. This value was found by trial and error, again approximately following a binary search. Twenty parallel versions were used as early trials found that the greater the level of parallelism the faster the completion, in terms of elapsed time, and for our cluster there was usually at least this many cores available. The individual jobs were executed via a Sun Grid Engine (SGE) upon which at least 20 cores should normally be available.

3.1.2 Individual Simulated Annealing Algorithm

For each individual simulated annealing algorithm, the initial temperature was set at 1000 with a cooling factor of 0.99. Cooling is performed every 10 moves and if no improved solution has been found within 25 moves the current solution is returned to the previous best. Each simulated annealing algorithm concludes when there has been a total of 250 moves. Future work could investigate the choice of search algorithms and perform parameter tuning of the algorithm, including what level of parallelism gives the most efficient and effective solution in terms of CPU time, and then trading this off against the desire to finish the search in the minimum elapsed time.

3.2 Evaluating the Fitness Function

In this work the fitness function has the usual purposes of guiding the search towards meeting the primary objective, which in this case is the hazard of an undetected fire, and then to achieve this with the minimum number of failures. Thus the fitness function has the following components. Each component is normalised to return a value between 0 and 1. This means that the overall cost value can also be in the range $[0, 1]$ as the final step is to sum the product of each component's result and its corresponding weight and then divide the result by the sum of the weights, shown in equation 6.

1. *Number of undetected fires* - This component maximises the number of undetected failures or hazards associated with a simulation. It is calculated to give a normalised value using equation (1). This objective is considered the most important. A challenge is to decide on the value for the $Maximum(\#undetected fires)$ - assuming that multiple fires at the same time in a single room is treated as one. The maximum is not known apriori. An upper bound equivalent to the number of rooms could be taken, however this has two distinct issues. Firstly it is unlikely that all rooms have an undetected fire so the value of g_1 may only take a very small part of the $[0, 1]$ possible range. Secondly each room could theoretically have more than one undetected fire, i.e. there is an undetected fire, the WSN is repaired and then there is another one etc.. Therefore the preferred approach, referred to as a *dynamic fitness function* is to set $Maximum(\#undetected fires)$ based on the maximum observed during the particular search. Our previous work, e.g. [3, 7], and initial trials showed the negative aspects of such a dynamic fitness function were found to be better than the bias found with an un-normalised fitness function.

$$g_1 = \frac{\#undetected\ fires}{Maximum(\#undetected\ fires)} \tag{1}$$

2. *Number of node failures* - This component attempts to minimise the number of nodes required to achieve a hazard. Its value is zero unless a hazard occurs. Otherwise it is calculated using equation (2). Again, for similar reasons to equation (1), a dynamic fitness function is used. The purposes of the value found with equation (2) are to fulfil the secondary objective, of minimising the failures that cause the hazard, but also to moderate the above objectives which would otherwise drive the search towards failing all nodes in the network in a permanent fashion. The designer may be interested in inspecting failure combinations, other than the minimum node failure, that lead to a hazard to see if any improvements in the design are needed, in which case the weight of this component could be raised. An important point to note is that the fitness function has been encoded with no knowledge of what failure combinations could lead to a hazard, i.e. the nature of the likely best solution is not included.

$$g_2 = \begin{cases} \frac{Maximum\ failures\ (=20) - \#Failures)}{Maximum\ failures} & \text{, if } g_1 \geq 1 \\ 0 & \text{, otherwise} \end{cases} \tag{2}$$

3. *Number of SPT failures* - The purpose of this component is to maximise the number of failed SPTs in order to guide the search towards situations where the primary objective is met as a SPT failure indicates that we are starting to lose more nodes (but we may currently still be able to detect fires). A larger number will indicate an unhealthy network. To normalise the value the actual component is calculated using equation (3). Again, for similar reasons to equation (1), a dynamic fitness function is used.

$$g_3 = \frac{\#SPT\ Failures}{Maximum(\#SPT\ Failures)} \tag{3}$$

The cost function f is calculated from the scalar product of a vector of the cost function components and a weightings vector.

$$\vec{g} = (g_1, \ldots, g_n)^T \tag{4}$$

$$\vec{w} = (w_1, \ldots, w_n)^T \text{where } w_i \in \mathbb{R}\ \forall\, i \tag{5}$$

$$f = \frac{\vec{g} \cdot \vec{w}}{\sum_1^n w_i} \tag{6}$$

The weights, including the reasons for their relative sizes, are given below. Part of the reason for the ratios between the weights are the relative values of the fitness function

elements, which tended to have quite different ranges despite being normalised.

1. w_1 was given a value of 10 which makes it more important than the number of SPT failures but less important than minimising the number of hazards.
2. w_2 was given a value of 100. Remembering that g_2 only has a non-zero value when a hazard has been found, then the concentration of the search moves to minimising the value of g_2.
3. w_3 was given a value of 1 as it was least important

3.3 Moving Through the Search Landscape

To move from one solution to another, a randomly modified version of the existing solution is chosen by the function $modify\_config()$. If a move takes the parameters outside of the ranges defined in Table 1 then no change is performed. A move is taken by randomly choosing a single action from the following list.

1. A node can be added with the random failure characteristics based on the ranges in Table 1.
2. A node failure can be removed.
3. A randomly chosen node can have its period increased or decreased by one hour.
4. A randomly chosen node can have its duration increased or decreased by one hour.
5. The maintenance period can have its duration increased or decreased by one hour.
6. The probability of a randomly chosen node failure can be increased or decreased by 5%.

An initial solution for each individual simulated annealing algorithm is generated as follows:

1. A random number of node failures in the range [1,20]
2. Each failed node has the following characteristics:
   (a) A random integer failure period in the range [1,12], all time units are hours
   (b) A random integer failure duration in the range [1,12] but not greater than the period
   (c) A random probability in the range [0,100] selected at multiples of 5, i.e. 0, 5, 10, ··, 100. Each time a node is due to next fail, the probability is used to decide whether an actual failure occurs, i.e. a random value is generated in the range [0,100] and if its less than or equal to the probability then it occurs
3. A random integer maintenance period in the range [4,24]

4. EVALUATION

The objective of this evaluation is to understand the feasibility of the proposed search-based approach in the context of the original motivating problem: finding the minimum number of failures that can lead to a hazardous fault. These requirements are expressed as three experimental questions:

1. **EQ1** - Is it possible to search for failure conditions that lead to a hazardous fault?
2. **EQ2** - Can the minimum number of failures be found?
3. **EQ3** - How does the approach compare to other possible approaches?

For this problem it is known that a hazard may be detected, subject to there being a real or simulated fire, when there is not a sufficient number nodes left in a single room to detect the event. This means there must be less than two nodes for the hazard to occur. If the starting position is four nodes per room, then the *best* solution (i.e. a hazard occurs with the minimum number of node failures) is three failures within the entire network as any more is unnecessary.

4.1 Experimental Method

The experimental method employed is to run fifty trials for each of the three following methods:

1. *Parallel simulated annealing algorithm* - as described in section 3.
2. *Sequential simulated annealing algorithm* - The parallel simulated annealing algorithm described in section 3, however instead of twenty parallel searches there is only one. This is equivalent the sequential version of simulated annealing in Figure 2. This is aimed at providing an understanding of whether the parallel search is making reasonable usage of the available resources. The only difference to the algorithm in section 3 is that different parameters are used. The ones that are different are the total moves is ($= 20 \cdot 250 = 5000$), the cooling factor ($= 0.995$) and the number of moves before the current solution is returned to the last best ($= 20 \cdot 25 = 500$). The parameters are chosen to give this algorithm the same number of moves as the parallel version and to maintain a higher level of divergence allowing for the larger number of moves. Clearly with this algorithm there is no swapping of best results between individual simulated annealing algorithms as there is only one
3. *Random search* - This again takes 5000 moves in order to give the same number of moves as the parallel simulated annealing algorithm. For each move it chooses a random solution in the same fashion as the initial solution for the simulated annealing algorithm. This search is sequential however this can be parallelised as each of the 5000 moves are independent

The metrics to be used for the evaluation are the fitness functions outlined in section 2.4 and the following:

1. $\#Haz$ - How many times each search method manages to find a hazard, i.e. an undetected fire, across each of the fifty repeated trials.
2. $\#FH$ - The number of moves to find the first hazard, i.e. an undetected fire. To give a reasonably fair comparison, the results for the parallel search are multiplied by 20 (To calculate the number of simulated evaluations performed).
3. $\#Mo$ - The number of moves to find the best solution. Again for the parallel search the number of moves is multiplied by twenty.
4. Cl - How close each approach gets to the *best* solution as shown in the equation below, where $trial$ is an individual trial, mf is the minimum failures that can lead to a hazard, and f_x is the number of node failures associated with trial x, and pt is the set of positive trials, where for each trial there is at least one undetected fire. In most cases mf would not be known, however in the case of this paper an exact value is known as it is when there are too few nodes in a single room to detect a fire. For this paper five rooms with four sensors in each is used, which means mf is equal to 3. The results for this metric are examined across a number of repeated runs using the median and inter-quartile ranges as well as the minimum and maximum values.

$$closeness_{trial} = \begin{cases} \frac{f_{trial} - mf}{mf} & \text{, if trial} \in pt \\ = 2 \cdot max_{\forall i \in pt} f_i & \text{, otherwise} \end{cases}$$

4.2 Experimental Results

The parallel search algorithm was executed on a SGE and the fifty repeated trials took 8 days. It is noted that this time was dependent on what other work the SGE was doing. The random and uniprocessor search algorithms were run concurrently, i.e. all 100 trials at once, on a 64 core linux machine with 128 GB of memory. These trials took 4 days, however again the machine was not solely dedicated to this task. The results were processed using scripts written in *R*.

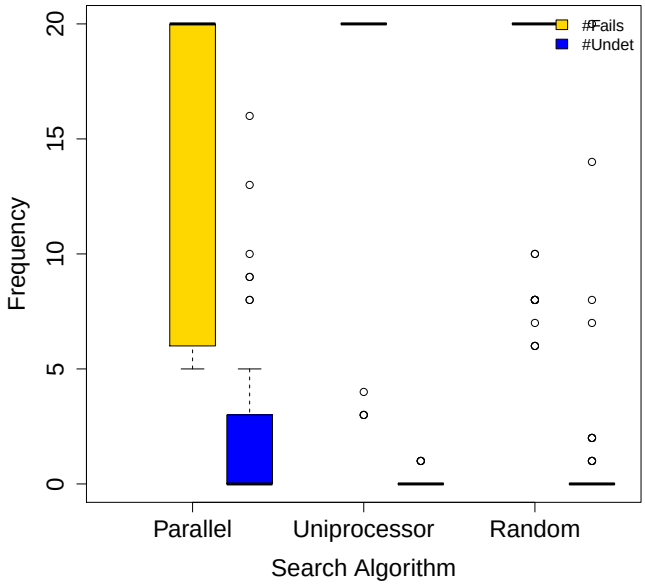

Figure 3: Comparison of Results

The results are summarised in Figure 3. The box plot shows the results for all three search algorithms against the three metrics introduced in section 4.1. A box plot is used as the box clearly shows the range of values between the upper and lower quartiles, the circles outliers, and the thicker black lines the minimum and maximum values.

1. *#Fails* - This indicates the minimum number of failures for which an undetected fire occurred. Where no undetected fire was found, the value of 20 is given as this is equivalent to every node failing. The box plot shows the parallel search algorithm found a large number of situations, 17, where there was an undetected fire. The best result was to have an undetected fire with 5 failures. In contrast the uniprocessor only twice created the undetected fire hazard, however it did it with fewer failures. These are considered random chance. The random case created an undetected fire more times, five, however it was with more failures than the uniprocessor case. A further trial was run with 500 moves for the parallel simulated annealing algorithm instead of 250. This time an undetected fire was reached with only two failures, i.e. the minimum number, which shows the approach is successful. This situation was found after 353 moves. Overall the results suggest the problem is in fact complex to solve and the parallel search algorithm was more reliable at solving the problem. It also showed given enough cases random search could do well. If failures were made less likely however, it is unlikely to succeed.
2. **#SPT** - This is not shown in the graph as the non-parallel search algorithms' best solution often features many failures, e.g. all the nodes in a single room at the same time. These situations lead to extremely high numbers of SPT failures, e.g. 10K+, which would skew the scale of the graphs. At the same time these are not good solutions as the aim of the work is to reach the hazardous state of an undetected fire with minimal failures. It is noted that the non-parallel search algorithms only reached the hazardous state when there were many failures for long durations in the same room, however this was always for a larger number of failures.
3. **#Undet** - The parallel search algorithm's best solution often had fewer undetected fires. This is seen as a positive result as assessing the detailed logs showed that there were solutions with more undetected fires, however this was at the expense of more node failures. The search then reduced the failures at the expense of less undetected fires,

Algorithm	#Haz	#FH	#Mo	Cl
Parallel	86.4	794.8	3972.4	14.42
Sequential	8.62	513.54	186.04	18.5
Random	9.6	651.64	1802.02	16.58

Table 2: Summary of the Search Algorithms

Table 2 presents a summary of how each of the three search algorithms perform against the metrics outlined in section 4.1. The table gives the mean for each of them. The results show parallel search is significantly better than the other two algorithms in terms of the number of hazards found and the closeness to the best possible result. The time to find the first hazard (when one is found) for the parallel approach is a little worse than the other approaches. However as the figures reported are multiplied by 20, the results show given a parallel computing platform then there is an almost linear speed up in terms of elapsed time to find the first solutions. For hard to reach hazards and with the abundance of processing cores then this is significant. Further investigation is needed to understand the precise relationship between cores available and the elapsed time to the first hazard being found. The final metric, Mo, indicates the parallel and sequential approaches tend to find their best solution after the same amount of steps through the algorithm (or elapsed time assuming sufficient processing cores are available) remembering the figures reported are multiplied by 20. Analysis of the results support the earlier claim the parallel search is less likely to suffer premature convergence. The analysis also show the parallelism allowed more diverse parts of the search space to be examined.

4.3 Manual refinement of the results

Two methods of manual refinement were used with equal effect. The first method was manual inspection. For the system considered and a result close to the best possible solution, this approach is possible. However in more complicated systems, this will be more difficult and prone to errors. The second is consider combinations of failures in the best found solution by feeding them back into the simulator. The mechanism for this is outlined below. The algorithm is guaranteed to give the minimum number of failures, based on the best solution found, that lead to a hazard. As stated in section 4.1, for the system used in this paper the minimum number of failures is an exact known value.

1. Step 1 - N is initialised to 1
2. Step 2 - For each combination of N failures from the best found solution, simulate the system
3. Step 3 - if no hazard found then increase N by 1 and repeat Step 2

4.4 Summary

The experimental questions posed at the start of this section are now re-visited.

1. **EQ1** - Is it possible to search for failure conditions that lead to a hazardous fault?
   The result for this question is clearly yes.
2. **EQ2** - Can the minimum number of failures be found?
   The result for this question is possibly yes. The exact value was found. The parallel search algorithm was shown to be much more effective at both creating the hazards and doing so with fewer failures. The "best" result was a hazard with five failures. This was sufficiently few failures for the designer to determine that only two of these were in fact needed. Three of the failures were deleted from the configuration, the test re-run and the hazard still found. The more failures there are the harder this manual step becomes.
3. **EQ3** - How does the approach compare to other possible approaches?
   The results show that a parallel search algorithm outperforms both sequential simulated annealing and random searches in terms of ability to find solutions. As each simulation takes minutes, if not hours, to perform, then the benefits of parallelisation are significant. Future work could look at stress testing other DSRs, and at how different search parameters and algorithms could improve efficiency and effectiveness.

5. CONCLUSIONS

This work has shown search-based testing can obtain a good estimate of the minimal failures that can cause a hazard. This technique can be used ahead of deployment allowing the designer then to make targeted (cost effective) changes to the WSN to make it more dependable. The designers can also use the results to validate the findings on the real system as once the situations have been established they would need less work to check. Parallel search was used which had the dual benefit of speeding up the evaluation and improving the results. Our belief is that the techniques proposed here are useful for more general problems related to WSNs and other complex software-based systems. Future work will examine whether this is true.

Acknowledgement

We thank Tiong Hoo Lim for his help analysing the results. We acknowledge the Swedish Foundation for Strategic Research (SSF) SYNOPSIS Project for supporting this work.

6. REFERENCES

[1] A. Avizienis, J.-C. Laprie, and B. Randell. Dependability and its threats - a taxonomy. In *Proceedings of the IFIP 18th World Computer Congress*, pages 91–120, 2004.

[2] I. Bate and A. Burns. An integrated approach to scheduling in safety-critical embedded control systems. *Real-Time Systems Journal*, 25(1):5–37, 2003.

[3] I. Bate and P. Emberson. Incorporating scenarios and heuristics to improve flexibility in real-time embedded systems. In *Proceedings of the 12th IEEE Real-Time And Embedded Technology And Applications Symposium*, pages 221– 230, 2006.

[4] I. Bate, Y. Wu, and J. Stankovic. Developing safe and dependable sensornets. In *Proceedings of the 37th Euromicro Conference on Software Engineering and Advanced Applications*, pages 279–282, 2011.

[5] D. Curren. A survey of simulation in sensor networks. Project report (CS580), University of Binghamton, 2005.

[6] J. Decotignie. The real-time and wireless sensor networks: are they compatible? Keynote at Euromicro Conference on Real-Time Systems, 2012.

[7] P. Emberson and I. Bate. Minimising task migrations and priority changes in mode transitions. In *Proceedings of the 13th IEEE Real-Time And Embedded Technology And Applications Symposium (RTAS)*, pages 158–167, 2007.

[8] M. Fairbairn, I. Bate, and J. Stankobic. Improving wireless simulation through noise modeling. In *Proceedings of the 9th International Conference on Distributed Computing in Sensor Systems*, page To Appear, 2013.

[9] S. Gordon and D. Whitley. Serial and parallel genetic algorithms as function optimizers. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, pages 177–183, 1993.

[10] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[11] J. Laprie. Dependable computing: concepts, limits, challenges. In *Proceedings of the Twenty-Fifth International Conference on Fault-tolerant computing*, pages 42–54, 1995.

[12] J. McDermid, M. Nicholson, D. Pumfrey, and P. Fenelon. Experience with the application of HAZOP to computer-based systems. In *Proceedings of the 10th Annual Conference on Computer Assurance*, pages 37–48, 1995.

[13] C. Perkins and E. Royer. Ad-hoc on-demand distance vector routing. In *Proceedings of the Workshop on Mobile Computing Systems and Applications*, pages 90–100, 1997.

[14] L. Ruiz, J. Nogueira, and A. Loureiro. Manna: A management architecture for wireless sensor networks. *IEEE Communications Magazine*, 41(2):116–125, 2003.

[15] J. Rushby. *Kernels for Safety?*, chapter 13, pages 210–220. Blackwell Scientific Publications, 1989.

[16] J. Tate and I. Bate. YASS: A scaleable sensornet simulator for large scale experimentation. In *Proceedings of Communicating Process Architectures*, 2008.

[17] J. Tate and I. Bate. Sensornet protocol tuning using principled engineering methods. *The Computer Journal*, 53(7):991–1019, 2010.

[18] J. Tate, B. Woolford-Lim, I. Bate, and X. Yao. Comparing design of experiments and evolutionary approaches to multi-objective optimisation of sensornet protocols. In *Proceedings of the 10th IEEE Congress on Evolutionary Computation*, pages 1137–1144, 2009.

[19] J. Tate, B. Woolford-Lim, I. Bate, and X. Yao. Evolutionary and principled search strategies for sensornet protocol optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 42(1):163–180, 2012.

[20] K. Wika and J. Knight. On the enforcement of software safety policies. In *Proceedings of the 10th Annual IEEE Conference on Computer Assurance*, June 1995.

[21] Y. Wu, K. Kapitanova, J. Li, J. Stankovic, S. Son, and K. Whitehouse. Run time assurance of application-level requirements in wireless sensor networks. In *Proceedings of the ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN 2010)*, pages 197–208, 2010.

[22] F. Zhao. Challenge problems in sensornet research. Keynote at NSF NOSS PI meeting and Distinguished Lectures at Johns Hopkins and Princeton, 2005.