

# *forward*

*a future of reliable wireless ad hoc networks of roaming devices*

## A Methodology for Assessing Performance

March 10, 2005

Date	Version	Comment
10-03-2005	1.0	Initial Release

## **Authors**

Nick Moffat, QinetiQ, [n.moffat@eris.qinetiq.com](mailto:n.moffat@eris.qinetiq.com)

Michael Goldsmith, Formal Systems (Europe) Ltd, [michael@fse1.com](mailto:michael@fse1.com)

Sadie Creese, QinetiQ, [screese@qinetiq.com](mailto:screese@qinetiq.com)

## Executive summary

This report forms deliverable D21 of the FORWARD project. It is the final Deliverable of Workpackage 5 *Basic Mechanisms for Assuring QoS in Ad-Hoc Networks*, superseding the originally planned Deliverables D13 and D16.

Workpackage 5 is aimed at establishing basic mechanisms for assuring quality of service in ad-hoc networks, a core component of Next Wave, future ubiquitous computing, environments. Tasks 5.1 and 5.2 addressed black-and-white questions of “correctness” for routing and distributed data replication respectively. The work reported here addresses the task of assessing “performance”, including how environmental factors affect “correctness” and “efficiency;” (the latter being issues such as the relative speed of restabilisation after network changes and quantifying the overhead cost in attempting to maintain a correct view of the network topology).

Some aspects of “correctness” were addressed in deliverables D3 and D9, but other aspects were deferred to the present deliverable: in many cases, for instance, it is not possible to guarantee that a protocol achieves its goals with absolute logical certainty; but it can be argued that the counterexamples will arise with negligible (or zero) probability. This report focusses largely on questions of probabilistic correctness. Formalising such arguments requires appeal to more complex mathematical models, and tool support has benefited from access to systems such as Birmingham University’s PRISM.

We describe a methodology we have developed for performance analysis, and illustrate some of its capabilities by using it to characterise the performance of GLS, an important part of the GRID protocol suite.

In Deliverable D3 we reported our investigations into the correctness of GLS bootup under ideal conditions: no message loss and no node movement. The performance methodology allows us to characterise GLS bootup in more realistic circumstances – when messages are lost with defined probabilities and when nodes are mobile. It characterises performance by calculating accurate minimum and maximum bounds on the probability of a specified condition being satisfied.

Our methodology makes use of the  $pCSP_M$  to PRISM translator developed within FORWARD and reported in Deliverable D14. It includes guidance on how to extend possibilistic models (written in  $CSP_M$  and generated by techniques such as those described in Deliverable D3) into probabilistic models (written in  $pCSP_M$ ) suitable for performance analysis. This guidance describes ‘bolt-on’ techniques that can be applied systematically and easily.

GRID has been a useful case study, providing a context in which to explore general performance analysis issues and to develop systematic modelling techniques. In the process, we have investigated the capabilities and limitations of the methodology. To a large degree, we have focussed on a particular type of property: correctness of the GLS bootup phase. This phase is relied on by GLS to establish location servers for each node at a particular subset of nodes across the network. The intended location servers are determined by the node identifiers and their locations. If bootup fails to establish the location servers correctly then the subsequent Query/Reply phase of GLS will fail to communicate messages between nodes correctly.

The results obtained serve to validate the approach. They make sense intuitively and they agree with performance predictions we made based on hand calculations.

Limitations of the methodology and of its application to GLS are identified. Future work is outlined that can address these limitations. Perhaps chief among these limitations is the state explosion problem, which remains a significant hurdle to overcome when model-checking systems of realistic size (many nodes, many locations, etc.). Some progress tackling this problem is reported here. A more comprehensive treatment will be possible once work within FORWARD Workpackage 7 has completed.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Purpose . . . . .	1
1.2	Correctness, Efficiency and Performance . . . . .	1
1.3	Structure of the Document . . . . .	2
<b>2</b>	<b>Beyond Correctness Analysis</b>	<b>3</b>
2.1	Motivation for Formal Analysis of Performance . . . . .	3
2.2	Available Model-Checking Technology . . . . .	5
2.3	Performance Analysis: Overview of the Approach . . . . .	6
2.4	Performance Analysis: Specific Modelling Techniques . . . . .	7
<b>3</b>	<b>Analysing the Performance of Grid</b>	<b>12</b>
3.1	Introduction . . . . .	12
3.2	An Overview of Grid . . . . .	12
3.3	Models of GLS . . . . .	15
3.4	Performance Analyses and Results . . . . .	20
<b>4</b>	<b>Limitations and Scope for Further Development</b>	<b>31</b>
4.1	Limitations . . . . .	31
4.2	Extending the Performance Analysis of GRID . . . . .	31
4.3	Further Development of the Methodology . . . . .	32
<b>5</b>	<b>Conclusions</b>	<b>33</b>
	<b>References</b>	<b>34</b>

**This page is intentionally blank**

# 1 Introduction

## 1.1 Purpose

This report forms deliverable D21 of the FORWARD project. It is the final Deliverable of Workpackage 5 *Basic Mechanisms for Assuring QoS in Ad-Hoc Networks*, superseding the originally planned Deliverables D13 and D16.

Workpackage 5 is aimed at establishing basic mechanisms for assuring quality of service in ad-hoc networks, a core component of Next Wave, future ubiquitous computing, environments. Tasks 5.1 and 5.2 addressed black-and-white questions of “correctness” for routing and distributed data replication respectively. The work reported here addresses the task of assessing “performance”, including how environmental factors affect “correctness” and “efficiency;” (the latter being issues such as the relative speed of restabilisation after network changes and quantifying the overhead cost in attempting to maintain a correct view of the network topology).

Some aspects of “correctness” were addressed in deliverables D3 and D9, but other aspects were deferred to the present deliverable: in many cases, for instance, it is not possible to guarantee that a protocol achieves its goals with absolute logical certainty; but it can be argued (as was done informally for Piconet formation in D3) that the counterexamples will arise with negligible (or zero) probability. This report focusses largely on questions of probabilistic correctness. Formalising such arguments requires appeal to more complex mathematical models, and tool support has benefited from access to systems such as Birmingham University’s PRISM. Such an approach is presented in the current report.

As with the earlier deliverables of this Workpackage, the work reported here has close links in both directions with the continuing verification-tool enhancements being developed within Workpackage 7. In many ways the current Workpackage is a natural consumer of the WP7 technologies. This report is just one example of tool development within WP7, and they will certainly be brought to bear on the later tasks. In return, the tool development has benefited from experience gained by application to the GRID protocol, reported here. The models of GRID generated in this Workpackage have formed a unifying strand linking the technological advances with their applications. Similarly, the challenges in moving from particular network configurations which are within the scope of exhaustive exploration to proofs of correctness in the general are addressed by the work on data-independent induction [3] in Task 7.2, and the large state-spaces involved in all the modelling here provide opportunities for compression and symmetry reduction in Task 7.3.

To a large extent, therefore, the work reported here should be viewed as a single step on the journey towards the project goals in this area.

## 1.2 Correctness, Efficiency and Performance

To appreciate how the work reported here relates to earlier work performed in Workpackage 5, and to the overall objectives of the Workpackage, it is important to recognise the distinction between correctness, efficiency and performance of algorithms in general, and of routing protocols in particular. The general definitions below are illustrated by examples that relate to routing protocols:

**Correctness.** Correctness is about whether the algorithm fulfils its essential function (e.g., whether packets are delivered successfully by a routing algorithm), and is typically considered separately from efficiency considerations, described below. Complex algorithms are designed so that their ultimate objectives (such as to route packets) are achieved by the combined efforts of simpler algorithms. It is often useful to understand the correctness of these components in isolation, as a staging post on the way to understanding overall correctness.

*In the case of routing protocols, we are ultimately interested in end-to-end correctness (i.e., whether packets are sent correctly from source to destination). A commonly considered intermediate form of correctness is the correctness of the routing tables stored at routers. An*

*effective strategy for demonstrating end-to-end correctness of routing protocols can be to demonstrate correctness of the routing tables, and to show separately that having correct routing tables implies end-to-end correctness.*

**Efficiency.** Efficiency means how much resource (time, bandwidth, etc.) is used by the algorithm. This can either be measured as a function of time (perhaps abstracted: a number of rounds, etc.) or with respect to some goal (e.g., how much time/bandwidth is needed to route a packet). Efficiency in the latter sense pre-supposes correctness (i.e., that the goal is indeed achieved). Again, it can be helpful to consider separately the efficiency of different parts of a complex algorithm.

*In the case of routing protocols, efficiency can mean how much bandwidth is consumed by control traffic when servicing a given pattern of communications (whether or not the communications are successful). It can also mean how much time/bandwidth is required to route packets successfully or to establish correct routing tables. Similarly to correctness, consideration of the efficiency with which routing tables are established can be a useful step towards showing end-to-end efficiency.*

**Performance.** Performance is a measure of how the Quality of Service (QoS) of the algorithm depends on its operating context (its environment), where QoS can include both correctness aspects and efficiency aspects. QoS may for example be the probability of a specified aspect of the system's behaviour being correct (such as the probability of a specified packet reaching its destination), or an expected amount of resource required for some task (such as the amount of control traffic, or bandwidth, needed to route a specified pattern of data traffic).

*In the case of routing protocols, it is important to understand how well they cope with possible node mobility or communications interference. These aspects of the operating context can cause loss of packets, and mobility may have additional deleterious effects. Mobility and interference both typically lead to some routing failure and to reduced efficiency.*

The analysis approach described in this report characterises performance using mathematically rigorous model-checking techniques. Task 5.1 (reported in Deliverable D3 [2]) and Task 5.2 (reported in Deliverable D9 [9]) both relied on model-checking techniques, but the approach proposed in this report goes further: Task 5.1 considered complete correctness (and, to a lesser extent, efficiency properties) under rather strong assumptions about node movement (none was allowed) and packet loss (there was none). In contrast, this report considers performance properties, focussing on the degree of correctness achieved when the operating context allows defined amounts of node movement and packet loss.

### 1.3 Structure of the Document

This report is structured as follows. Section 2 introduces how the verification methodology exploited for assessing correctness of such systems in Deliverables D3 [2] and D9 [9] can be combined with the technological advances described in Deliverable D14 [5] to give less black-and-white assessment of performance. Section 3 continues the analysis of our running example, the MIT GRID protocol suite [6]. Section 4 outlines the limitations of the current approach and the scope for future enhancements, and Section 5 draws together the lessons learnt in this strand of the research.

## 2 Beyond Correctness Analysis

Deliverable D3 [2] considered correctness properties (and, to a lesser extent, efficiency properties) under strong assumptions about node movement and packet loss: no nodes moved and no interference occurred. In contrast, this report considers performance properties, focussing on the dependence of correctness on context (i.e., focussing on variation of correctness). This section proposes a formal analysis approach aimed at understanding the effects of node movement and of packet loss on the correctness of routing algorithms. The next section applies this approach to the analysis of an important part of the MIT GRID protocol suite [6].

An advantage of the probabilistic approach we propose here is that, to a large degree, it extends the possibilistic approach reported in Deliverable D3 [2]. Much of the modelling effort invested in correctness/efficiency analyses of the type demonstrated in Deliverable D3 can be re-used in the approach proposed below.

First we motivate the application of formal model-checking techniques for performance analysis. Next we summarise the technology on which our approach is based. Then we outline the analysis approach itself and discuss techniques for modelling key aspects of an operating context.

### 2.1 Motivation for Formal Analysis of Performance

Quality of Service (QoS) of a system (algorithm, routing protocol, etc.) can mean the probability of a specified aspect of its behaviour being correct (e.g., the probability of a specified packet reaching its destination) or a proportion of a defined subset of the system's behaviour being correct (e.g., the probability that an arbitrarily chosen packet is delivered to its destination). QoS can also be a measure of 'resource-to-settle' aspects of behaviour; an example is how much time it takes a routing protocol to recover a correct view of the network topology. Successful use of a system can depend critically on QoS characteristics, so there is a need for performance analysis.

Industrial practice for assessing the performance of routing protocols often relies on simulation, emulation, and (field) testing. Simulation involves constructing a model and 'running' it to produce some executions representative of the modelled protocol. Emulation involves 'running' a distributed implementation of a protocol over a network of computers, with inter-node communications implemented as messages in the network. Field testing involves actually trying out an implementation in a representative operating environment. Simulation, emulation and testing are frequently considered in this order, because the later techniques apply to systems closer to actual implementations. These techniques can be effective, but they typically only explore tiny subsets of the possible system executions and reachable states.

Model checking has some advantages over simulation, emulation, and testing. In the domain of routing protocols the key advantages of model checking manifest as follows: 1) it can assess some aspects of a routing protocol more dependably than simulation/emulation/testing, because it explores the whole statespace; and 2) it can assess some aspects of all faithful implementations of a protocol, reducing the problem of verifying that a particular implementation will behave as desired to the easier problem of verifying that it is faithful to a protocol definition.

On the other hand, model checking is limited to small models (e.g., routing protocol models configured with few nodes, no or little movement, etc) because it is exhaustive. Figure 1 contrasts the capabilities of model checking and simulation. The 100% coverage achieved by model checking comes at the cost of restriction to few nodes and few locations. Simulation typically considers many more nodes and locations but covers much less of the statespace. Fortunately, there are several techniques that enable model checking to be applied to large systems; these are discussed briefly below.

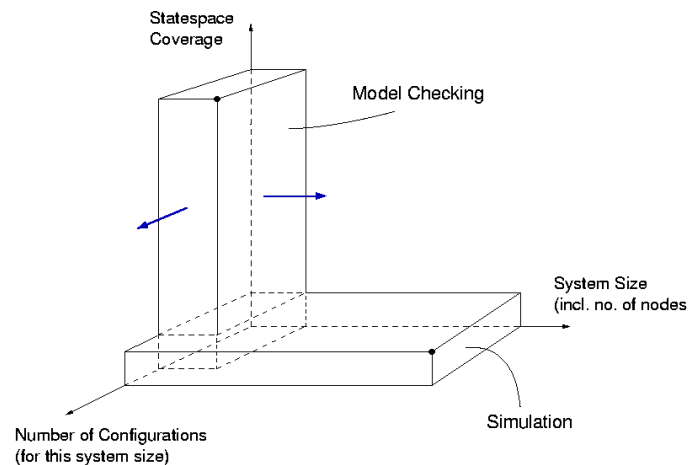


Figure 1: An illustration of the comparative capabilities of model checking and simulation. Model checking achieves 100% statespace coverage of the statespace for small systems (few nodes, few locations). Simulation achieves very small coverage of relatively large systems (many nodes, more locations). Both techniques can assess multiple configurations (initial locations, degree of interference, etc.), by running multiple model checks or multiple simulations. The thick arrows represent the existence of techniques that can extend model checking to larger systems and more configurations, such as symmetry exploitation and inductive reasoning.

'System Size' and 'Number of Configurations' are shown on separate axes of Figure 1 and multiple configurations can be checked by both techniques, one configuration at a time. However, it is possible to model check multiple configurations together, in a single check, by modelling a notional start-up phase in which the configuration itself is chosen; the effect is to increase the size of the statespace to be explored, but it means that the number of configurations, too, can be increased by application of scaling techniques.

Model checking is restricted to small systems/few configurations because of the *state-explosion problem*: models of realistically sized systems are often intractable to explore, especially when modelled naively. Consequently, model-checking is often only applied to small systems in few configurations. Abstraction can alleviate this problem: use of abstraction can lead to smaller models than one can obtain naively, without sacrificing the ability to assess properties of interest. When systems are large in reality, only small configurations or highly abstract models can be explored exhaustively; for many systems this limits what can be demonstrated by model checking alone. In such cases, however, one can sometimes argue that results obtained for small or restricted configurations extend to more realistic configurations.

The state explosion problem has received much attention in the research community in the last decade, and progress continues to be made. Workpackage 7 of the FORWARD project has contributed here, as reported in Deliverable D5; Task 7.3 *Automation of FDR Compression Techniques* focussed on developing a 'watchdog transformation' to make FDR's state machine compression more efficient. D5 also reported longer term research aimed at exploiting symmetry when model checking. The challenges in moving from particular network configurations which are within the scope of exhaustive exploration to proofs of correctness in the general are addressed by the work on data-independent induction [3] in Task 7.2. When automated support is not available, it is sometimes possible to scale up the results obtained for small systems to large systems – perhaps arbitrarily large – by use of informal proof.

## 2.2 Available Model-Checking Technology

### 2.2.1 FDR

The Failures-Divergences Refinement (FDR) tool is a refinement-style model checker, developed and marketed by Formal Systems (Europe) Ltd., which is capable of checking  $CSP_M$  refinement assertions automatically.<sup>1</sup>

A refinement assertion of the form

```
assert SPEC [T= IMPL
```

asserts that the  $CSP_M$  process *SPEC* is *refined by* the  $CSP_M$  process *IMPL*, where refinement is considered according to CSP's *traces semantics*. These terms are explained more fully in the FDR documentation [4]. Essentially, refinement properties express that the set of behaviours of one process (the implementation process) is contained in the set of behaviours of another process (the specification process).<sup>2</sup> The specification and implementation process are possibilistic, in the sense that all transitions between states represent possibilities – there is no notion of transitions being followed with defined probabilities. FDR provides yes/no answers – e.g., a refinement holds or it does not. This is frequently sufficient for verification of critical system properties. However, there is often a need for less black-and-white analysis of systems, which is addressed by our use of PRISM.

### 2.2.2 PRISM

PRISM [7] is a probabilistic model-checking tool developed by Birmingham University. It is a powerful tool for checking a wide range of properties of probabilistic models written in the PRISM language.

As explained below, our methodology for performance analysis involves PRISM models of a particular type: MDPs (Markov Decision Processes). The methodology involves calculating only particular types of property: the minimum and maximum probabilities of certain state variables eventually becoming true. MDPs, and the properties we check of them, are explained more fully in Deliverable D14. Some examples of the use of PRISM in this way appear in Section 3.

MDPs can model nondeterminism as well as probability. A key point to appreciate is the distinction between PRISM's treatment of nondeterministic choices and that of probabilistic choices. The bounding probabilities on the  $pCSP_M$  refinement holding are calculated by minimising (or maximising) over all nondeterministic choices reached during exploration of the model, while taking account of the probabilities of alternative probabilistic choices. The following paragraph explains this in more detail.

In the absence of nondeterminism, PRISM calculates the probability that the property is satisfied by a randomly chosen path through the MDP, taking account of the probabilistic annotations on transitions. However, when nondeterminism is present this probability is not well-defined, because resolving nondeterminism can influence subsequent behaviour. Instead, guaranteed minimum and maximum bounds on the probability are calculated, the quantification being over all possible nondeterministic choices resolved during exploration of the model; these values will coincide when the PRISM model is deterministic. Each probability bound is the probability that the implementation process behaves in a way allowed by the specification process, minimised (or maximised) over all possible resolutions of nondeterministic choices that are reached. One can think of these bounds as being the probabilities obtained by scheduling the nondeterministic choices within the model either angelically or demonically. All this is explained more fully in the PRISM Manual [7] and in D14.

<sup>1</sup> Additionally, FDR can check whether  $CSP_M$  processes are deterministic.

<sup>2</sup> The exact meaning of 'behaviours' here depends on the type of refinement. The above form of assertion asserts traces refinement – refinement in the traces semantic model – which is denoted by the symbol  $[T=$ . The traces of a process express which sequences of the process's 'events' can occur in which orders. Traces refinement expresses that all sequences of events of the implementation are 'allowed by' the specification. FDR also supports automatic checking of richer notions of refinement – Stable Failures and Failures/Divergences refinement – that are capable of expressing availability and (non-)divergence properties.

A new feature of PRISM, made available in a recent release, is its ability to calculate expected costs. This offers the prospect of assessing richer notions of efficiency. Two examples are the expected time (number of rounds, etc.) needed to achieve a defined level of correctness, and the expected amount of control traffic.

### 2.2.3 pCSP<sub>M</sub> to PRISM Translator

Deliverable D14 [5] reports the development, within FORWARD, of a translator from assertions about models written in pCSP<sub>M</sub> – a probabilistic extension of CSP<sub>M</sub> – to the PRISM language, and in particular to MDPs (Markov Decision Processes). For technical reasons the specification process is constrained to be possibilistic; however, the implementation process can be probabilistic.

The translator from a pCSP<sub>M</sub> assertion to a PRISM model (an MDP) has been designed such that the results obtained using PRISM are minimum and maximum bounds on the probability of the asserted refinement property being true for a randomly chosen trace of the implementation process. In order to give appropriate guidance about how to construct suitable pCSP<sub>M</sub> models of algorithms, it is necessary to explain a little about the various forms of choice available in pCSP<sub>M</sub>.

CSP has two forms of ‘choice’ operator: internal choice and external choice. They both model a choice between alternative processes; the difference is that internal choice is resolved by the process itself and the external choice is resolved by its environment. The environment of a process includes all processes with which it is put in parallel. When a process is composed in parallel with others, external choices can become constrained by removing or internalising them: choices are removed when the environment can refuse them, and internalised when the environment can itself choose between them internally. When modelling in CSP, external choice is often thought of as ‘deterministic’ (since the environment of the process can determine/control how the choice is made) and internal choice is often thought of as ‘nondeterministic’ (since the process will appear to resolve the choice itself) but this terminology should be used with care in the context of pCSP<sub>M</sub>. The pCSP<sub>M</sub> to PRISM translator treats all hanging external choices – external choices not removed or internalised – as nondeterministic, along with all internal choices.

PRISM’s treatment of nondeterminism has a direct influence on the modelling techniques we have used. Because minimisation/maximisation occurs over all nondeterminism in the PRISM model, and because this nondeterminism derives from nondeterminism and hanging external choices in the specification or the implementation, the calculated probabilities represent the best/worst chances of the implementation satisfying the specification *when all nondeterministic choices and hanging external choices are completely controlled by the environment*.<sup>3</sup> In contrast, the probabilistic choices (whether or not they represent activity of the environment) have defined probabilities of occurrence, which are factored into the calculated probability.

This gives rise to a simple rule: use probabilistic choice (obviously) when analysing situations where the feature modelled occurs with a known, or hypothesised, probability; otherwise, treat its occurrence, or not, as entirely controlled by the environment (and possibly even dependent on factors not modelled) by using nondeterministic choice, or hanging external choice. The obtained probability bounds will be the best and worst chances of the algorithm behaving as specified in the modelled environment.

## 2.3 Performance Analysis: Overview of the Approach

Our proposed performance analysis methodology is as follows:

1. Model the algorithm and formulate a pCSP<sub>M</sub> refinement assertion;
  - (a) Choose an appropriate semantic model for the property being analysed;

---

<sup>3</sup>Classically, nondeterministic choices represent decisions that are not within the control of the environment so we use ‘environment’ here in a powerful sense. Effectively, we obtain results that hold for all (classical) environments and all possible implementations and scheduling strategies.

- (b) Write a *probabilistic* model of the algorithm, as a  $pCSP_M$  implementation process;
  - (c) Write a *possibilistic*  $CSP_M$  specification process;
2. Run the  $pCSP_M$  to PRISM translator on this assertion;
  3. Run PRISM on the output of the translator.

The main skill required by the methodology is to model the algorithm and formulate a refinement assertion, in Step 1. (Steps 2 and 3 are automatic.) The three activities constituting Step 1 can be performed in any order, but would typically be performed in the listed order or together. They are discussed in more detail below. Step 2 translates the  $pCSP_M$  assertion to a PRISM model. In Step 3, PRISM calculates bounding probabilities of an execution of the PRISM model satisfying a certain path property, as prescribed in Deliverable D14. The path property expresses that the execution is a counterexample to the asserted refinement.<sup>4</sup> One can then plot the bounds calculated by PRISM (or complements of them, to bound the probability of an execution conforming to the specification).

The choice of semantic model (Traces, Stable Failures, or Failures-Divergences) is often dictated by the type of property being assessed [8]. An important class of properties is the class of *safety properties*, which express that no state can be reached in which some undesirable event is possible. Safety properties can typically be expressed as Traces refinements. Another class of properties is the class of *liveness properties*, which express that the system will remain suitably responsive; an example is the property that a sent message is eventually delivered. Liveness properties can be checked using Stable Failures refinement and Failures-Divergences refinement.

When a (possibilistic)  $CSP_M$  model of the algorithm – a  $CSP_M$  implementation process – is available, it is beneficial if the  $pCSP_M$  model can be an extension of it. One clear benefit is the potential for reduced modelling effort, through code re-use. We believe there may be other benefits, including modularity (separation of concerns into getting the possibilistic modelling right and then getting the probabilistic modelling right) and validation (there may be opportunities to validate the probabilistic results against possibilistic ones). For similar reasons, we believe it will frequently be beneficial to base possibilistic  $pCSP_M$  specification processes on the  $CSP_M$  specification processes used in a possibilistic analysis. Indeed, it may be that the same specification process should be used for both types of analysis. These issues should become clearer, leading to better guidance, through continued application of the methodology.

Deliverable D3 demonstrated how key possibilistic features of an algorithm can be modelled using  $CSP_M$ . The following section discusses how existing possibilistic implementation and specification processes can be extended to probabilistic processes suitable for performance analysis.

## 2.4 Performance Analysis: Specific Modelling Techniques

This section gives some advice about how to develop a probabilistic implementation process, and an accompanying specification process, from possibilistic versions. We advise how to model three important aspects of systems and their environments: message loss, mobility and random initial state. Message loss and mobility frequently have large effects on performance of distributed algorithms, and random initial configuration of a system must be considered when assessing average or expected performance. For the reasons explained above, we prefer these aspects to be modelled using bolt-on extensions to existing possibilistic  $CSP_M$  specification and implementation processes or by minor changes to existing modelling code.

The section begins with a general discussion of how to go about modelling systems, focussing on when probabilistic techniques are useful. It then discusses transformation of the possibilistic models to probabilistic ones that model message loss, random initial state and mobility. It ends with a discussion of efficiency analysis.

<sup>4</sup> For technical reasons, it does not appear possible to express the negation – that the execution complies with the specification. However, it is a simple matter to complement (subtract from 1.0) the probabilities calculated by PRISM, to obtain bounding probabilities of a randomly chosen execution *not* being a counterexample. Future work will either express the desired property directly using PRISM, or map the PRISM results as outlined.

### **2.4.1 A General Discussion**

This section provides some general guidance on how to go about modelling systems and when probabilistic techniques are useful.

The phenomena of interest (e.g., message loss) and their direct causes (e.g., interference and movement out of range) must be identified. Of course, the causes themselves are then 'phenomena of interest' and one should consider whether to model their causes. Ultimately, one continues looking for causes in this way until a complete set of phenomena have been identified that allow useful analysis. Essentially, one must consider whether the properties of interest can be assessed effectively in terms of this set of phenomena.

Where causes are modelled, the causal relationships should be identified/hypothesised as far as is practical; one should then decide whether to model these relationships deterministically (i.e., causally), nondeterministically, or probabilistically. When making this decision, one should consider the importance of the causes, and how accurately they should be modelled; initial hunches can be confirmed later when the analysis is performed. By definition, deterministic modelling is only possible if all the causes are modelled (at the level of abstraction of the model, and ignoring factors that are out of scope of the analysis). It often happens that some causes are not modelled, in which case it is necessary to resort to nondeterministic or probabilistic modelling. Nondeterministic modelling amounts to modelling that certain behaviours might, or might not, arise. Probabilistic modelling can be thought of as a half-way house between definite causal relationships and nondeterministic ones; it models events occurring with certain probabilities. Causal and non-deterministic modelling come within the realm of possibilistic modelling, so one can use the techniques outlined in Deliverable D3, for example. We focus on probabilistic modelling here.

So, when physical causes are modelled there is an opportunity to model their effects deterministically (i.e., causally), but when they are not modelled there is a need to model their effects non-deterministically or probabilistically. (This may also be necessary to avoid the model becoming too complex to write or to explore.) When modelling probabilistically, the probabilities will in general be with respect to certain defined quantities; examples include the probability that a message is lost *in a given time period* and the probability that a node moves *to a distant location*.

We assume below that possibilistic implementation and specification processes are available. When describing the modelling techniques, we consider how to obtain a probabilistic implementation process (expressed in  $pCSP_M$ ) from a possibilistic one (expressed in  $CSP_M$ ) and what changes to make, if any, to the possibilistic specification process.

### **2.4.2 Modelling Message Loss**

One should first decide on the type of message loss to be modelled by the analysis. Questions to consider include which types of message can be lost, under what circumstances, and with what probabilities. When there are broadcast/multicast messages it must be decided whether these can sometimes be received by some intended recipients and not by others.

Perhaps the two most common reasons for message loss are interference and movement out of transmission range, but there are several other reasons. The remainder of this paragraph accords with the general discussion above. If such physical aspects of a system are modelled directly then there is an opportunity to model the dependence of message loss on them deterministically (i.e., causally). However, when physical causes are not modelled message loss must be modelled non-deterministically or probabilistically. As is the case with any other type of activity, message loss probabilities will in general be with respect to certain defined quantities; e.g., the probability that a message is lost *in a given time period*. The discussion below assumes that the terms in which these probabilities are expressed have been identified.

It is usual to have events in a  $CSP_M$  model that represent the reception of messages. A naive way to model probabilistic message loss would be to make these 'reception' events probabilistic. However, this has an unfortunate consequence when the event also represents other activity: it models all such

activity as probabilistic, not just the receptions. A common example is when sending and reception of messages are represented by the same events – synchronous modelling of communication. In this case, making the receptions probabilistic simply by making these events occur probabilistically would also mean the sends are modelled probabilistically. Similar problems arise when multicast/broadcast messages are modelled by single reception event for each send: here, the naive approach does not allow some of the target nodes to receive a message that others do not receive.

Therefore, we recommend an alternative way of modelling message loss, which is simple and yet works even when reception events serve other purposes. The idea is simply to insert a probabilistic choice immediately following the reception event, wherever this event is interpreted locally as a reception. Note that this same event might elsewhere be interpreted as something other than a reception – e.g., a send; in such places no change is made to the model. In the synchronous communication example, above, sends are modelled as occurring just as they do in the possibilistic case *whether or not they are later deemed to arrive.*) The probabilistic choice should be between ‘accepting’ the message (modelling that it has indeed been received) and ‘dropping’ the message (modelling that it did not arrive – it was lost). In the former case the process should behave as if it had arrived, simply becoming (the probabilistic version of) whatever process the possibilistic version becomes. In the latter case the process should revert to the state it was in when the reception event occurred.

### 2.4.3 Modelling Random Initial State

When a system’s initial state is unknown (e.g., initial node locations or velocities) it is desirable to assess its behaviour with respect to any possible initial state. Ideally, this would be done in a way that takes account of the perceived likelihoods of alternative initial states. A probabilistic analysis can model random choice of initial state and yield the probability of certain types of property holding (e.g., the probability that an arbitrary packet is delivered) or an expected cost (e.g., the expected number of hops taken by a packet from source to destination). A probability or expected cost will be with respect to whatever probability distribution of initial states has been modelled.

Possibilistic models often model the initial state being ‘learned’ by a process performing special start-up events – such as the `location_oracle` events of Section 3 – that convey the initial state information. In such cases there is an opportunity to learn the initial state probabilistically: instead of modelling that any one of the possible initial states can be learned, using a  $CSP_M$  input or external choice, one models that the possible initial states are learned *using probabilistic choice*. The effect is to turn the start-up transitions into probabilistic ones. A suitable  $pCSP_M$  model can be obtained from a  $CSP_M$  model by replacing possibilistic start-up event transitions by probabilistic transitions, with probabilities chosen according to the likelihood of the corresponding initial states. An example of this occurs in Section 3. There is a cleaner way to achieve the same effect when the possibilistic model uses external choice to learn the initial state: in such cases one can simply compose the original process in parallel with a probabilistic process responsible for choosing which start-up events happen.

### 2.4.4 Modelling Mobility

In a mobile system we refer to the entities that move as ‘dynamic nodes’. (Some nodes may be static.) In the case of mobile routing protocols, the routers are dynamic nodes.

For simplicity, we will assume that the static model represents nodes by individual processes that record node locations explicitly (in the form of state parameters). In our experience it is straightforward to transform static system models to have these characteristics. Static models will often not record node locations at all, but these can be added without difficulty.

Two important questions arise when characterising node movement. The first is ‘when do particular nodes move?’ and the second is ‘where do they move to?’ These ‘When’ and ‘Where’

aspects of node movement must be understood if mobility is to be modelled correctly.<sup>5</sup> Each can be characterised either nondeterministically or probabilistically. These aspects are considered below.

The basic mechanism proposed here for modelling node movement is a standard one: introduce special 'move' events (perhaps with more descriptive names indicating what is moving, where from, and to where – the choice of which aspects of movement to model will depend on the system under study). The move events should occur in a way that accords with how the nodes move in reality. Faithful modelling of the 'when' and 'where' aspects is discussed next.

The approach we have taken is to model the effects of any possible node movement within the scope of the analysis (at any potential time and to any potential destination) by adding move events to the node processes. Upon performing a move event, a node process recurses to a version of itself with a suitably modified parameter list – the new location parameter reflects the new location. Further constraints on node movement (e.g., to restrict movement to a particular phase of the algorithm, or to restrict the nodes that may move) are modelled by a separate Movement Regulator process. The movement regulator is synchronised with the rest of the model to implement these constraints about which movements are possible. Importantly, it can serve a second purpose: to resolve the choices between alternative move events *probabilistically*. This amounts to choosing probabilistically which nodes move, and to where, when movement occurs.<sup>6</sup>

In terms of the pCSP<sub>M</sub> model, this approach is implemented using external choice of move events when adding possible node movement to the static model, and choosing either possibilistically or probabilistically between node movements in the Movement Regulator process.

In short, we model node movement in two parts: 1) add move events into the static model to allow for all possible movement within the scope of the analysis; and 2) encode any further constraints in a separate Movement Regulator process that resolves the choices about when and where nodes move, probabilistically if desired.

#### 2.4.5 Analysing Efficiency

A simple form of efficiency analysis can be performed using the possibilistic modelling approach presented in Deliverable D3. Using those models we showed that correct location databases are always established by the end of the final GLS bootup round. The analysis used CSP Failures assertions, such as ReportCorrectlyAfterBootup [F= GLS\_BOOTUPS \ BootupComms.

That possibilistic analysis was restricted to the static case without message loss. Both these restrictions can be removed using the techniques described above, to yield probabilistic models. Model checking can then calculate the probability that all location databases are correct at the end of the final bootup round. As with the possibilistic analysis, this enables efficiency issues to be explored to some extent.

In addition, more general notions of efficiency can be measured by analysing probabilistic models with respect to expected cost. For example, costs can be used to represent the passage of time (the number of rounds, etc.) or the communication of messages (the number of messages or their size). Expected cost analysis would then yield the expected time or expected amount of control traffic needed to achieve a defined level of correctness.

Indeed, within the FORWARD project Birmingham University have recently explored issues of expected cost. There is an opportunity to bring expected cost analysis within the performance methodology we have described, by mapping pCSP<sub>M</sub> assertions (or assertions in an extension of this language that allows individual costs to be defined) to PRISM expected cost properties. However, this remains as future work.

---

<sup>5</sup> More generally, these might be considered as 'Whether' and 'Which' aspects: choice of whether a particular type of event occurs, and choice of which particular one of these events occurs.

<sup>6</sup> A limitation is that the choice of *when* movement occurs is left unresolved – it remains nondeterministic using this approach. However, this choice too can be made probabilistically, by a generalised approach: additionally introduce 'non\_move' events into the static model as prefixes to the events that do not model movement; then have the movement regulator choose probabilistically between movement or not, by choosing between move and non\_move events.

### 2.4.6 Complementary Modelling Idioms

Much of the earlier collaboration between QinetiQ and Formal Systems on analysing the correctness of less complex routing schemes was centred on reasonably efficient modelling of “variable audibility”, that is, that links between nodes may fluctuate between full connection and none at all [10]<sup>7</sup>. Rather than intrusively modify the functional code of the node, this effect was modelled by composing a fallible link process with each transmitter for each potential receiver.

When the link is “up” the transmission is allowed to synchronise with the corresponding receiving process as expected. When it is down, a renamed copy of the transmission is either silently discarded, or if appropriate mutated into a timeout event at the receiver; in either case, the sender is unaware that the transmission was unsuccessful. The change in link state is mediated by *link\_up* and *link\_down* events under the control of the environment. This is a technique which might be adapted to fit the current scenario.

In particular, if we wanted to model the lower-level GF component of GRID, then the change in reception as nodes move closer together or further apart could be modelled by suitably constraining the *link\_up* and *link\_down* events in the process(es) monitoring nodes’ positions.

It is unfortunate that the reallocation of resources away from “Technical Focus 3: Quality of Service” of this project did not allow this approach to be investigated in more depth.

---

<sup>7</sup> In the spirit of the work here, this could be extended to include degraded levels of performance.

## 3 Analysing the Performance of Grid

### 3.1 Introduction

All computing networks rely on routing protocols to route messages between physically separate computing devices (such as PDAs, phones, etc. – referred to generically as ‘nodes’). Routing protocols operate by establishing and maintaining routing information at particular nodes on the network (the ‘routers’) that enables them to forward messages to their destinations (either directly, or via other routers). Thus, messages are ‘routed’ along paths in the network by being sent from node to node until they reach their destinations.

In the pervasive computing domain routing protocols have an especially difficult task because nodes can move around (changing the pattern of physical connections) and join/leave networks frequently, both of which make it difficult to maintain sufficiently accurate routing information.

There are three significant overheads associated with routing protocols: (1) control traffic overhead (the amount of traffic that flows around the network in order to establish and maintain the routing information); (2) storage overhead (the amount of routing information itself); and (3) processing overhead (the amount of processing needed to establish, maintain, and act on the routing information). The larger these overheads, the smaller the amount of ‘useful’ traffic that can be routed around the network, or the smaller the network must be for routing to be possible.

As networks become large, it becomes increasingly difficult to design routing protocols that prevent these overheads swamping the available resources (bandwidth/memory/processing power). Grid [6, 1] is a routing algorithm intended to provide scalable routing for large ad-hoc mobile wireless networks, without the various types of overhead becoming too large. It requires no fixed infrastructure and is designed to be deployed rapidly. It has been designed as part of the MIT Oxygen Project <http://oxygen.ai.mit.edu>, focused on enabling pervasive, human-centered computing through a combination of specific user and system technologies.

Grid employs the Grid Location Service (GLS) as a mechanism by which nodes learn the positions of others, together with Geographic Forwarding (GF), a simple technique for routing messages to nodes at geographic locations specified by the senders.

GLS consists of a bootup phase followed by a query/reply phase; it is billed by its developers as “a scalable location service for geographic ad hoc routing” [6]. In support of this claim, [6] presents two pieces of evidence: an informal proof of both correctness and efficiency of the query/reply phase of GLS, and simulation results that indicate performance characteristics in the presence of mobility.

The models we presented in Deliverable D3 [2] allowed us to analyse the correctness of GLS and GF in static networks. Those same models could have been used for analysing efficiency (where efficiency essentially means how many hops packets take to get from source to destination). This section presents analyses of GLS performance during its bootup phase, taking some account of the effects of message loss and node movement.

### 3.2 An Overview of Grid

Grid uses a grid to support the location service. The grid is a hierarchical partitioning of a top-level square (containing all nodes) into 4, then 16, then 64 etc. squares, at each stage dividing squares into 4, until the smallest squares are reached, of which there are  $4^D$  in a square of depth  $D$ . Figure 2 shows a grid with  $D = 2$ . The smallest squares are the order-0 squares, four of which make up an order-1 square, etc., up to the single order- $D$  square (which covers the whole grid). The numbering shown in the figure assigns a unique label to each square in the hierarchy; each label is a sequence of numbers indicating successive choices of quadrant, at increasing depth.

For each  $0 \leq k < D$ , each order- $k$  square has a parent square, which is the containing order- $k + 1$  square, and three sibling squares, which are the other order- $k$  squares in its parent square. For example, square  $\langle 2, 3 \rangle$  has parent  $\langle 2 \rangle$  and siblings  $\langle 2, 1 \rangle$ ,  $\langle 2, 2 \rangle$  and  $\langle 2, 4 \rangle$ . Notice that all squares of the same size completely cover the whole area without overlapping, so each square has

exactly one parent, one grandparent, etc. up to the top-level square  $\langle \rangle$ . We sometimes speak of the  $k$ -parent of a node, where the 0-parent is the containing order-0 square, the 1-parent is the parent of that square, etc. For example, if a node is in square  $\langle 2,3 \rangle$  then its 0-parent is  $\langle 2,3 \rangle$ , its 1-parent is  $\langle 2 \rangle$  and its 2-parent is  $\langle \rangle$ .

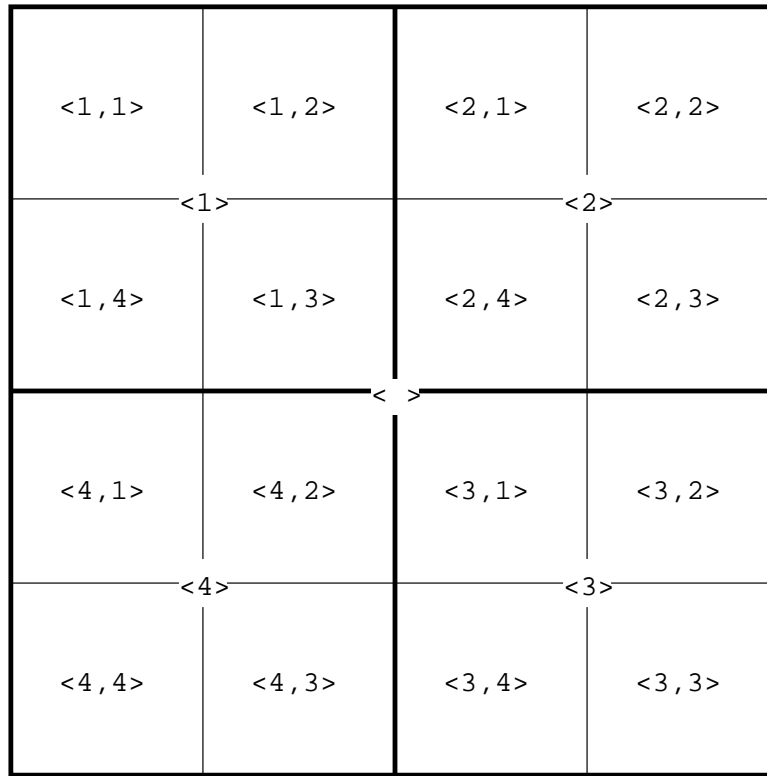


Figure 2: A labelled grid of depth  $D = 2$

Each node is assumed to know the size, position, depth and orientation of the grid.

### 3.2.1 GF

We only give a brief overview of the basic GF algorithm here, since our probabilistic analyses have focussed instead on GLS. Further details of GF can be found in Deliverable D3 [2] and in the literature [6, 1].

Geographic forwarding routes packets to destinations using a claimed destination location, provided by the sender. (As explained above, Grid uses GLS as the general mechanism by which nodes discover the location of a destination.) Each node on the route to the destination chooses the next hop along the growing route in an effort to minimise the geographic (straight line) distance that will remain to the destination, without actually sending to a neighbour that is further from the destination than the sending node is. (So each node tries to send the packet to the neighbour that is closest to the claimed destination.) It may happen that no such hop exists, i.e., that all neighbours are believed to be further from the destination than the sender; in such cases a deadend has been reached and the packet is dropped.

### 3.2.2 GLS

GLS makes use of a notion that some nodes are ‘better’ than others (as location servers) for a given node. This is defined as a total ordering of nodes: supposing nodes have integer identifiers,  $A$  is better than  $B$  as a location server for  $C$  exactly when  $A - C \pmod{N} < B - C \pmod{N}$ , that is, when  $A$  is closer to  $C$  from above, modulo  $N$ , than  $B$  is. This extends to a notion of the best

node, among any given set of nodes, for a given node – the best node is the one that is better than all others in the set. For example, node 2 is the best node for node 17 from the set {node 2, node 7, node 16}.

For each node  $n$ , GLS aims to recruit a set of nodes as  $n$ 's location servers. The objective is to recruit different sets of location servers for different nodes, with each set skewed with a higher concentration of location servers close to the corresponding node than far away.

GLS begins by performing the bootup phase (consisting of rounds 0 up to  $D$ ), which is designed to establish the location servers for all nodes, and then moves to the query/reply phase. Bootup is described in [6] as being performed simultaneously by all nodes in the network; all nodes agree when bootup is occurring, and indeed they agree about which bootup round is in progress. Without loss, duplication or re-ordering of messages, and perhaps also assuming synchronous communications, it appears true that nodes will necessarily remain synchronised in this sense if they all follow the GLS algorithm correctly (by sending and accepting the prescribed messages in each bootup round); however, for the purposes of this analysis this remains an assumption. In successive rounds, more and more location servers are established, for all nodes in the network, at increasing distances from these nodes. Specifically, the nodes behave as described in figure 3.

**Round 0: establish location servers in all order-0 squares.** Each node becomes a location server for all other nodes in its own order-0 square; nodes learn the locations of these other nodes by using what [1] describes as a 'constant overhead local distance vector routing protocol'.

**Round  $k > 0$ : establish location servers in all order- $k-1$  squares.** Each node  $B$  sends an update packet to the three sibling order- $k-1$  squares. Consider a node  $L$  that receives this packet. If  $L$  is the best node for  $B$  in  $L$ 's location database then  $L$  becomes a location server for  $B$ ; otherwise  $L$  forwards the update packet to the best node for  $B$  that  $L$  knows the location of; each subsequent node does the same.

Figure 3: GLS bootup phase

Note that the published descriptions of the bootup phase of GLS [6, 1] contain an ambiguity: whether, during bootup round  $k$ , say, GLS bootup forwards updates only within the containing order- $k-1$  square (WithinSquare mode) or to the best node chosen from the current location database (WholeTable mode). The analysis reported in D3 determined that the WithinSquare mode is the correct one.

After bootup GLS proceeds to the query/reply phase. When a node wants to send a message to a destination but doesn't know its location, it first sends a query about its location to the best node for the destination that it knows the location of; this is then forwarded via the best nodes known to the subsequent recipients until a location server for the destination is reached, which forwards the request to the destination. The destination replies with its current location to the requesting node (whose identifier and location are carried within the request packets). Each of these communications is a location query step, and is implemented using GF. The sender is then able to communicate with the destination node directly.

This report focusses on performance analysis of the GLS bootup phase. We make use of the following precise characterisation – arrived at in Deliverable D3 – of a correct location database, at a given node and at the completion of a given bootup round:

On completion of bootup round  $0 \leq k \leq D$  the location database at node  $n$  is *correct* if it contains exactly: the locations of all nodes in the same order-0 square; and the locations of all nodes in all siblings of  $n$ 's  $j$ -parent, any  $0 \leq j < k$ , for which  $n$  is the best node in  $n$ 's  $j$ -parent.

The analysis reported in Deliverable D3 expressed this correctness test using machine-readable Communicating Sequential Processes (CSP). That work investigated the correctness of the bootup phase

by model-checking.<sup>8</sup> The same characterisation of correctness – indeed, the same  $CSP_M$  process – has been used in our assessment of the performance of Grid, as explained below.

### 3.3 Models of GLS

We have extended the (non-probabilistic)  $CSP_M$  model of GLS reported in Deliverable D3 [2] to some probabilistic versions modelled in  $pCSP_M$  that are suitable for analysing performance effects of message loss and node movement. Random initial locations have been modelled, in order to obtain results averaged over all possible sets of initial node locations. After outlining the non-probabilistic model of GLS, we describe in turn how the GLS model has been extended to model message loss, random initial node locations and node movement.

#### 3.3.1 $CSP_M$ Model of GLS

Details of the purely possibilistic model of GLS were reported in Deliverable D3 [2]; an outline follows.

As explained above, nodes communicate their own locations and their beliefs about the locations of other nodes. Further, they use this information to decide what to do with received messages. Locations are thus fundamental to the operation of GLS. Locations in the model denote small squares in the grid – the level of abstraction ignores physical location within a small square.

The  $CSP_M$  model defines individual Node processes, one for each node identifier. The following process models the node with identifier `id`. Parameters `do_qs` and `fmode` can be ignored.

```
-- A node discovers its location from the location oracle and bootstraps
```

```
Node(do_qs)(fmode)(id) =
  -- Node's location is input using the loc_oracle channel
  loc_oracle.id?loc ->
  Bootstrap(do_qs)(fmode)(id, loc)
```

Each node is modelled as ‘learning’ its initial location on the `loc_oracle` channel. This location is then passed as a parameter `loc` to the `Bootstrap` process. The initial node locations appear in `loc_oracle` events in the traces of the model, so subsequent bootup behaviour can be judged correct, or not, relative to them. This is explained when specification processes are discussed.

Each `Bootstrap` process models a node starting the bootup phase. As bootup proceeds, a node is modelled in turn by a number of processes: `BootstrapBase`, `BootstrapBase'`, and `BootstrapStep`, in each case with parameter values that include the current location `loc` and the current location database `id_locs`. Where necessary, further details are provided below.

#### 3.3.2 Extensions to Model Message Loss

The purely possibilistic  $CSP_M$  model of GLS was extended to model message loss. Constants were introduced into the model which represent the relative likelihood of certain types of GLS message being delivered or lost. For example, the constants `HEAR_BROADCAST` and `DROP_BROADCAST` were introduced to represent the relative likelihood of broadcast messages (used by GLS during bootup round 0) being heard, or not, by nodes in the same location (the same small square). These constants are defined as follows:

```
HEAR_BROADCAST = 9          -- per-receiver success proportion for broadcasts
DROP_BROADCAST = 1         -- per-receiver failure proportion for broadcasts
```

<sup>8</sup> A limited form of efficiency property was also established, using the Stable Failures semantic model.

so the probability of broadcast message loss is 0.1, and that of delivery is 0.9.

Probabilistic choices – available in  $pCSP_M$  – using these constants were introduced into the  $CSP_M$  processes responsible for modelling reception of such messages. For example, the following code shows the  $pCSP_M$  process that models bootup round 0 of the GLS bootup phase, up to sending of the node's location.

```
-- Bootstrap behaviour up to sending of node's own broadcast. A node
-- can receive broadcasts from other nodes or send its own broadcast
-- and continue bootstrap having sent.

BootstrapBase(do_qs)(fmode)(0, id, loc, id_locs) =
  -- send broadcast and change to 'have sent' mode
  snd_broadcast!id.loc ->
    BootstrapBase'(do_qs)(fmode)(0, id, loc, id_locs)
  []
  -- receive all broadcasts from other nodes, but quietly drop those
  -- that aren't sent from the same small square.
  -- Quietly drop the remainder with probability PROB_DROP_BROADCAST=
  -- DROP_BROADCAST/(HEAR_BROADCAST+DROP_BROADCAST).
  rcv_broadcast.id?id2:others(id)?loc2 ->
    if not same_small_square(loc2,loc) then
      -- quietly drop it, since outside this node's small square
      BootstrapBase(do_qs)(fmode)(0, id, loc, id_locs)
    else
      (
        -- process the received broadcast with probability PROB_HEAR_BROADCAST
        BootstrapBase(do_qs)(fmode)(0, id, loc, record_id_loc(id2,loc2,id_locs))
        [HEAR_BROADCAST~DROP_BROADCAST]
        -- drop the 'received' broadcast with probability PROB_DROP_BROADCAST
        BootstrapBase(do_qs)(fmode)(0, id, loc, id_locs)
      )

```

The lines marked \* have been added to model message loss. In accordance with the advice given in Section 2.4.2, the modification adds a probabilistic choice `[HEAR_BROADCAST~DROP_BROADCAST]` between on the one hand processing the reception (treating the message as having arrived successfully) and on the other hand dropping it and not changing state (treating the message as if it had not arrived). That is, when reception is deemed to have occurred in the non-probabilistic model, a final 'hurdle' must be overcome in the probabilistic model: with the defined relative likelihoods, the message is treated as having indeed been received or as having been lost.

Delivery/loss of other messages sent during GLS bootup was modelled similarly (updates sent to target nodes, and updates sent to sibling squares). All message losses have been modelled as message drops by the 'receiver', after which it reverts to its state immediately before 'reception'. An important consequence is that broadcast messages, as modelled, may be delivered to some nodes and fail to reach others. Likewise, update messages may be delivered to a node in one sibling square, but fail to reach any node of another sibling square. An alternative, simpler scheme for modelling message loss would be to model message sends, instead of receptions, probabilistically. We chose to model GLS message loss in this way, since the simpler scheme is unrealistic in the case of radio broadcasts.

A probabilistic assertion is required. Most of the results reported later were obtained by checking a PRISM model derived using the following  $pCSP_M$  assertion:

```
assert ReportCorrectly [T= GLS_BOOTUPS\BootupComms :[probabilistic translation]
```

This assertion states that during the GLS bootup phase no node ever incorrectly reports the location of a node either for which it should not be a location server or for which it has a wrong location.

### 3.3.3 Extensions to Model Random Initial Locations

A constant `random_locs` was introduced to indicate whether or not initial node locations should be random. (Of course, in the static models the initial node locations are permanent.) When `random_locs` is true, each Node process makes a uniform random choice of initial location, from among all grid locations:

```
-- A node discovers its location from the location oracle and bootstraps

Node(do_qs)(fmode)(id) =
  if random_locs then
    -- Uniform distribution of locs, reported on loc_oracle channel
    ~ loc : Location @ [1]
    loc_oracle.id!loc ->
    Bootstrap(do_qs)(fmode)(id, loc)
  else
    -- Node's location is input using the loc_oracle channel
    loc_oracle.id?loc:InitLocs(id) ->
    Bootstrap(do_qs)(fmode)(id, loc)
```

A node's random initial location is modelled using the probabilistic choice operator  $\sim$  of  $\text{pCSP}_M$ , in its replicated form. The same relative proportion (constant value 1) is specified for each choice of `loc` in the set `Location` of all grid locations. This implies that each location is chosen with equal probability. Once `loc` has been chosen this value is advertised, for this node, on the `loc_oracle` channel and bootstrap proceeds with the node in this location. When `random_locs` is false, the else branch is taken and `loc` is input, which models the node 'learning' its location on the `loc_oracle` channel. A new feature, compared to the original possibilistic model, is that nodes can only learn locations from a defined set, `InitLocs(id)`, of possible initial locations. (This is a convenience to allow restricted scenarios to be analysed separately when `random_locs` is false.) In both cases, the initial locations of nodes are explicit in traces of the model, enabling subsequent bootup behaviour to be judged correct, or not, relative to them.

The specification process `ReportCorrectly` was modified as shown below. This change was made simply to accommodate the restriction of possible initial locations (according to the `InitLocs` function) when `random_locs` is false.

```
ReportCorrectly = ReportCorrectly1(0, empty_id_locs)

ReportCorrectly1(i, id_locs) =
  i < NumNodes &
    -- not heard all locations yet
    loc_oracle?id:unrecorded_ids(id_locs)?loc:(if random_locs
                                                then Location
                                                else InitLocs(id)) ->
    ReportCorrectly1(i+1, record_id_loc(id,loc,id_locs))
  []
  i == NumNodes &
    -- heard all locations, so start reporting
    boot -> tock.0 -> ReportCorrectLocations(0, id_locs)
```

The modified `ReportCorrectly` process restricts which `loc_oracle` events can happen. The possibilistic version allowed `loc_oracle` events with `loc` chosen from the `Location` datatype, but the probabilistic version restricts this in general when `random_locs` is false. The effect is to still allow any initial locations when `random_locs` is true, but otherwise to allow exactly those initial locations permitted by `InitLocs`.

As was the case with the possibilistic version of `ReportCorrectly`, subsequent reports are correct for the particular initial locations. The difference in the probabilistic version is that the initial locations are restricted to those that the implementation process is allowed. This has no effect when checking refinement in the traces semantic model, but it is necessary to avoid uninteresting counterexamples in the case of failures refinement.

### 3.3.4 Extensions to Model Node Movement

The GLS model was extended to model node movements (thus moving from a static model to a dynamic model). Nodes are represented by CSP processes that explicitly store their locations as part of their state, so it was a simple matter to model mobility. This was done by allowing them to perform special `move.id.new_loc` events (on a new channel `move`); the occurrence of such an event indicates that the node with the given identifier (`id`) moves to the specified location (`new_loc`).

The following CSP code shows extra lines, marked with `*`, in the `BootstrapBase` process. These lines model the ability of a node to move, as explained below. The rest of the process continues as above.

```
BootstrapBase(do_qs)(fmode)(0, id, loc, id_locs) =
  -- Always able to move to a new location (subject to the environment *
  -- 'allowing' the movement). *
  move!id?new_loc -> *
    (let new_id_locs = record_id_loc(id,new_loc,id_locs) *
     within BootstrapBase(do_qs)(fmode)(0, id, new_loc, new_id_locs)) *
  [] *
  -- send broadcast and change to 'have sent' mode
  snd_broadcast!id.loc ->
    BootstrapBase'(do_qs)(fmode)(0, id, loc, id_locs)
  []
  .
  .
  etc.
```

The extra code allows the `BootstrapBase` process to perform `move.id.new_loc` events where `id` is this node's identifier and `new_loc` can be any location, representing the node's new location. `BootstrapBase` then evolves to the same state, except that the parameters `loc` and `id_locs` are replaced by `new_loc` and `new_id_locs` respectively. The first replacement reflects the assumption that nodes know their own locations at all times. The second replacement models that a node updates its own location database – represented by a set of (`id,location`) pairs – when it moves. The new location database is calculated as the old one updated by recording this node's new location. Hence, the modified process behaves exactly as in the static model, except that it models the ability of nodes to change their locations and update their location databases accordingly.

A mechanism to model node movement restrictions was implemented, in part to help avoid hitting state space problems – a constant concern when model-checking. An additional motivation was to focus analysis on particular patterns of node movement. Focussing the analysis is especially

important when analysis is quantitative, as here, and less so for black-and-white properties.<sup>9</sup>

Several movement regulator processes were defined, one of which is as follows:

```
MovementRegulator =
  ||| id <- NodeId @ RegulateNodeMovements(id)

RegulateNodeMovements(id) =
  move!id?new_loc:LocsNodeCanMoveTo(id) -> STOP
```

`MovementRegulator` regulates the movement of nodes independently of each other. This is achieved by interleaving individual movement regulator processes, `RegulateNodeMovements(id)`, one for each node. At most one move is allowed per node, to any destination from a set of permitted destinations for the node (determined by a configuration function `LocsNodeCanMoveTo`). The restriction arises because each `RegulateNodeMovements(id)` evolves to `STOP` upon performing a move event.

Some alternative definitions of the configuration function `LocsNodeCanMoveTo` are as follows. (All but the final one are commented out here.)

```
-- This function determines the locations to which each node can move,
-- from anywhere. An empty set means the node cannot move.

-- Static system
-- LocsNodeCanMoveTo(id) = {}
--
-- Dynamic system: node 1 can move to location <1>.
-- LocsNodeCanMoveTo(1) = {<1>}
-- LocsNodeCanMoveTo(id) = {}
--
-- More dynamic: node 1 can move to any location.
-- LocsNodeCanMoveTo(1) = Location
-- LocsNodeCanMoveTo(id) = {}
--
-- More dynamic still: each node can move to any location.
LocsNodeCanMoveTo(id) = Location
```

These alternatives represent increasingly liberal restrictions on the types of node movement that are allowed. A movement regulated version of `GLS_BOOTUPS` was defined, as follows:

```
GLS_BOOTUPS_REG = GLS_BOOTUPS [| Moves |] MovementRegulator
```

The synchronisation forces `GLS_BOOTUPS` to perform move events only when permitted by the regulator. Thus, `GLS_BOOTUPS_REG` models the GLS bootup phase when all nodes are allowed a single move to a permitted destination.

When checking properties of the model, we are interested in knowing whether nodes have correct knowledge of the current locations. Therefore, we require judgements about the correctness of reports to be with respect to the current node locations. A configuration parameter `dynamic` was added to `ReportCorrectly1`. A new specification process `ReportCorrectlyDynamic` was defined. The new/modified lines, marked by \*'s below, set this parameter to true and pass is through to a new process `PostBootReportCorrectly` when the boot event happens:

<sup>9</sup> A quantitative value, such as a probability, for a given scenario may be difficult or impossible to infer from the corresponding value for a more general scenario; the extra possibilities will generally skew the value in a way that is excessively difficult to account for. In contrast, a (universal) black-and-white property, such as a CSP refinement property, can always be inferred in a given scenario when true in a more general one.

```

ReportCorrectlyDynamic = ReportCorrectly1(true)(0, empty_id_locs)      *

ReportCorrectly1(dynamic)(i, id_locs) =                               *
  -- Listen to all initial location information.
  i < NumNodes &
    -- not heard all locations yet
    loc_oracle?id:unrecorded_ids(id_locs)?loc:(if random_locs
                                                then Location
                                                else InitLocs(id)) ->
    ReportCorrectly1(dynamic)(i+1, record_id_loc(id,loc,id_locs))      *
  []
  -- Start reporting when have heard them all.
  i == NumNodes &
    -- heard all locations, so start reporting
    boot -> PostBootReportCorrectly(dynamic)(id_locs)                  *

```

PostBootReportCorrectly behaves exactly as the possibilistic version of ReportCorrectly would behave after a boot event, except that it allows move events to occur if dynamic is true. These movements are constrained to occur before the first tock event, so in this case movement is restricted to occurring before commencement of GLS bootup round 1.

```

PostBootReportCorrectly(dynamic)(id_locs) =                           *
  -- tock.0 can happen at any time, at which point settle on these id_locs
  tock.0 -> ReportCorrectLocations(0, id_locs)
  []                                                                     *
  -- in the dynamic case node movements can occur before tock.0; adjust *
  -- behaviour accordingly                                              *
  dynamic == true &                                                    *
    move?id.new_loc ->                                                 *
      let new_id_locs = record_id_loc(id,new_loc,id_locs)              *
      within PostBootReportCorrectly(dynamic)(new_id_locs)            *

```

### 3.4 Performance Analyses and Results

Various system configurations were analysed. This involved varying the values of configuration constants in the  $pCSP_M$  model, and in each case running a PRISM 'experiment' on (a slightly modified version of) the PRISM model produced by the translator. The PRISM experiments varied the probabilities of message loss for one or more types of bootup message. These aspects of configuration are discussed next, after which the results are presented and discussed.

#### 3.4.1 Configuration Constants

There are four configuration constants in the  $pCSP_M$  model. In the following example they are defined with values for a particular configuration: 3 nodes in different initial locations of an order-1 grid (4 squares).

```

-- Number of nodes
NumNodes = 3

-- Maximum order of Grid squares
MaxOrder = 1

-- Random initial locations?

```

```

-- When true, each node starts in a random location (uniform prob
-- distribution) chosen from entire grid, i.e. all locations.
random_locs = false

-- Possible initial locations of nodes (ignored when random_locs == true)
-- Different locations version, giving diff_locs results
-- (Runtime error if NumNodes > 4.)
InitLocs(n) = {<n>}

```

The constants `NumNodes` and `MaxOrder` are self-explanatory. Recall that the constant `random_locs` controls whether the initial node locations are chosen at random (from among all grid locations). When `random_locs` is false the function `InitLocs` determines the possible initial node locations. In this example the nodes are restricted to different initial locations: node 1 is restricted to location `<1>`, node 2 to location `<2>`, and node 3 to location `<3>`. Table 1 shows the five ways in which the model was configured with respect to initial node locations.

Type of configuration	Initial node locations	Definition of constants
single_loc	All nodes in the same location	random_locs = false InitLocs(n) = <1>
diff_locs	All nodes in different locations	random_locs = false InitLocs(n) = <n>
two_same_locs	Two nodes in the same location, others in different locations	random_locs = false InitLocs(1) = <1> InitLocs(2) = <1> InitLocs(n) = <n>
all_locs	All nodes chosen from all locations	random_locs = false InitLocs(n) = Location
random_locs	All nodes in random locations anywhere in the grid	random_locs = true

Table 1: Five ways of configuring the choice of initial locations

### 3.4.2 PRISM Experiments

In the simplest application of the methodology, PRISM is run directly on the output of the translator. However, we wished to plot graphs illustrating the dependence of GLS bootup performance on the probability of message loss, for different types of bootup message. For this purpose we used PRISM's support for 'experiments', whereby PRISM 'constants' are left undefined in the PRISM model and are varied by PRISM according to the experimental set-up. The PRISM experiments varied the probability of message loss for one or more types of GLS bootup message, causing PRISM to calculate the minimum/maximum probabilities for each combination of these probabilities. When more than one probability was varied, they were sometimes varied independently and sometimes in tandem.

To introduce the necessary PRISM constants, the PRISM models produced by the `pCSPM` to PRISM translator were modified by means of automatic scripts. The modifications replaced the numerical probabilities in the PRISM model by new PRISM constants, which were then varied in the

experiments.<sup>10</sup>

Table 2 shows the three experimental set-ups we used. `prob_hear_broadcast` denotes the probability that an arbitrary GLS broadcast message is heard by an arbitrary node in the same order-0 square during bootup round 0. `prob_hear_update` denotes the probability that an arbitrary GLS update message, whether targetted at a particular node or to sibling squares, is heard by the targetted node or by some node in an arbitrary occupied sibling square. (In fact, the two types of update message are treated separately in the  $pCSP_M$  model, as explained in Section 3.3.2. Our experimental set-ups defined the corresponding target/sibling-square probabilities to be equal.)

Type of Experiment	Defined Probabilities
Type 1	vary <code>prob_hear_broadcast</code> from 0.0 to 1.0 in steps of 0.1 set <code>prob_hear_update</code> to 1.0
Type 2	vary <code>prob_hear_broadcast</code> from 0.0 to 1.0 in steps of 0.1 vary <code>prob_hear_update</code> in tandem
Type 3	vary <code>prob_hear_broadcast</code> from 0.0 to 1.0 in steps of 0.2 vary <code>prob_hear_update</code> from 0.0 to 1.0 in steps of 0.2

Table 2: PRISM experiments vary the probability of delivery of bootup messages

### 3.4.3 Analysis Results

Later we give the results of our analyses as 2-D and 3-D graphs. First we describe how to interpret them. The 2-D graphs show the results for PRISM experiments of types 1 and 2 in Table 2. Each such graph shows the minimum and maximum probabilities of correct bootup as functions of the probability of successful delivery of GLS broadcast messages (type 1 experiments) or of GLS broadcast or update messages (type 2 experiments).

In the current context, ‘correct bootup’ means that at the end of the GLS bootup phase no location server for a node records a wrong location for the node or should not be a location server for the node. A stronger notion of correct bootup additionally requires that all the intended location servers have been established. We have focussed on the weaker form, because it can be assessed using refinement in the Traces model of CSP. In Section 4.2 we consider extension to the stronger condition; meanwhile, it should be remembered that the weaker notion of correct bootup is necessary for the stronger to hold.

For example, the 2-D graph in Figure 4 shows Experiment 2 analysis results for 3 static nodes in different locations.

<sup>10</sup> The scripts exploited the fact that we had defined the  $pCSP_M$  constants denoting proportions of messages dropped/heard to give distinct numerical probabilities for dropping and hearing of each type of bootup message. A more general way to support experiments would be to preserve symbolic probabilities in the translation – make the  $pCSP_M$  to PRISM translator map  $pCSP_M$  symbols representing probabilities (or proportions) to corresponding PRISM symbols. This possibility is discussed in Deliverable D14.

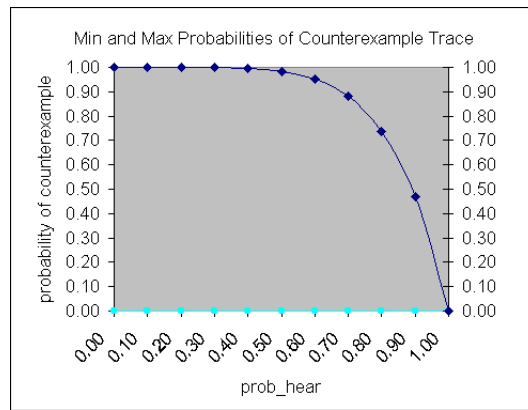


Figure 4: Minimum and maximum probability of incorrect bootup for 3 static nodes in different locations, by experiment type 2. The vertical axis is the probability of incorrect bootup; the horizontal axis is the probability of a broadcast message being received by an arbitrary node in the same order-0 square, which is the same as the probability of an update message (whether to a target node or to sibling squares) being received by some node in an arbitrary sibling square.

The 3-D graphs show the results for PRISM experiments of type 3 (as defined in Table 2). Each such graph shows the minimum or maximum probabilities of correct bootup as a function of two probabilities: the probability of successful delivery of GLS broadcast messages, and the probability of successful delivery of GLS update messages. For example, the 3-D graphs in Figure 5 show Experiment 3 analysis results for 3 static nodes in a single location.

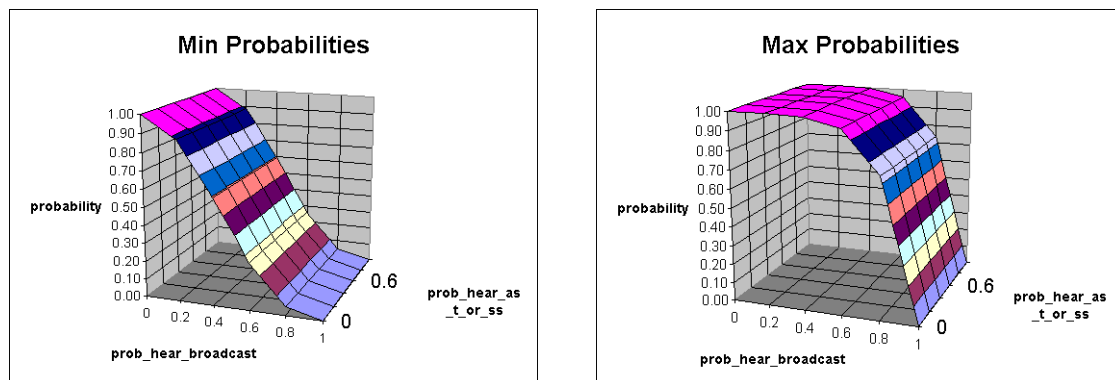


Figure 5: Minimum and maximum probability of incorrect bootup for 3 static nodes in the same location, by experiment type 3. The vertical axis is the probability of incorrect bootup; the horizontal axes are the probability of a broadcast message being received by an arbitrary node in the same order-0 square, and the probability of an update message (whether to a target node or to sibling squares) being received by some node in an arbitrary sibling square.

In all graphs, the size of the gap between the minimum and maximum probability curves (or surfaces) shows the degree of uncertainty about the exact probability. This uncertainty is genuine: it arises from the decision to quantify the probability over all possible environments. The environment effectively controls all nondeterministic choices in the PRISM model, which derive from internal and unresolved external choices of the  $pCSP_M$  model. The prescribed probabilities in the model are, however, respected.

Notice that gaps between the minimum and maximum probabilities – the degrees of uncertainty – tend to be largest for the `all_locs` configuration. More definite results are obtained when initial locations are restricted to a single location, to different locations or (for 3 nodes) to two in one location and the third in another.

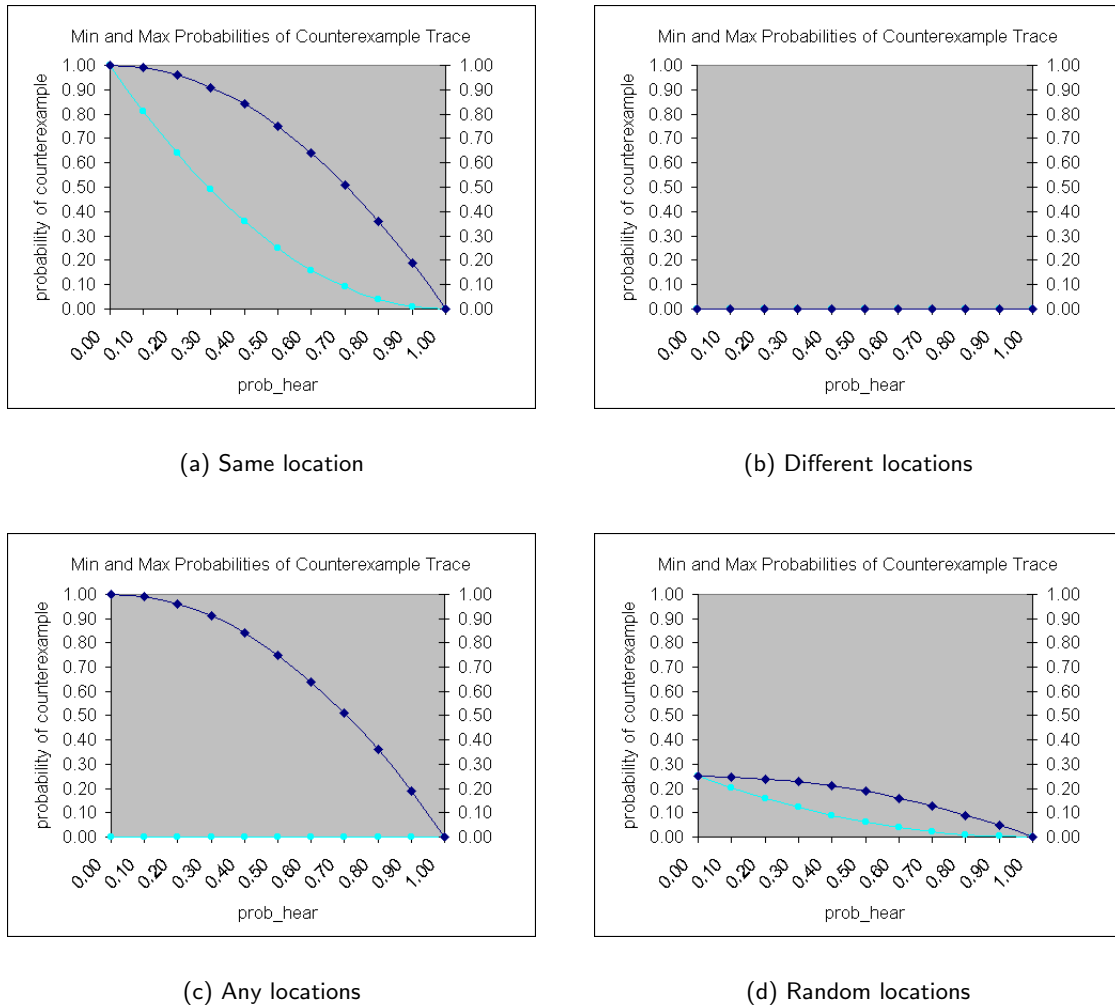
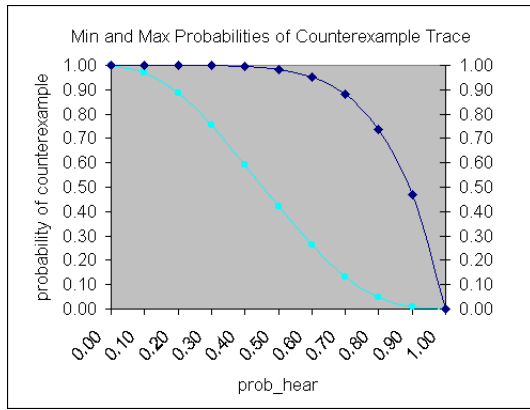


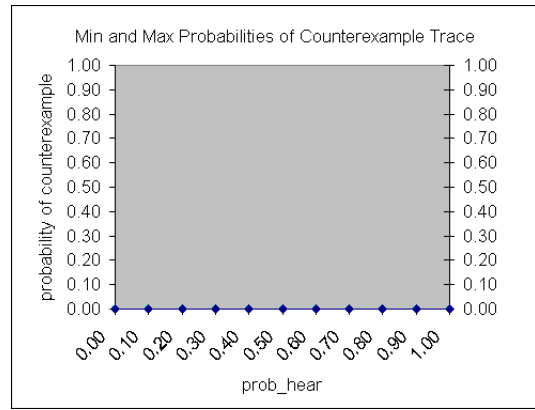
Figure 6: Minimum and maximum probability of incorrect bootup for 2 static nodes as functions of the probability of an arbitrary broadcast message being received, for various ways of choosing initial locations

Sometimes the minimum and maximum curves/surfaces coincide, giving exact probabilities of the outcome of interest (incorrect bootup, here). This happens in the `diff_locs` configuration when only broadcast messages can be lost (Figure 6(b) and Figure 7(b)); in these cases the probability of incorrect bootup is calculated to be exactly 0.0, which is to be expected.

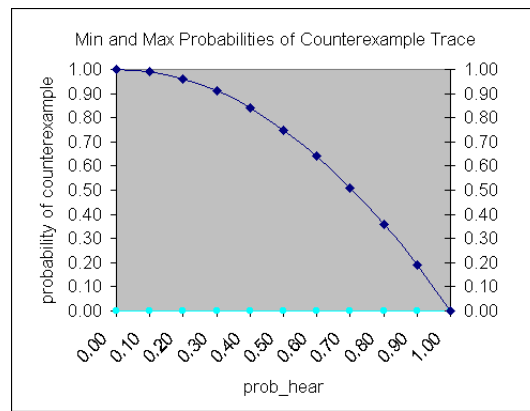
The `random_locs` configurations give quite small degrees of uncertainty, reflecting that they are effectively averaged over all possible sets of initial locations. Indeed, the results obtained agree precisely with predictions we made about how the `random_locs` results relate to the `single_loc`, `diff_locs` and `two_same_locs` results: the `random_locs` results are weighted averages of the others, the weights being the probabilities of these other configurations occurring in practice. The `all_locs` results also agree with our predictions: the minimum probabilities are the least of the corresponding minimum probabilities – giving the ‘min of mins’ curve. Similarly, the maximum probability curve is the ‘max of maxs’.



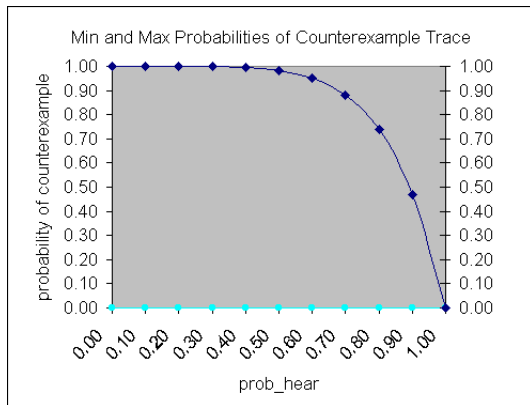
(a) Same location



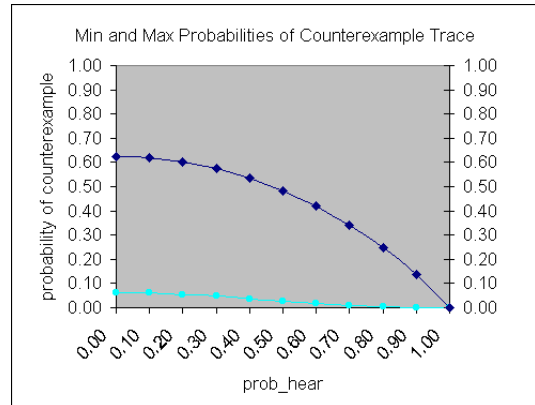
(b) Different locations



(c) Two same locations, one different



(d) Any locations



(e) Random locations

Figure 7: Minimum and maximum probability of incorrect bootup for 3 static nodes as functions of the probability of an arbitrary broadcast message being received (where no update messages are dropped) for various ways of choosing initial locations

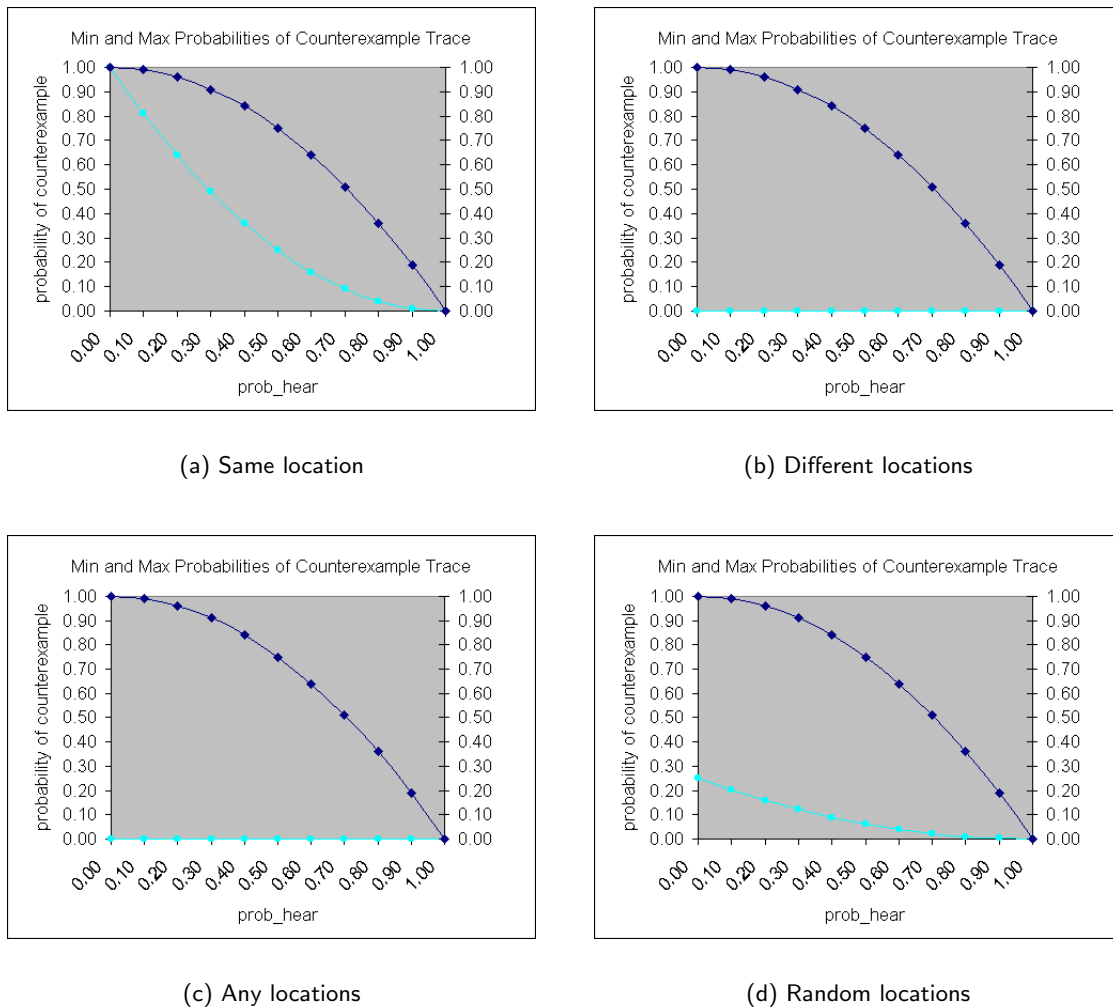
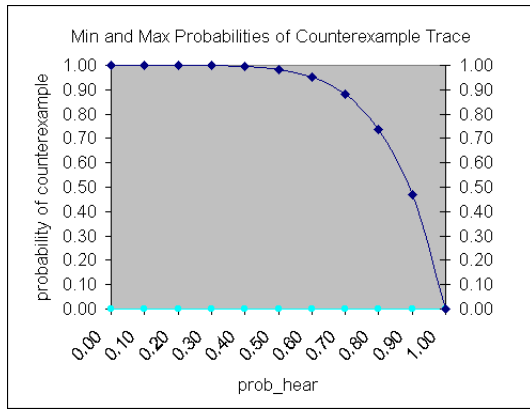


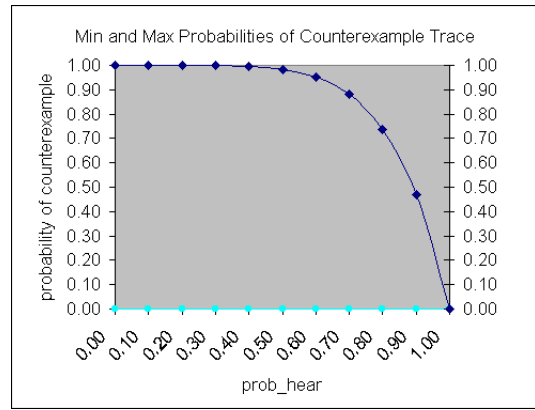
Figure 8: Minimum and maximum probability of incorrect bootup for 2 static nodes as functions of the probability of an arbitrary broadcast or update message being received, for various ways of choosing initial locations

Notice, too, that increasing the number of nodes from 2 to 3 makes the minimum and maximum bounds worse – it increases the (bounds on the) chances of incorrect bootup. This suggests that larger number of nodes will be more likely to boot up incorrectly. On the other hand, the property we have checked is very strict since it considers bootup to have failed if any location server is incorrect. It may be appropriate to consider alternative notions of correctness that allow some degree of incorrect bootup, but this is left to future work.

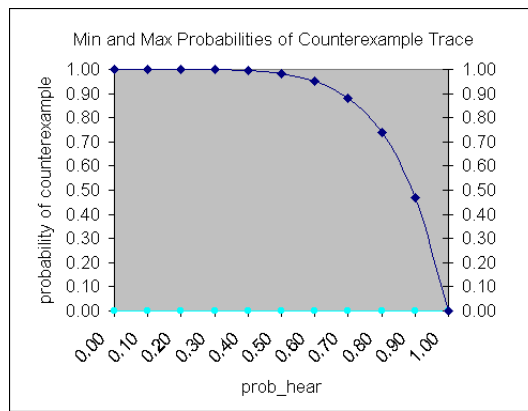
In many cases of experiment type 3 the results depend on only one or other varying probability. When results only depend on the probability of broadcast message loss, not update message loss, the 3 dimensional graph for a type 3 experiment project to the 2 dimensional graph for the corresponding type 1 experiment. For example, Figure 8(a) is a projection of Figure 10(a).



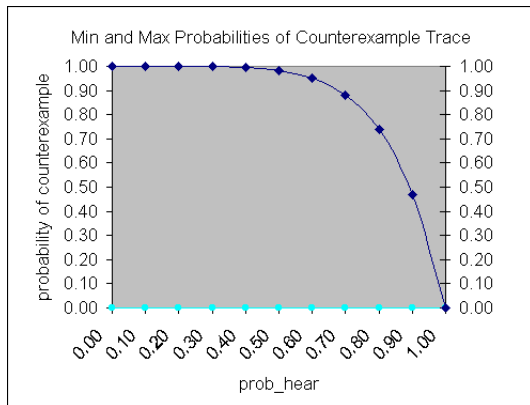
(a) Same location



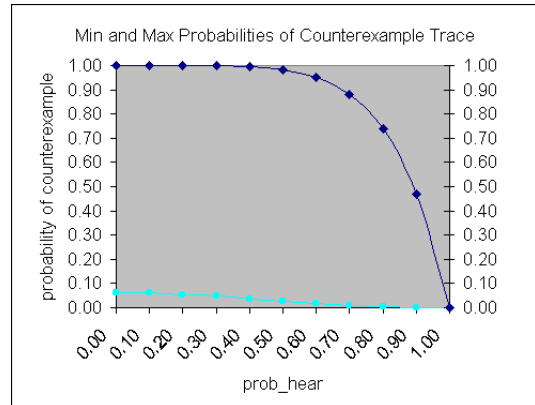
(b) Different locations



(c) Two same locations, one different

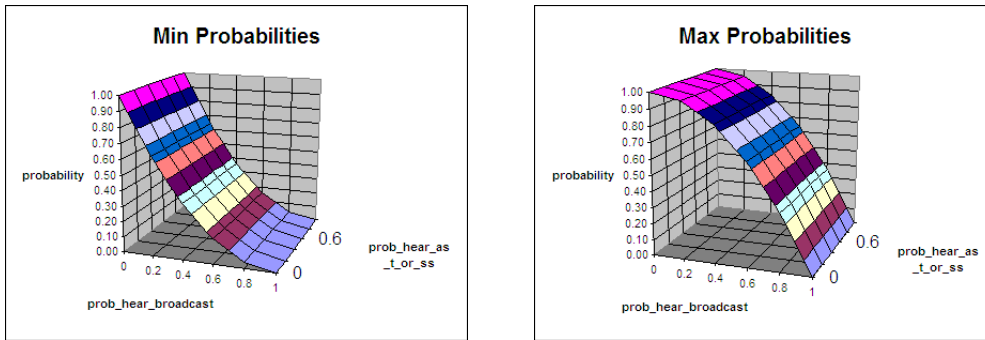


(d) Any locations

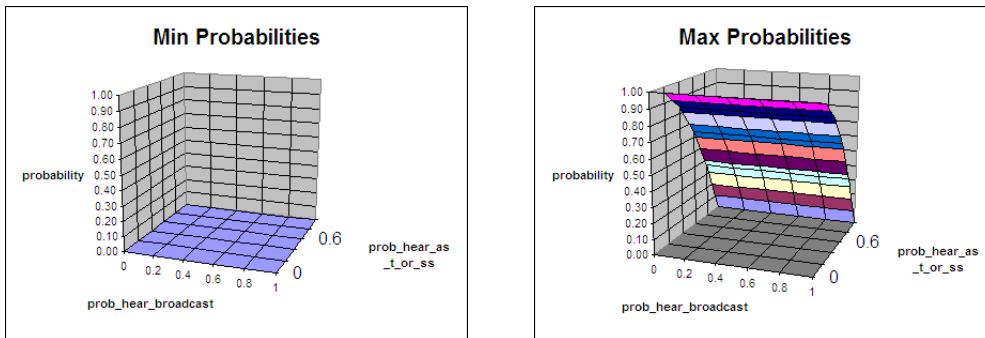


(e) Random locations

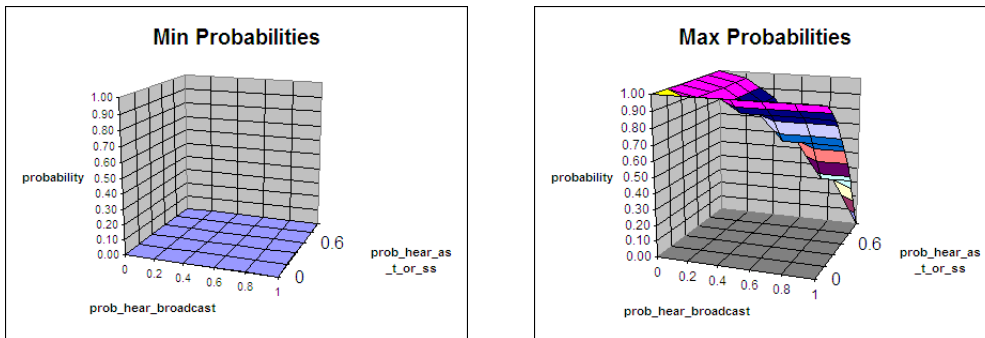
Figure 9: Minimum and maximum probability of incorrect bootup for 3 static nodes as functions of the probability of an arbitrary broadcast or update message being received, for various ways of choosing initial locations



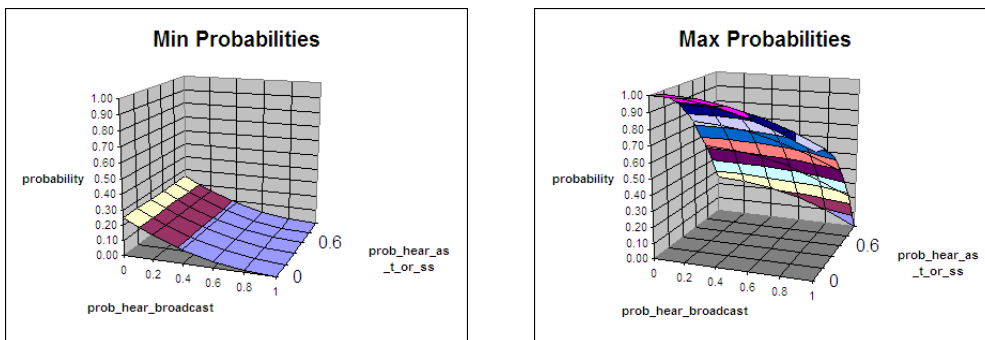
(a) Same location



(b) Different locations

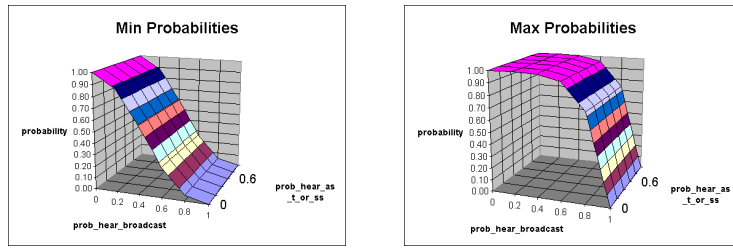


(c) Any locations

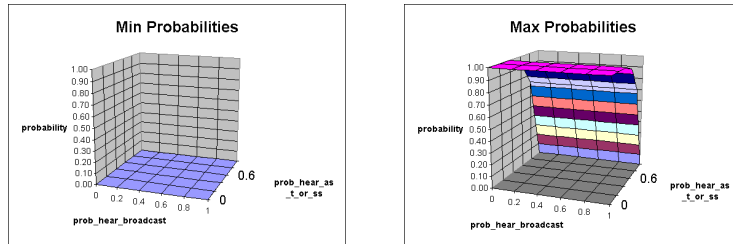


(d) Random locations

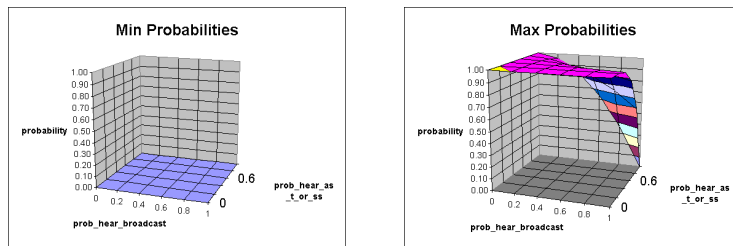
Figure 10: Probability of incorrect bootup for 2 static nodes as functions of the probabilities of broadcast message reception and of update message reception, for various choices of initial locations



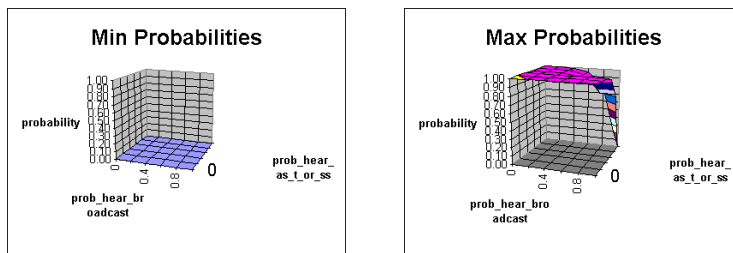
(a) Same location



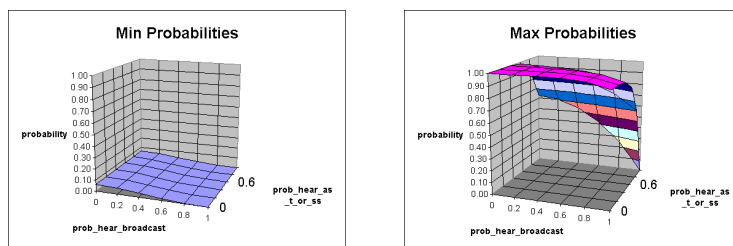
(b) Different locations



(c) Different locations



(d) Any locations



(e) Random locations

Figure 11: Probability of incorrect bootup for 3 static nodes as functions of the probabilities of broadcast message reception and of update message reception, for various choices of initial locations

### **3.4.4 Discussion**

The performance analysis of Grid has characterised the probability of correctness of the location servers established by the GLS bootup phase for small static networks and, to a limited extent, for small dynamic networks. Specifically, the analysis has characterised the probability of correctness as functions of:

- the probability of broadcast message delivery, when all update messages are delivered;
- the probabilities of broadcast and update message delivery, varied in tandem;
- the probabilities of broadcast and update message delivery, varied independently.

In addition, we have run some dynamic checks with limited node movement possible. However, these results have been omitted because they do not contribute much to this characterisation of GLS. However, our investigation of the performance of GLS in a dynamic environment has been valuable, since it has informed the methodology. In particular, it has made clear the need for probabilistic modelling of when nodes move, not simply where they can move to.

It is of course important to ask whether performance has been characterised in appropriate terms. Is it sensible, for example, to model the probability of broadcast message loss distinctly from that of update message loss? It is also important to understand how hard it is to choose the right kind of characterisation, and to have techniques that guide this choice. It is hard to answer such questions for a new methodology, but they will be addressed in future work. Nevertheless, we believe we have chosen to characterise performance in sensible terms. The discussion of methodological issues in Section 2.4 is a good starting point for further investigation.

As was the case with the correctness results of deliverable D3 [2], we have only obtained results here for small problem sizes – up to three nodes in a grid of depth 1 (four order-0 squares). However, these recent results have been obtained without assuming perfect delivery of protocol messages: they indicate how the performance of Grid, particularly of GLS, changes according to the probabilities of delivery of various messages on which Grid depends. Moreover, we have successfully modelled dynamism (node mobility) to some degree – specifically, the dynamic model of GLS allows limited node movement, with probabilistic choice of destinations and nondeterministic choice of when nodes move.

The analysis reported here has demonstrated that model-checking small systems can give insight not only into whether they work in a defined possibilistic environment (often an ideal one, as was the case with the analysis reported in Deliverable D3), but also into how well they work in more realistic environments, modelled probabilistically.

## 4 Limitations and Scope for Further Development

We describe the main limitations of the performance methodology and of its application to GRID, and then discuss some opportunities to address these limitations.

### 4.1 Limitations

The limitations of the reported performance analysis of GRID are as follows:

- It has only addressed GLS, not GF.
- It has only been used to analyse performance for a few nodes in a small grid.
- We have focussed on a particular correctness property, expressed by a traces refinement, which does not include availability.

This form of correctness is necessary for GRID to deliver the claimed service, but it is not sufficient. Furthermore, our analysis has focussed on correctness aspects of performance, not efficiency aspects. It is fair to expect that both these limitations can be addressed, to some extent at least, using failures refinement. (Failures refinement allows expression of availability properties.)

- We have only modelled the When aspect of mobility nondeterministically. (The Where aspect is modelled probabilistically.)

The current nondeterministic modelling of when movements occur means that our analysis of GLS in the presence of mobility is quite weak. In particular, the bounding probabilities calculated by our methodology are quantified over environments that effectively control when movements occur. (Only the choice of which movement occurs is currently probabilistic.) This has the effect of reducing the value of the analysis. However, it has served to highlight the importance of modelling the When aspect of movement probabilistically. Note that the methodology as described allows the When aspect to be modelled probabilistically.

- We have not modelled GLS mechanisms designed to cope with node movement.

GLS's mechanisms for coping with node movement could have been modelled within the current CSP framework, but it was felt important to concentrate our efforts on the central issues of modelling mobility, random initial state and message loss. However, GLS's coping mechanisms have no effect while nodes remain static, so our results for static systems remain valid.

The limitations of the underlying methodology are as follows:

- Scaling analyses to large networks remains a significant hurdle.
- There is currently no support for expected cost formulations of performance.

### 4.2 Extending the Performance Analysis of GRID

Our application of the methodology to GRID has focussed on assessing a weak correctness property, but there are opportunities to extend this analysis to other aspects of correctness and efficiency. This is possible using the existing methodology – one can consider other assertions that are in our possibilistic model of GRID, including traces refinements and failures refinements. Failures refinements allow availability properties to be expressed, so probabilistic analysis based on failures assertions has the potential to extend significantly the class of performance properties that are checked.

It is clearly desirable to model the When aspect of movement probabilistically, and to allow a greater degree of dynamism. Both of these extensions are easy to model using the existing methodology. Allowing greater node movement is likely to increase the size of the model – both the size of the state machine and the size of the state space – so it will require the use of scaling techniques.

Although not reported above, we have investigated opportunities to scale the performance analysis of GRID to more nodes and to larger grids. The first thing to understand when tackling scaling issues is of course how the problem has arisen – which stage of the analysis is the bottleneck. In the case of the GRID modelling the bottleneck has been compilation by FDR, both for the possibilistic analyses reported in Deliverable D3 and for the probabilistic analyses reported above. The main technique we have explored is the use of factoring to reduce the compilation effort required; this is a well-known approach to reducing compilation effort. The results to date show promise.

There are several other options for scaling the analysis, most notably the exploitation of symmetry (GRID is highly symmetric) and the use of induction (there are clear opportunities to apply inductive reasoning and use FDR to prove the base case and inductive steps). It may be possible to apply Data Independent Induction (D.I.I.) techniques [3]; automated support for D.I.I. is under development in Task 7.2 of FORWARD.

Restricting the initial state can be an important way of focussing the analysis to exclude uninteresting scenarios. In addition, it can be useful for scaling purposes because a model with restricted initial state can be much smaller than one with unconstrained initial state and the results of analysis of the unconstrained case can often be obtained indirectly, by analysing the individual restricted cases and combining the results. Section 3 demonstrates that this can be an effective way of scaling the analysis.

The algebraic properties of CSP allow the results of separate possibilistic analyses of the components of a system to be combined to give properties of the entire system. This opportunity was reported in Deliverable D3. A similar approach could be taken for probabilistic analysis of GRID (including GF). It may be that characterising the performance of GF, and using the results in a separate performance analysis of GLS, would give useful results. Such an approach would require care to ensure that the characterisation of GF performance does not lack significant information.

### **4.3 Further Development of the Methodology**

We believe the methodology described in Sections 2.3 and 2.4 represents a good starting point for further development. Continued application to GRID and to other protocols will of course allow us to refine the guidance we have provided.

An interesting and potentially useful way the methodology could be extended is to include support for expected cost analysis of  $pCSP_M$  models, exploiting PRISM's recent expected cost feature. As with the current methodology, we would hope to provide this support in a bolt-on fashion.

## 5 Conclusions

We have developed a methodology for performance analysis and have used it to characterise the performance of GLS, an important part of the GRID protocol suite. The methodology characterises performance by calculating accurate minimum and maximum bounds on the probability of a specified condition being satisfied. We have used it to calculate bounding probabilities that GLS boots up correctly in an environment where messages can be lost and node movement is possible.

Our methodology makes use of the  $pCSP_M$  to PRISM translator developed within FORWARD and reported in Deliverable D14 [5]. It includes guidance on how to extend possibilistic models (written in  $CSP_M$  and generated by techniques such as those described in Deliverable D3 [2]) into probabilistic models (written in  $pCSP_M$ ) suitable for performance analysis. This guidance describes 'bolt-on' techniques that can be applied systematically and easily.

GRID has been a useful case study, providing a context in which to explore general performance analysis issues and to develop systematic modelling techniques. In the process, we have investigated the capabilities and limitations of the methodology. To a large degree, we have focussed on a particular type of property: correctness of the GLS bootup phase. This phase is relied on by GLS to establish location servers for each node at a particular subset of nodes across the network. The intended location servers are determined by the node identifiers and their locations. If bootup fails to establish the location servers correctly then the subsequent Query/Reply phase of GLS will fail to communicate messages between nodes correctly.

In Deliverable D3 we reported our investigations into the correctness of GLS bootup under ideal conditions: no message loss and no node movement. The performance methodology has allowed us to characterise the correctness of GLS bootup when messages are lost with defined probabilities and when nodes can move. We have focussed on investigating the effects of message loss more than the effects of node movement.

The results obtained serve to validate the approach. We were surprised by some of the results presented in this report, but in each case we realised after a little thought that the methodology had found the correct answers. More objectively, hand calculations allowed us to predict some results – most notably the `single_loc` and `diff_locs` results for 2 static nodes, and the results for `random_locs` and `all_locs` as functions of the `single_loc`, `diff_locs` and `two_same_locs` results. These predictions were confirmed by the methodology.

We have identified several limitations of the methodology and its application to GLS, and have discussed how these limitations can be addressed.

## References

- [1] D. Aguayo, D. De Couto, W. Lin, H. Lee, and J. Li. Grid: Building a robust ad hoc network. In *Proceedings of the 2001 Student Oxygen Workshop*. MIT Laboratory for Computer Science, 2001.
- [2] Christie Bolton, Sadie Creese, Michael Goldsmith, Gavin Lowe, Nick Moffat, Helen Roscoe, and Paul Whittaker. Correctness of routing in wireless communications environments. Deliverable D3, *FORWARD* Project, October 2003.
- [3] Sadie Creese. *Data independent induction: CSP Model Checking of Arbitrary Sized Networks*, 2001. D.Phil. Thesis, University of Oxford.
- [4] Formal Systems (Europe) Ltd. *Failures-Divergence and Refinement: FDR2 User Manual*, 6 edition, May 2003. Available from [www.fsel.com](http://www.fsel.com), as part of the FDR2 download, key = Fse, bib = manuals/manuals.bib.
- [5] Michael Goldsmith and Paul Whittaker. A CSP frontend for probabilistic tools. Deliverable D14, *FORWARD* Project, February 2005.
- [6] J. Li, J. Jannotti, D. De Couto, D. Karger, and R. Morris. A scalable location service for geographic ad-hoc routing. In *Proceedings of the 6th ACM International Conference on Mobile Computing and Networking (MobiCom '00)*, pages 120–130, August 2000.
- [7] Dave Parker, Gethin Norman, and Marta Kwiatkowska, February 2004. PRISM 2.0 Users' Guide, available at [www.cs.bham.ac.uk/~dxdp/prism](http://www.cs.bham.ac.uk/~dxdp/prism).
- [8] A.W. Roscoe. *The Theory and Practice of Concurrency*. Prentice Hall, 1998. ISBN 0-13-6774409-5, pp. xv+565.
- [9] William Simmonds and Tim Hawkins. A CSP framework for analysing fault-tolerant systems. Deliverable D9, *FORWARD* Project, June 2004.
- [10] Irfan Zakiuddin, Michael Goldsmith, Paul Whittaker, and Paul H. B. Gardiner. A methodology for model-checking ad-hoc networks. In *Proceedings of the 10th International SPIN Workshop. Portland, OR, USA, May 9-10, 2003*, pages 181–196, May 2003.