Formal Techniques for the Analysis of Wireless Networks

AK McIver
Dept. Computer Science,
Macquarie University,
NSW 2109 Australia

and National ICT Australia
Email: anabel@ics.mq.edu.au

A Fehnker
National ICT Australia
Sydney NSW
Email : ansgar@cse.unsw.edu.au

***Abstract*— Wireless networks consist of small (possibly) portable devices which combine battery-operated computing power and wireless communications. There are a number of technical challenges associated with their operation. These are addressed in part by emerging protocols which attempt to make trade-offs between the various network phenomena in order to optimise overall performance relative to an intended application.**

Central to the protocol design process is the availability of rigorous tools and techniques for quantifying any putative performance advantage gained by a particular protocol, and the degree to which its use degrades overall network functionality. The tools performing this important task today are simulators but the results from them are often not realistic as they have not been validated against empirical data [6].

In this paper we explore the benefits of a formal approach to the analysis of wireless networks; in particular we investigate how a careful mix of model checking and proof may be used both to validate design decisions, and to provide a full profile of quantitative performance-style behaviours. Moreover the counterexample facility of model checking can illustrate clearly the limitations of some standard protocols. We demonstrate the methods on flooding and communications protocols.

Keywords: Formal quantitative analysis; quantitative program logic; wireless protocols; probabilis;tic verification; (probabilistic) model checking.

I. Introduction

Wireless networks consist of an aggregate of small (often portable) devices (or "nodes"), which combine battery-operated computing power and wireless communications, allowing them to form consistent communications networks without human intervention. There are a number of computational challenges related to wireless networks, many of them driven by the technological constraints imposed by the hardware and intended applications. These include schemes to allow nodes to join and leave the network "at will", and schemes to conserve power. Emerging protocols attempt to address these issues [27], amongst others, and each design is successful in treating them to a greater or lesser extent as various trade-offs are made between (for example) proposed energy-saving schemes and the effect on the overall network performance.

Central to the protocol design process is the availability of rigorous tools and techniques for quantifying any putative performance advantage gained by a particular protocol, and the degree to which its use degrades overall network functionality.

The tools performing this important task today are simulators but the results from them are often not realistic as they have not been validated against empirical data [6]. Indeed many of the assumptions they routinely make are at best unspecified, so that the results say as much about the idiosyncracies of a particular underlying model rather than reflecting the characteristics of the protocol design under study. One of the benefits of a formal approach is that those assumptions are clearly and explicitly stated "up front", enabling strong guarantees about performance to be obtained for a range of scenarios, including those which are error-prone.

In recent years technology for analysing correctness [3] and quantitative performance [16], [12] of probabilistic systems has been developed using a combination of algorithmic (model checking) and specialised logics and proof-based methods [21] to help designers judge the quality of their designs. Together they represent a powerful toolkit for quantitative performance evaluation, and the methods are beginning to be applied to wireless networks, an application domain which is subject to significant random disturbances, caused by both the operating conditions and by randomised "back-off" strategies employed by protocol designers.

The theme of this paper is to explore to what extent current formal techniques may be applied to the analysis of performance and correctness of protocols for wireless networks. Specifically we investigate how

1) The counterexample facility of model checkers may be used to demonstrate very concisely protocol limitations (Sec. II);
2) The exhaustive search facility of probabilistic model checking may be used to compute the full range of probabilistic behaviours (Sec. III);
3) The use of models which naturally contain a theory of refinement and abstraction implies that detailed performance studies on small examples apply equally to larger-scale systems (Sec. III and Sec. IV).

Finally we illustrate the methods on two small examples, set out at Sec. IV.

The notational conventions used are as follows. Function application is represented by a dot, as in $f.x$. Given predicate *Pred* we write [*Pred*] for the *characteristic* function mapping states satisfying *Pred* to 1 and to 0 otherwise.

II. Traditional model checking: robust guarantees

Traditional model checking techniques have been used successfully in the analysis of a wide-range of computing applications [8]. The fundamental principal underlying a model-checking approach is that of *exhaustive search*, implying that a "model checked system" is guaranteed to satisfy the specified property. In addition, any discovered counterexample can supply important feedback on problematic network or protocol design.

To illustrate the model checking approach, we study a simple flooding protocol routinely used in many wireless networks: typically a flooding protocol implements a simple procedure for transferring data or instructions throughout the entire network. One of the first formal treatments of flooding [5] used a formal model and a hand proof to establish the "folk theorem" that the standard flooding protocol is unreliable despite the use of redundancy to reduce the effect of message collisions. In this section we implement (essentially) that model as a network of timed automata, and show how a key result of that paper can be achieved more concisely using the counterexample facility provided (in this case) by the Uppaal model checker [3].

A. Model checking the flooding protocol

We take as our starting point the formal model for flooding proposed by Cardell-Oliver [5], which works roughly as follows. First, nodes wait until they receive a message, and as soon as they do, they attempt propagate it to all nodes within range, after which they "go to sleep". If a node notices that sending would cause a garbled message, it "backs off", i.e. the node waits for a random period before attempting to transmit — in the version of flooding studied here, nodes transmit at most once. The purpose of the Uppaal model, which will be described below, is to illustrate the suitability and limitations of standard, non-probabilistic model checking for protocols in the wireless domain, as a replacement of a manual proof. A more thorough treatment of flooding and related protocols, including the estimation of performance is described elsewhere [10].

Although a detailed model of the physical timing relating to the various network activities would result in a model too complicated for a feasible formal treatment, as Cardell-Oliver points out, appeal to the strong causal relationships between those activities leads to dramatic simplifications. For example, nodes may only forward a message *after* receiving one — put together with the timing constraints on communications, we may assume that the network activities are organised in a sequence of "logical" rounds. In each round all "sending" nodes transmit their messages to the "landscape" (explained below) which are then received, if possible, by "receiving" nodes. [1]

The idea of a *landscape* abstracts from the physical terrain, and defines the connectivity of the various nodes in the network as follows. It is modelled as a grid, with the gridpoints delimiting the broadcasting range of a transmitted message — thus a node is within range of another if both reside at adjacent gridpoints. When a message is transmitted, in this simple scenario, we assume that any node within range may pick it up — we do however allow the possibility that messages may arrive at their destination garbled whenever two nodes broadcast to the same gridpoint at the same time.

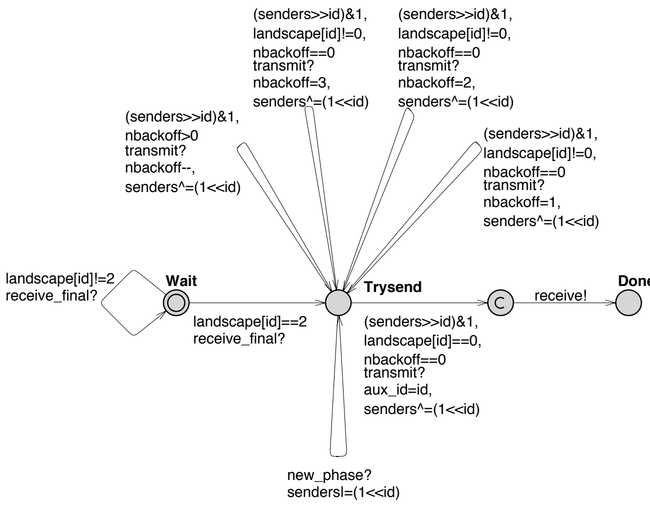

Fig. 1. The flooding protocol at a node

Uppaal uses as input networks of timed automata which communicate via channels and/or shared variables. The timed automata for flooding are set out at Fig. 1 and Fig. 2 which together define the formal model of the flooding protocol.

We use variables `can_hear`, `senders`, and `landscape` to record respectively the connectivity in the network, the identity of the nodes which are ready to transmit the message, and the signals currently broadcast in the landscape. For example `landscape[id]` is 1 means that the message was garbled, and `landscape[id]` is 2 means the message was received successfully. At the start of each round the landscape is cleared of messages (`landscape[id]` is 0).

A node waits until it receives a message that is not garbled (location `Wait`, Fig. 1), and when it does it tries to forward it to its neighbours. A successful transmission is modelled by the sender putting its identity into buffer `aux_id`, an action which triggers the landscape recording the transmission locations to be updated. Before sending however, a node checks whether a message was received in the current round, and that being the case (guard `landscape[id]!=0`), it backs off, with

[1]Using logical phases is reasonable for a significant class of multi-hop, wireless networks since in many cases the propagation time is insignificant to the transmission time [5].

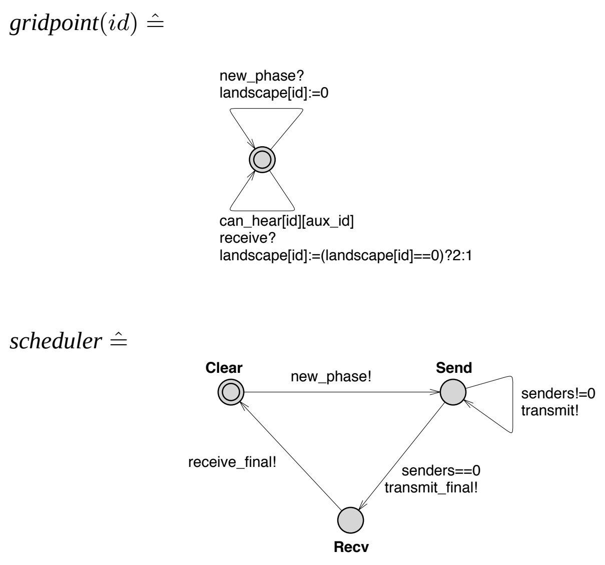

Fig. 2. The gridpoints and the scheduler

a backoff counter set nondeterministically. Note, that in this model we have used nondeterminism (rather than probability) in the backoff procedure since we are only interested in the existence of a failing behaviour in the network. Later sections addressing quantitative evaluation use probabilistic models.

In Fig. 2 we set out models for the gridpoints and the scheduler. The former facilitates a synchronous assignment to the landscape, whilst the scheduler orchestrates the causal relationships between the network activities. Finally the formal model is the parallel composition of the scheduler, the nodes and the positions.

B. *Flooding is unreliable*

One of Cardell-Oliver's key findings [5] was to prove that there exists a counterexample that demonstrates the unreliability of flooding. To obtain the same result by model checking we set up a small example of a grid network consisting of four nodes. We used the Uppaal model checker to test whether all nodes could receive a message originating at node 0. The counterexample, interpreted at Fig. 3, indicates that node 3 never receives the message, as the message forwarded by nodes 1 and 2 interfere. The vulnerability is (of course) the fact that receiving nodes always transmit their message exactly once, and in a topology admitting overlapping transmission ranges a collision is inevitable.

In this section we have shown how to use standard non-probabilistic model checking to obtain quick feedback that shows the limitations of a protocol. Although the protocol was probabilistic, unreliability could be proven by traditional means, using a model in which probabilities were replaced by nondeterminism. In the next section we consider formal analysis of quantitative properties.

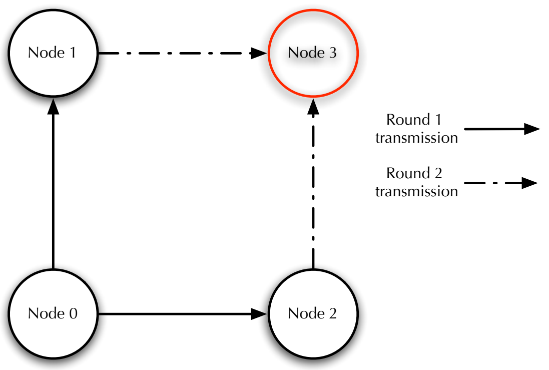

Fig. 3. A small grid network, depicting the message clash.

Skip	$\mathsf{wp}.\mathsf{skip}.E \mathrel{\hat{=}} E$,
abort	$\mathsf{wp}.\mathsf{abort}.E \mathrel{\hat{=}} 0$,
Miracle	$\mathsf{wp}.\mathsf{magic}\ .E \mathrel{\hat{=}} 1$,
Assignment	$\mathsf{wp}.(x := f).E \mathrel{\hat{=}} E[x := f]$,
Sequential composition	$\mathsf{wp}.(r; r').E \mathrel{\hat{=}} \mathsf{wp}.r.(\mathsf{wp}.r'.E)$,
Probabilistic choice	$\mathsf{wp}.(r\ {}_p{\oplus}\ r').A \mathrel{\hat{=}} p \times \mathsf{wp}.r.E + (1{-}p) \times \mathsf{wp}.r'.E$,
Demonic choice	$\mathsf{wp}.(r \| r').A \mathrel{\hat{=}} \mathsf{wp}.r.E \sqcap \mathsf{wp}.r'.E$,
Boolean choice	$\mathsf{wp}.(\mathsf{if}\ G\ \mathsf{then}\ r\ \mathsf{else}\ r').E \mathrel{\hat{=}} [G] \times \mathsf{wp}.r.E + [\neg G] \times \mathsf{wp}.r'.E$,
Guarded command	$\mathsf{wp}.(G \rightarrow r).E \mathrel{\hat{=}} [G] \times \mathsf{wp}.r.E + [\neg G]$,
Iteration	$\mathsf{wp}.(\mathsf{lt}\ G \rightarrow r\ \mathsf{tl}).E \mathrel{\hat{=}} (\mu X \bullet [\neg G] \times E + [G] \times \mathsf{wp}.r.X)$.

The meaning of a program is given by wp, which takes the syntax of a program and maps it to a functional of type $\mathcal{E}S \rightarrow \mathcal{E}S$. Here E is a (bounded) expectation (random variable) in $\mathcal{E}S$, f is a function of the state, and $\sqcap$ is the pointwise minimum in the interpretations on the right. The real p is restricted to lie between 0 and 1. This semantics focusses on properties as they can be expressed via random variables.

Fig. 4. Structural definitions of wp for pGCL.

III. Nondeterminism: extremal bounds on performance-style analysis

In this section we summarise probabilistic action systems [19], a framework for the formal analysis of probabilistic distributed systems; we illustrate our definitions with the development of a wireless protocol in Sec. IV below. Our framework allows the specification of randomised behaviour at various levels of abstraction, and includes a "built-in" theory of refinement which captures formally how properties between levels can be compared. Although the approach encourages a "top-down" development style of analysis [1], [2], it also contributes to the management of the so-called "state space explosion" problem of model-checking, as the rapidly increasing state space size correlating with more detailed models means that the only feasible formal treatment is via a formal abstraction.

A. Probabilistic action systems

Action systems [2] are a "state-based" formalism for describing so-called reactive systems, *viz*. systems that may execute indefinitely. Although others [11], [25] have added probability to action systems, our work is most closely related to Morgan's version of labelled probabilistic action systems [22], which has been extended in various ways [19] described below.

A (probabilistic) action system consists of a (finite) set of labelled guarded commands, together with a distinguished command called an initialisation. An action system is said to *operate* over a state space S, meaning that the variables used in the system define its state space. Operationally an action system first executes its initialisation, after which any labelled action may "fire" (provided its guard is true) by executing its body, and in so doing the state is updated according to any probability distribution specified in its body. (We reserve the label τ for so-called "hidden" actions, described below.) Actions may continue to fire indefinitely until all the guards are false; if more than one guard is true then any one of those actions may fire nondeterministically, thus at their most general action systems are probabilistic and nondeterministic in the style of Markov Decision Processes.

One of the aims of formal model specification is to be able to extract properties, and in the case of probabilistic systems those properties will typically be quantitative expressing the partial or "expected" satisfaction of some logical property. Our formal semantics given at Fig. 4 sets out the meaning of an action system in the form of a "probabilistic test" articulated as the expected value relative to a random variable over the state. This type of semantics, first suggested by Kozen [15] for purely probabilistic programs, and developed by Morgan and McIver [21] (to include for example nondeterminism) is expressive enough to specify many quantitative properties including probabilistic temporal logic pCTL [21], expected reachability times [7] and long-run averages [20]. For example demonic nondeterminism, can be thought of as being resolved by a "minimal-seeking demon", providing (lower bound) guarantees on all program behaviour and properties, such as is expected for total correctness.

Whilst there are many other formalisms treating probability and nondeterminism [14], [24] those approaches tend to deal directly with operational features of systems, compared to action systems presented here, whose domain-theoretical basis exposes the (standard) mathematical structures underlying the probabilistic and nondeterministic features of the semantics. Moreover the inclusion of non-standard "programs" such as miracles admits the development of program algebra which often promotes simplicity in proofs of general system properties [21]. Meanwhile the operational semantics is essentially equivalent to that of the PRISM probabilistic model checker [16], so that where possible properties may be computed automatically.

We make use of the following definitions.

- Given a family $\mathcal{I}$ of commands we write $\|_{i:\mathcal{I}} C_i$ for the generalised demonic (nondeterministic) choice over the family, and $\sum_{i\in\mathcal{I}} C_i @ p_i$ for the generalised probabilistic choice (where $\sum_{i\in\mathcal{I}} p_i \leq 1$).
- For action system P and label a we write P_a for the generalised choice of all actions labelled with a, and P_i for its initialisation. The set of labels (labelling actions in P) is denoted $\alpha.P$, and called P's *alphabet*. Thus $\mathsf{wp}.P_a.E.s$ is the greatest guaranteed expected value of E from execution of P_a, when s satisfies the guard of P_a.
- We use $\{G\}$ for the "coercion", defined $\mathsf{skip} \triangleleft G \triangleright \mathsf{magic}$, so that the guarded command $G \rightarrow P$ may be written equivalently as $[G]; P$.
- We say that a command is *normal* if it is of the form of a generalised probabilistic choice over standard (non-probabilistic) commands F_i, i.e. of the form $\sum_{i\in\mathcal{I}} F_i \ @p_i$, where the F_i are standard.

The next definition sets out how to execute two commands simultaneously.

Definition 1: Given normal guarded commands $C \mathrel{\hat{=}} G \rightarrow \mathit{Prog}$ and $C' \mathrel{\hat{=}} G' \rightarrow \mathit{Prog}'$, we define their *composition* as follows.

$$C \otimes C' \quad \hat{=} \quad (G \wedge G') \rightarrow \sum_{(i,j)\in I\times J} (F_i \otimes F'_j) \ @(p_i \times p'_j) \ ,$$

where $\mathit{Prog} = \sum_{i\in I} F_i \ @p_i$ and $\mathit{Prog}' = \sum_{j\in J} F'_j \ @p'_j$, and $\mathsf{wp}.F \otimes F'$ is given by the *fusion* operator of Back and Butler [4]. In the case that F and F' operate over distinct state spaces (as in our case study) $F \otimes F'$ is equivalent to $F \ ; \ F'$.

B. Structural language features: modularity and hiding

Action systems are usually structured as a set of separate *modules*, where each module is itself an action system, with (normally) a state space independent from that of the other modules. In the complete system modules synchronise on shared actions, whilst all other actions fire independently, interleaving with any others.

Definition 2: Given action systems P and Q in which all actions are normal; we define their *parallel composition* $P||Q$ as follows.

1) $P||Q$ operates over the union of the two state spaces, and $\alpha.(P||Q) = \alpha.P \cup \alpha.Q$;
2) $(P||Q)_i \quad \hat{=} \quad P_i \otimes Q_i$;
3) $(P||Q)_b \quad \hat{=} \quad$ if $(b \in \alpha.P)$ then P_b else Q_b, for $b \notin \alpha.P \cap \alpha.Q$;
4) $(P||Q)_a \quad \hat{=} \quad \|_{\{P^a\in P, Q^a\in Q\}} P^a \otimes Q^a$, for $a \in \alpha.P \cap \alpha.Q$, where P^a and Q^a are the individual a-labelled actions belonging to P and Q respectively.

Our second feature is *hiding*, which is a standard technique in process algebras semantics allowing a system to be understood in a "layered" manner.

Definition 3: Given a labelled action system P, and a set of labels H, we define the action system $P\backslash H$ as follows:

1) $P\backslash H$ operates over the same state space as P, and $\alpha.(P\backslash H) \mathrel{\hat{=}} (\alpha.P) - H$;

2) $(P \backslash H)_a \hat{=} P_a$, if $a \notin H$;
3) $(P \backslash H)_\tau \hat{=} P_\tau \| \ (\|_{h \in H} P_h)$.

Observe that (according to Def. 2) hidden actions do not synchronise.

C. *Refinement and abstraction: scaling up formal analysis*

Refinement and abstraction are the standard techniques for coping with complexity in large computing systems [1], [13]. An abstract "high level" specification, for example, effectively summarises gross system behaviours, in such a way that sufficient detail remains for interesting properties to be tested. Thus such a specification serves the dual purpose of enforcing correct operation in a concrete system (formally verified as a refinement), and allowing those crucial properties to be computed automatically, which in many cases the concrete models would be too "resource needy" to be investigated directly.

Our definition of refinement set out at Def. 4 uses the well-versed technique of "simulation" used in process algebras [23], [12] and elsewhere [21], [9]. A specification action system must be able to "match", or "simulate" all named actions of a refinement; however since hidden actions cannot be observed directly, we merely require that the overall effect of hidden actions to be appropriately correlated. Here we use the iteration construct set out at Fig. 4 to express an arbitrary number of hidden actions, noting that if infinitely many of them occur (with probability greater than 0) the action system is said to "diverge"; such behaviour is equivalent to abort.

Definition 4: Given action systems P and Q with global variables g, and local variables a and c respectively, we say that $P \preceq_g Q$, or Q *refines* P *with respect to* g if there is a standard (i.e. non-probabilistic) *simulation* program *rep*, mapping variables a to c such that

1) P_i ; *rep* $\sqsubseteq$ Q_i;
2) P_a ; *rep* $\sqsubseteq$ *rep* ; Q_a, for $a \in \alpha.Q$;
3) (It P_τ tI) ; *rep* $\sqsubseteq$ *rep* ; (It Q_τ tI)
4) skip_g $\sqsubseteq$ *rep* ; skip_g ,

where skip_g is a special "do nothing" program which projects the state onto that defined by the global variables g.

Typically *global* variables are used to specify observable behaviours, whereas *local* variables are used for "internal" processing by the individual modules. In this paper we are interested in estimating the delays within the network, thus we use a reserved variable t which is incremented whenever some time critical action is fired. We may compute the expected message delay by computing the expected value of the variable t once all the messages have been received. [2] Other global properties could be the expected number of nodes which receive a message in a flooding protocol [10].

[2]It has been shown elsewhere [17] that using discrete time in this way is a sufficient abstraction from real time for the purposes of some performance-style properties including expected reachability times.

Most significantly however is that the upper- and lower bound estimates for expected delays are preserved by refinement, so that if a specification (or less detailed) model of the system satisfies a property then so too does the refined model. The next theorem sets out the details.

Theorem 5: If $A \preceq_t B$ then an upper (lower) bound on expected delay on A is an upper (lower) bound on expected delay on B. *Proof:* This follows from standard results about expected values, whose proof appears elsewhere. [21], [17], [7]. ■

It turns out that refinement, parallel composition and hiding all work well together according to the notions of compositionality, whose main advantages are that systems may be specified and refined in a modular fashion. Again this is a widely-used idea [14], [12] — the next theorem sets out the details for probabilistic action systems.

Theorem 6: Suppose $A \preceq_t A'$. If A and B operate over independent (finite) state spaces, do not diverge and at least one of them is standard then the following refinements are valid,

$$A || B \quad \preceq_t \quad A' || B \tag{1}$$
$$A \backslash H \quad \preceq_t \quad A' \backslash H \ , \tag{2}$$

where H is some subset of actions. [3]

IV. Stepwise refinement: combining probabilistic model checking and proof

In this section we illustrate our definitions by demonstrating how a stepwise specification and refinement of various characteristics of wireless communication may be analysed formally, using a combination of proof and model checking. We consider a small topology based on a multi-hop wireless network used to study energy-efficient protocols [19]. This

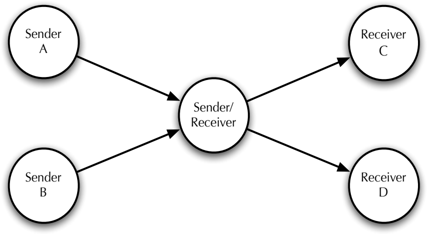

Fig. 5. A small network

small network depicted at Fig. 5 consists of two sender nodes, A and B, whose aims are to send messages to receiver nodes C and D via an intermediary. The "high level" behaviour is very simple — for example the senders and receivers,

[3]Found at http://www.comp.mq.edu.au/∼anabel/isola06.pdf.

whose formal models are set out at Fig. 7 send by firing a *send* action and wait for an acknowledgement (signalled by an *ack* action). Additionally both nodes share a *clash* event, indicating that for this topology it is possible for messages sent simultaneously (by A and B) to collide. Similarly the behaviour of the intermediary at Fig. 6 is also simple — it receives and acknowledges messages from the senders, and then forwards them to the receivers.

In our specification all the delays are contained in the module *Channel*, set out in Fig. 8. In the case of a message collision, the event *clash* is concomitant with the probabilistic update of the state of the channel (either *block*ed or *clear*). However, even in the worst case, standard probability threory implies that with probability 1 the *Channel* must become clear. A second delay is caused by the wireless communication itself which is well-known to exhibit probabilistic behaviour [5], and that too is modelled by a probabilistic event, which is forced to fire between a *send* and an *ack*. Note however that the nondeterminism means that a *range* of delays has been specified (namely those determined by parameters r and q).

$$\mathit{SR} \mathrel{\hat{=}}$$

$$\left(\begin{array}{ll}
\multicolumn{2}{l}{\textbf{var } r\colon \{\mathit{listen}, \mathit{ack}, \mathit{forward}\},} \\
\multicolumn{2}{l}{\textbf{initially } r := \mathit{listen}} \\
\,[\!]_{i:\,\{a,b\}}\mathit{send}_i: & (r = \mathit{listen}) \rightarrow r := \mathit{ack} \\
\,[\!]_{i:\,\{a,b\}}\mathit{ack}_i: & (r = \mathit{ack}) \rightarrow r := \mathit{forward} \\
 & \\
\,[\!]_{i:\,\{c,d\}}\mathit{send}_i: & (r = \mathit{forward}) \rightarrow r := \mathit{ack} \\
\,[\!]_{i:\,\{c,d\}}\mathit{ack}_i: & (r = \mathit{ack}) \rightarrow r := \mathit{listen}
\end{array}\right)$$

Fig. 6. A *sender/receiver* node.

$$\mathit{Sender}_A \mathrel{\hat{=}}$$

$$\left(\begin{array}{ll}
\multicolumn{2}{l}{\textbf{var } s_a\colon \{\mathit{wait}, \mathit{sent}, \mathit{recv}\}} \\
\multicolumn{2}{l}{\textbf{initially } s_a := \mathit{wait}} \\
\mathit{send}_a: & (s_a = \mathit{wait}) \rightarrow s_a := \mathit{sent} \\
\mathit{ack}_a: & (s_a = \mathit{sent}) \rightarrow s_a := \mathit{recv} \\
\mathit{clash}: & (s_a = \mathit{wait}) \rightarrow \mathsf{skip}
\end{array}\right)$$

$$\mathit{Receiver}_C \mathrel{\hat{=}}$$

$$\left(\begin{array}{ll}
\multicolumn{2}{l}{\textbf{var } s_a\colon \{\mathit{wait}, \mathit{sent}, \mathit{recv}\}} \\
\multicolumn{2}{l}{\textbf{initially } s_a := \mathit{wait}} \\
\mathit{send}_c: & (s_c = \mathit{wait}) \rightarrow s_c := \mathit{sent} \\
\mathit{ack}_a: & (s_c = \mathit{sent}) \rightarrow s_c := \mathit{recv}
\end{array}\right)$$

Fig. 7. The behaviour of nodes A and C. Nodes B and C are similar.

Finally we may define the network as the parallel composition of all the components,

$$\begin{array}{l}
\mathit{Network} \quad \hat{=} \\
\mathit{Sender}_A \;||\mathit{Sender}_B \;||\; \mathit{SR} \;||\; \mathit{Receiver}_C||\mathit{Receiver}_D \;||\mathit{Channel}\;.
\end{array}$$

At this point, even though many details are not included in the system, we may investigate the full range of possible

$$\mathit{Channel} \mathrel{\hat{=}}$$

$$\left(\begin{array}{ll}
\multicolumn{2}{l}{\textbf{var } c\colon \{\mathit{block}, \mathit{clear}, \mathit{forward}, \mathit{sending}, \mathit{ac}, \mathit{done}, \mathit{waiting}\},} \\
\multicolumn{2}{l}{\; t\colon \{0 \ldots T\}} \\
\multicolumn{2}{l}{\textbf{initially } c := (\mathit{block} \;[\!]\; \mathit{clear}) \;;\; t := 0} \\
\mathit{clash}: & (c = \mathit{block}) \rightarrow t := t + 1; \\
 & \qquad \mathsf{skip} \;{}_p\!\oplus c := \mathit{clear} \\
\,[\!]_{i:\,A}\mathit{send}_i: & (c = \mathit{clear} \vee \mathit{waiting}) \rightarrow c := \mathit{sending} \\
\,[\!]_{i:\,A}\mathit{send}_i: & (c = \mathit{forward}) \rightarrow c := \mathit{ac} \\
\mathit{delay}: & (c = \mathit{sending}) \rightarrow t := t + 1; \\
 & \qquad c := \mathit{sending} \;{}_q\!\oplus \mathit{forward} \\
\mathit{delay}: & (c = \mathit{sending}) \rightarrow t := t + 1; \\
 & \qquad c := \mathit{sending} \;{}_r\!\oplus \mathit{forward} \\
\,[\!]_{i:\,A}\mathit{ack}_i: & (c = \mathit{ac}) \rightarrow c := \mathit{done} \\
\,[\!]_{i:\,A}\mathit{ack}_i: & (c = \mathit{done}) \rightarrow c := \mathit{waiting}
\end{array}\right)$$

Fig. 8. The channel with delays: nondeterminism specifies a range of delays.

probabilistic behaviour over message delays using the PRISM model checker; the results are displayed at Fig. 9. As we shall see below these results apply equally to the more detailed system model described in the next section.

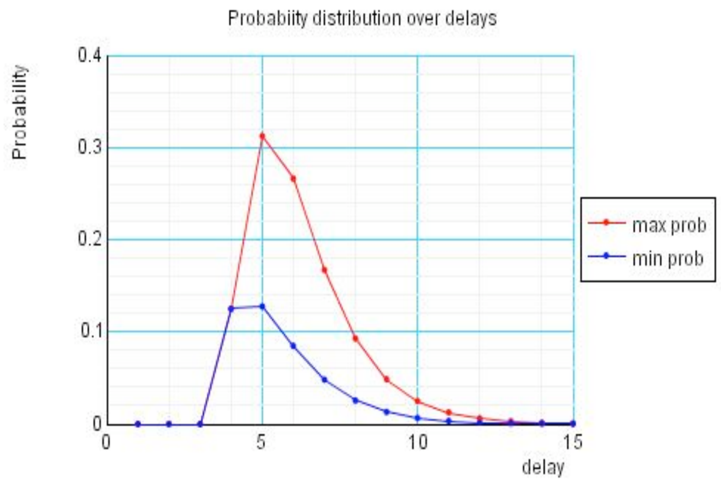

Fig. 9. Maximum and minimum probabilities for message delay.

A. Refinement: introducing detail

In this section we introduce the randomised backoff procedure in each sender (namely the cause of the delay incurred by event *clash*). Specifically. when each process detects that it is in collision with another — the details of how they do this have still been suppressed[4] — it sets its “backoff counter” to some random number and then counts down. As there is a good chance that the two backoff counters will be set to different values, the implication is that one of the senders will try to re-send at a time when the other is still “counting down”.

In Fig. 10 and Fig. 11 we set out the refined *Sender*$'$ and *Channel*$'$ — with now the backoff procedure residing in the *Sender*$'$. The new *Network*$'$ is similarly the sequential

[4]These details may be added in a further refinement step.

$$Sender'_A \mathrel{\hat{=}}$$

$$\left(\begin{array}{ll} \textbf{var}\ s_a\colon \{wait, sent, recv\},\ bc_a\colon \{0 \ldots N\} & \\ \textbf{initially}\ s_a := wait;\ bc_a := 0 & \\ \quad send_a: & (s_a = wait) \rightarrow s_a := sent \\ \quad ack_a: & (s_a = sent) \rightarrow s_a := recv \\ \quad clash: & (bc_a = 0 \wedge s_a = wait) \rightarrow flip(bc_a) \\ \quad tick: & (bc_a > 0) \rightarrow bc_a := bc_a - 1 \\ \quad tick: & (bc_a = 0 \wedge s_a = recv) \rightarrow \mathsf{skip} \end{array}\right)$$

where $flip(x)$ sets the value of variable x to be n with probability $1/p^{n+1}$, if $n < N$, and to N with probability $1-(1/p)^N$.

Fig. 10. A sending station with a backoff procedure.

$$Channel' \mathrel{\hat{=}}$$

$$\left(\begin{array}{ll} \textbf{var}\ c'\colon \{clear, forward, sending, ac, done\},\ t\colon \{0 \ldots T\} & \\ \textbf{initially}\ c' := clear\ ;\ t := 0 & \\ \quad clash: & t := t+1 \\ \quad [\!]_{i\colon A} send_i: & (c = clear) \rightarrow c := sending \\ \quad [\!]_{i\colon A} send_i: & (c = forward) \rightarrow c := ac \\ \quad delay: & (c = sending) \rightarrow t := t+1; \\ & \qquad c := sending\ {}_q\!\oplus forward \\ & \\ \quad delay: & (c = sending) \rightarrow t := t+1; \\ & \qquad c := sending\ {}_r\!\oplus forward \\ & \\ \quad [\!]_{i\colon A} ack_i: & (c = ac) \rightarrow c := done \\ \quad [\!]_{i\colon A} ack_i: & (c = done) \rightarrow c := clear \end{array}\right)$$

Fig. 11. The revised channel: it counts clashes and handles delays.

compostion of all the components:

$$\begin{array}{rcl} Network' & \hat{=} & Sender'_A\ ||Sender'_B||SR\ || \\ & & Receiver_C||\ Receiver_D\ ||\ Channel'\ . \end{array}$$

Next, we use a formal proof in Lem. 7 to show that *Network*$'$ is a "refinement" of *Network*, in the sense that all behaviours of *Network*$'$ can be accommodated by *Network* — the consequence of this is that the bounds on *Network*'s probabilistic behaviour also apply to *Network*$'$.

Lemma 7:

$$Network \quad \preceq_t \quad Network'\backslash\{tick\}\ .$$

Proof: (Sketch.) First we prove that $Channel \preceq Channel''\backslash\{tick\}$, where *Channel*$''$ is the same as *Channel*$'$ except for the event *clash*; we also introduce action *tick*. The actions are defined as follows:

$$\left(\begin{array}{ll} clash: & (bc_a = 0 \wedge bc_b = 0) \rightarrow flip(bc_a); flip(bc_b); \\ & \\ & \qquad (c' := clear) \triangleleft (bc_a \neq bc_b) \triangleright \mathsf{skip} \\ tick: & (bc_a > 0 \wedge bc_b > 0) \rightarrow \\ & \\ & \qquad bc_a, bc_b := bc_a - 1, bc_b - 1 \\ [\!]_{i\colon A} tick: & (c = waiting \wedge bc_i > 0) \rightarrow bc_i := bc_i - 1 \end{array}\right).$$

To prove the refinement we use

$$\begin{array}{rl} rep & \hat{=}\ \mathsf{if}\ \ (c = block)\ \mathsf{then}\ \ (bc_a, bc_b :\in (bc_a = bc_b) \\ & \qquad \mathsf{else}\ \ (bc_a, bc_b :\in (bc_a \neq bc_b)\ , \end{array}$$

in Def. 4. Next we use Thm. 6 to show that

$$\begin{array}{l} Network \preceq_t Sender_A\ ||Sender_B|| \\ \qquad SR\ ||\ Receiver_C||\ Receiver_D\ ||\ (Channel''\backslash\{tick\})\ , \end{array}$$

noting that the hidden events must terminate with probability 1, and that all of the probabilistic activity resides in the channel Now we distribute the $\backslash\{tick\})$ to be the outermost operator (possible since *tick* does not appear in the other modules), and finally we redistribute the actions using Def. 2, and note (informally) that *waiting* may only occur after one of the messages has been sent. ■

Aside from establishing the refinement, one advantage of proofs such as these is that we learn more about the operation of the protocol. In this case we discover, for example, that it is safe for the *Sender*s to suspend their countdown in the backoff procedure for an interval corresponding to the time it takes for the other to receive its acknowledgement, and thus we have verified this strategy advised in the IEEE802.11 standard [26].

V. Conclusions

In this paper we have described how to use standard formal tools and techniques for the analysis of some performance and correctness criteria for wireless networks. Besides the benefits of formality making the assumptions explicit, the exhaustive search of model checking can illustrate effectively weaknesses in the system, and provide upper and lower bounds on quantitative behaviour without the need for using large numbers of simulations. Moreover, techniques based on action systems may be developed to investigate optimal protocol behaviour, in terms of trading off energy requirements with performance [18].

There is however much work to be done in building more realistic formal models, and tailoring techniques to specific performance criteria for wireless networks.

Acknowledgements

This work was supported by the National ICT Australia, which is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

References

[1] J.-R. Abrial. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[2] R.-J.R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.

[3] G. Behrmann, A. David, and K.G. Larsen. A tutorial on UPPAAL. In M. Bernardo and F. Corradini, editors, *Formal Methods for the Design of Real-Time Systems: 4th International School on Formal Methods for the Design of Computer, Communication, and Software Systems, SFM-RT 2004*, number 3185 in LNCS, pages 200–236. Springer–Verlag, September 2004.

[4] M.J. Butler and C.C. Morgan. Action systems, unbounded nondeterminism and infinite traces. *Formal Aspects of Computing*, 7(1):37–53, 1995.

[5] R. Cardell-Oliver. Why Flooding is Unreliable (Extended Version). Technical Report UWA-CSSE-04-001, CSSE, University of Western Australia, 2001.

[6] David Cavin, Yoav Sasson, and Andre Schiper. On the accuracy of manet simulators.

[7] O. Celiku and A. McIver. Cost-based analysis of probabilistic programs mechanised in HOL. *Nordic Journal of Computing*, 2004.

[8] E. Clarke and J. Wing. Formal methods: state of the art and future directions. *ACM Computing Surveys*, 28(4):626–643, 1996.

[9] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proc. PAPM/PROBMIV'01*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[10] A. Fehnker and P. Gao. Formal verification and simulation for performance analysis for probabilistic broadcast protocols. In T. Kunz and S.S. Ravi, editors, *5th International Conference on AD-HOC Networks & Wireless*, LNCS, 2006.

[11] S. Hallerstede and M. Butler. Performance analysis of probabilistic action systems. 2005.

[12] J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.

[13] B. Jonsson and K.G. Larsen. Specification and refinement of probabilistic processes. In *Proc. 6th Conf. LICS*, 1991.

[14] B. Jonsson, K.G. Larsen, and Wang Yi. Probabilistic extensions of process algebras. *Handbook of Process Algebras*, (1):685–710, 2001.

[15] D. Kozen. Semantics of probabilistic programs. *Jnl. Comp. Sys. Sciences*, 22:328–50, 1981.

[16] M. Kwiatkowska, G. Norman, and D.Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. 2002. Accepted for TACAS 2002.

[17] M. Kwiatkowska, G. Norman, and J. Sproston. Probabilistic model checking of the ieee 802.11 wireless local area network protocol. In *Proc. PAPM/PROBMIV'02*, volume 2399 of *LNCS*, pages 169–187, 2002.

[18] A. McIver. mu-calculus analysis of power management in wireless networks. 2006. Under consideration for the International Symposium on Theoretical Aspects of Computing.

[19] A.K. McIver. Quantitaive refinement and model checking for the analysis of probabilistic systems. Accepted for FM 2006.

[20] A.K. McIver. A generalisation of stationary distributions, and probabilistic program algebra. In Stephen Brookes and Michael Mislove, editors, *Electronic Notes in Theo. Comp. Sci.*, volume 45. Elsevier, 2001.

[21] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Technical Monographs in Computer Science. Springer Verlag, New York, 2004.

[22] C.C. Morgan. Of probabilistic Wp and CSP. *25 years of CSP*.

[23] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *CONCUR '94*, number 836 in LNCS.

[24] Roberto Segala and Nancy Lynch. Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing*, 2(2):250–73, 1995.

[25] Kaisa Sere and Elena Troubitsyna. Probabilities in action systems. In *Proc. of the 8th Nordic Workshop on Programming Theory*, 1996.

[26] IEEE Standard. Medium Access Control (MAC) and Physical Layer Specifications. Technical report, IEEE, 1999.

[27] Wei Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, (3), 2004.