Quantitative μ-calculus analysis of power management in wireless networks

AK McIver

Dept. Computer Science, Macquarie University, NSW 2109 Australia,
and National ICT Australia **
anabel@ics.mq.edu.au

Abstract. An important concern in wireless network technology is battery conservation. A promising approach to saving energy is to allow nodes periodically to enter a "low power mode", however this strategy contributes to message delay, and careful management is required so that the system-wide performance is not severely compromised. Moreover the management of low/high power schedules is particularly difficult due to the uncertainties caused by message collisions and clock drift.
In this paper we show how to manage the low/high power scheduling in network nodes so that the time spent in the low power mode is optimised relative to the cost of message delay. Our approach is to use the quantitative modal μ-calculus which allows the specification of a quantitative performance property as a game in which a maximising player's optimal strategy corresponds to the optimal low/high power schedule.
We extend the standard results on discounted games to infinite state systems, and illustrate our results on a small case study.
Keywords: Probabilistic abstraction and refinement, structured specification and analysis of performance, probabilistic model checking.

1 Introduction

The theme of this paper is the specification and analysis of performance-style properties for wireless networks. The problem is particularly challenging in this domain because of the many sources of underlying uncertainty, including collision avoidance, clock-drift and degrading battery life [28]. We model such uncertainties with *probability* or standard *nondeterminism*, using the former when the uncertainty may be quantified and appealing to the latter when it cannot [18].

The formal investigation of probabilistic distributed systems (*i.e.* those combining both probability and nondeterminism) is normally limited to the consideration of *probabilistic temporal properties* [1], such as "the system eventually satisfies predicate *Pred* with probability at least 1/3". There are however many other performance-style quantities which cannot be specified in this way, as they are examples of the more general *stochastic parity games* [6, 15] in which two

** National ICT Australia is funded through the Australian Government's *Backing Australia's Ability* initiative, in part through the Australian Research Council.

players try to optimise a quantitative cost function according to their opposing goals.

Our principal aim in this paper is to illustrate how the quantitative modal μ-calculus [15], a generalisation of the standard μ-calculus of Kozen [12] to probabilistic systems, may be used as a convenient language for specifying and analysing such a stochastic parity game relevant to power-management schemes in wireless networks [28] — indeed careful power management is a pressing concern, where the integrity of the communication relies on preserving battery lives of the individual nodes that make up the network.

Our specific topic is how to optimise a scheme in which nodes periodically enter a low power mode during which communication via so-called "sleeping" nodes is impossible — the obvious drawback here is that messages may be severely delayed, as they are forced to wait for nodes to wake up. Thus the scheduling problem is to decide how to choose the length of a sleep time so that an interval of low power lasts as long as possible without significantly compromising throughput.

Our approach is to specify a game, via the quantitative μ-calculus, in which one player seeks to maximise power savings, whilst the other tries to minimise it; we show that computing the optimal strategies of the maximising player is equivalent to optimising the power management scheme so that the low power modes may be applied most advantageously. Our particular contributions are as follows:

1. *A proof that discounted stochastic parity games* for certain unbounded cost functions are well defined and that the players have optimal solutions which can be computed by the standard value-iteration method (Sec. 2.5). This extends the standard results, applicable only to finite state systems [15];
2. *A formal model* of a wireless communication protocol incorporating power management (Sec. 3) using labelled *probabilistic action systems* (Sec. 2.1);
3. *A novel stochastic-parity-game specification* using quantitative μ-calculus, $qM\mu$ for analysing the optimal expected time in the low power mode constrained by the relative cost of message delay (Sec. 3.1). Using this we are able to compute the optimal sleep schedules.

The advantage of using probabilistic action systems is that they come equipped with a full theory of abstraction and refinement, and thus we anticipate that a formal proof will establish that this paper's results apply even to large-scale networks [14]. Moreover action systems also give access to detailed numerical experiments using state-of-the-art probabilistic model checkers whose results may be used to determine the optimal schedule, as well as computing detailed expected delays and power savings.We illustrate our formal models in Sec. 3.2 using the the PRISM probabilistic model checker [23].

The notational conventions used are as follows. Function application is represented by a dot, as in $f.x$. We use an abstract state space S, and denote the set of discrete probability distributions over S by $\overline{S}$ (that is the sub-normalised

functions from S into the real interval $[0,1]$, where function f is sub-normalised if $\sum_{s:\,S} f.s \leq 1$). Given predicate $Pred$ we write $[Pred]$ for the *characteristic* function mapping states satisfying $Pred$ to 1 and to 0 otherwise, punning 1 and 0 with "True" and "False". The $(p, 1-p)$-weighted average of distributions d and d' is denoted $d \,{}_p\!\oplus\, d'$.

2 Probabilistic guarded commands

When programs incorporate probability, their properties can no longer be guaranteed "with certainty", but only "up to some probability". For example the program

$$coin \quad \hat{=} \quad b\colon = 1 \;{}_{2/3}\!\oplus\; b\colon = 0\;, \tag{1}$$

sets the variable b to 1 only with probability $2/3$ — in practice this means that if the statement were executed a large number of times, and the final values of b tabulated, roughly $2/3$ of them would record b having been set to 1 (up to well-known statistical confidence).

The language pGCL and its associated *quantitative logic* [18] were developed to express such programs and to derive their probabilistic properties by extending the classical assertional style of programming [20]. Programs in pGCL are modelled (operationally) as functions (or transitions) which map *initial states* in S to (sets of) probability distributions over *final states* — the program at (1) for instance has a single transition which maps any initial state to a (single) final distribution; we represent that distribution as a function δ, evaluating to $2/3$ when $b = 1$ and to $1/3$ when $b = 0$.

Since properties are now quantitative we express them via a logic of *real-valued functions*, or *expectations*. For example the property "the final value of b is 1 with probability $2/3$" can be expressed as the *expected value* of the function $[b = 1]$ with respect to δ, which evaluates to $2/3 \times 1 + 1/3 \times 0 = 2/3$.

Direct appeal to the operational semantics quickly becomes impractical for all but the simplest programs — better is the equivalent transformer-style semantics which is obtained by rationalising the above calculation in terms of expected values rather than transitions, and the explanation runs as follows. Writing $\mathcal{E}S$ for the set of all functions from S to $\mathbb{R}$[1], which we call the set of *expectations*, we say that the expectation $[b = 1]$ has been transformed to the expectation $2/3$ by the program *coin* set out at (1) above so that they are in the relation "$2/3$ is the expected value of $[b = 1]$ with respect to the *coin*'s result distribution". More generally given a program *prog*, an expectation E in $\mathcal{E}S$ and a state $s \in S$, we define $Wp.prog.E.s$ to be the expected value of E with respect to the result distribution of program *prog* if executed initially from state s [18]. We say that $Wp.prog$ is the *expectation transformer* relative to *prog*. In our example that allows us to write

$$2/3 \quad = \quad Wp.(b\colon = 1 \;{}_{2/3}\!\oplus\, b\colon = 0).[b = 1]\;.$$

[1] Strictly speaking we also include ∞ and $-\infty$ for well-definedness of fixed points [10]. All expectations used in this paper are finitary.

In the case that *nondeterminism* is present, execution of *prog* results in a *set* of possible distributions and we modify the definition of *Wp* to take account of this, depending on whether the nondeterminism is angelic or demonic. At Fig. 1 we set out the details of pGCL, a variation of Dijkstra's GCL with probability [2]. All the programming features have been defined previously elsewhere, and (apart from probabilistic choice) have interpretations which are merely adapted to the real-valued context. For example demonic nondeterminism, can be thought of as being resolved by a "minimal-seeking demon", providing guarantees on all program behaviour, such as is expected for total correctness. Angelic nondeterminism, on the other hand, can be thought of as a maximal-seeking angel, and provides an upper bound on possible behaviours. Finally *probabilistic choice* selects the operands at random with weightings determined by the probability parameter p. 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 $\sum_{i\in\mathcal{I}} p_i \leq 1$).

• 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 only containing angelic nondeterminism, *i.e.* of the form $\sum_{i\in\mathcal{I}} F_i \ @p_i$, where the F_i are standard, possibly angelic (but nowhere demonic) commands.

• We say that a pair of states (s_0, s') are *related via (normal) command* C if it is possible to reach s' from initial s_0 via C with some non-zero probability. [3]

We shall need to be able to compose the effect of "running" commands simultaneosly, and the next definition sets out how to do it.

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

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

where $prog = \sum_{i\in I} F_i \ @p_i$ *and* $prog' = \sum_{j\in J} F'_j \ @p'_j$*, and* $Wp.F \otimes F'$ *is given by the* fusion *operator of Back and Butler [5]. In the case that* F *and* F' *operate over distinct state spaces (as in our case study)* $F \otimes F'$ *is equivalent to* $F \ ; \ F'$*.*

2.1 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 [8, 24] have added probability to action systems, our work is most closely related to Morgan's version of labelled probabilistic action systems [19], which have been extended in various ways [14] described below.

[2] The language also includes abortion, miracles and iteration, but we do not need them here.

[3] For non-miraculous commands this condition is expressed as $Wp.C.[s = s'].s_0 > 0$.

Skip	$Wp.\mathsf{skip}.E \mathrel{\hat{=}} E$,
Assignment	$Wp.(x\colon = f).E \mathrel{\hat{=}} E[x\colon = f]$,
Sequential composition	$Wp.(r;r').E \mathrel{\hat{=}} Wp.r.(Wp.r'.E)$,
Probabilistic choice	$Wp.(r\ {}_p\!\oplus r').A \mathrel{\hat{=}} p \times Wp.r.E + (1-p) \times Wp.r'.E$,
Demonic choice	$Wp.(r \sqcap r').A \mathrel{\hat{=}} Wp.r.E \sqcap Wp.r'.E$,
Angelic choice	$Wp.(r \sqcup r').A \mathrel{\hat{=}} Wp.r.E \sqcup Wp.r'.E$,
Boolean choice	$Wp.(r \triangleleft G \triangleright r').E \mathrel{\hat{=}} \overline{G} \times Wp.r.E + \overline{\neg G} \times Wp.r'.E$,
Guarded command	$Wp.(G \to r).E \mathrel{\hat{=}} \overline{G} \times Wp.r.E + \overline{\neg G} \times \infty$,

E is a bounded expectation in $\mathcal{E}S$, and f is a function of the state, and $\sqcap$, $\sqcup$ are respectively the pointwise minimum and maximum in the interpretations on the right. The real p is restricted to lie between 0 and 1.

Fig. 1. Structural definitions of Wp for pGCL.

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" if its guard is true by executing its body. 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, demonically.

In Fig. 2 we set out a small example of a probabilistic action system *Guesser* which operates over the state defined by its variables x, y, d and t. They are all initialised to 0 or 1, and then action a or b fires depending on whether d is 0 or 1; the random flipper used to govern the setting of d degrades over time so that it becomes more likely for d to change whenever an action fires. In terms of actions, *Guesser* executes strings of a's and b's, whose relative frequency depends on the probabilistic selection of d.

$$
\mathit{Guesser} \mathrel{\hat{=}} \left(\begin{array}{l}
\textbf{var}\ x, y, d\colon \{0,1\}, t\colon \mathbb{N} \\
\textbf{initially}\ x, d\colon = 0\ ;\ y, t\colon = 1 \\
a:\ (d = 0) \to\ t\colon = t+1\ ;\ x\colon = 1\ {}_{1/3}\!\oplus\ x\colon = 0\ ; \\
\qquad\qquad\qquad\quad d\colon = 1\ {}_{(t-1)/t}\!\oplus\ d\colon = 0 \\
b:\ (d = 1) \to\ t\colon = t+1\ ;\ y\colon = 1 \sqcup y\colon = 0\ ; \\
\qquad\qquad\qquad\quad d\colon = 1\ {}_{1/t}\!\oplus\ d\colon = 0
\end{array}\right)
$$

Fig. 2. Guessing a random number.

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*. The semantics of an action system is given by pGCL set out at Fig. 1, so that, for example, $Wp.P_a.E.s$ is the

greatest guaranteed expected value of E from execution of P_a, when s satisfies the guard of P_a.

Action systems are normally constructed in a modular fashion from a set of separate *modules*. Each module is itself an action system, with (normally) a state space independent from that of the other modules. In the complete system however the modules operate essentially independently, except for having to synchronise on shared actions.

2.2 Synchronising actions

Synchronisation is defined so that all action systems participating in a parallel composition simultaneously fire their shared actions — in this mode the demonic nondeterminism (arising from possibly overlapping guards) is resolved first, followed by any probability, and last of all any angelic nondeterminism (in the bodies). 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. $(P||Q)_b \quad \hat{=} \quad \mathsf{if}\ (b \in \alpha.P)\ \ \mathsf{then}\ \ P_b\ \mathsf{else}\ Q_b, \quad$ *for* $b \in \alpha.(P||Q) \backslash (\alpha.P \cap \alpha.Q)$;
4. $(P||Q)_a \quad \hat{=} \quad [\!]_{\{P^a \in P, Q^a \in Q\}} P^a \otimes Q^a, \quad$ *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.*

Finally we also make use of the convenient message-passing syntax found elsewhere [9, 11] and set out at Fig. 3.

Sending x	$chan!x :\ G\ \rightarrow P$	$\hat{=}$	$[\!]_{(y:\,\mathcal{Y})}\, chan :\ G \wedge (x = y)\ \rightarrow P$
Receiving x	$chan?x :\ G'\ \rightarrow Q(x)$	$\hat{=}$	$[\!]_{(y:\,\mathcal{Y})}\, chan :\ G' \wedge (x = y)\ \rightarrow Q(y)$

$\mathcal{Y}$ is the set of values over which x ranges.

Fig. 3. Message-passing.

In this section we have described pGCL and action systems as a basis for describing probabilistic reactive systems. In the next section we consider how to specify and analyse quantitative performance-style properties of those systems.

2.3 Property specification and optimisation problems

Probabilistic temporal logic [1] is the well-known generalisation of standard temporal logic for investigating properties such as "the system will eventually establish condition *Pred* with probability 1/2", and indeed such properties can be readily expressed using the expectation-transformer semantics and pGCL [18].

Besides standard temporal logic however there other quantitative properties pertinent to the analysis of performance, and in this section we investigate one such example. To begin, we consider the expression

$$\diamond A \quad \hat{=} \quad (\mu X \cdot A \sqcup \mathit{Wp.System.X}) \ , \tag{2}$$

where A is an expectation, and *System* an action system. The term $(\mu X \dots)$ refers to the least fixed point of the expectation-to-expectation function $(\lambda X \cdot A \sqcup \mathit{Wp.System.X})$ with respect to $\leq$, lifted pointwise to real-valued functions. If A is any expectation then the expression at (2) is well-defined for any action system *System* [21], since in this case the definitions set out at Fig. 1 guarantee a least fixed point. [4]

To understand the interpretation, consider for example when A is the expectation defined by the "random variable"

$$\mathsf{if}\ (x = y \wedge d = 1 \wedge t < 5)\ \mathsf{then}\ 1\ \mathsf{else}\ 1/8 \tag{3}$$

and *System* is *Guesser*. Here (2) gives a numerical value which averages the possible outcomes of repeatedly executing *Guesser*, so that a payoff of 1 is given if ever $(x = y \wedge d = 1 \wedge t < 5)$ is established, and only 1/8 otherwise. To see that, we extend the basic idea of probabilistic temporal logic, which relies on the observation that a probabilistic (action) system generates (a set of) distributions[5] over so-called "computation paths", where a *computation path* is the sequence of states through which the computation passes. Since formulae such as the above define measurable functions over the space of path distributions, the expected value is well defined (for bounded expectations). We discuss the above interpretation in more detail below.

Expectations like (3) are relevant whenever some behaviours are ranked more highly than others in the context of the problem, such as in the case of the expected time to establish a condition, or in a risk-style analysis.

2.4 A game view

In more complicated situations we sometimes combine both angelic and demonic nondeterminism with probabilistic choice in *System*. Here (2) is also well-defined, although the operational interpretation is now somewhat more involved.

When all three — demonic, angelic and probabilistic nondeterminism — are present, the formula on the right-hand side of (2) describes a game [16], in which two players *Max* and *Min* decide how to resolve respectively the angelic and demonic nondeterminism in *System*, and (in this particular game) *Max* also decides when to terminate for an immediate "payoff" defined by A. The aims of *Max* and *Min* are opposite [6] — as the ultimate payoff goes to *Max* he tries to

[4] The reals form a complete partial order if augmented with $+\infty$ [10] and, by Tarski's result [26], that is sufficient to guarantee a least fixed point.

[5] Usually called the Borel probability algebra over computation paths.

[6] This is effectively a *zero-sum* game.

maximise it, whereas *Min* tries to minimise it. We call the expected payoff the *value* of the game.

We call a particular resolution of the nondeterminism (by a player) to be a *strategy* (of that player), and it turns out that in this kind of game (where both players have full knowledge of the state) the *Wp*-semantics gives a quantitative result which may be interpreted as the optimal payoff which *Max* may achieve against any strategy played out by *Min* [16], and furthermore that strategy is equivalent to replacing all angelic choices with a Boolean choice which describes which branch *Max* should take from any particular state.

More generally when A is a (fixed) bounded expectation, both players may find optimal strategies in their play *viz.* each nondeterministic choice (either angelic or demonic) in *System* may be replaced by a Boolean choice so that the resulting $System'$ satisfies

$$(\mu X \cdot A \sqcup Wp.System.X) \quad = \quad (\mu X \cdot A \sqcup Wp.System'.X) \ .$$

The strategies are *memoryless*, only depending on the current state [16]. For example the command labelled b in Fig. 2 may be replaced by

$$\begin{array}{l} b: \ (d = 1) \rightarrow \ t := t + 1 \ ; \ (y := x \lhd \ (t > 3) \ \rhd y := 0) \ ; \\ \qquad\qquad\qquad\quad d := 1 \, {}_{1/t}\!\oplus d := 0 \ , \end{array} \tag{4}$$

for it defines the optimal strategy that player *Max* should play for optimising his expected overall payoff when A is defined by (3). In this case the optimal resolution of the original angelic choice is to set y to 0 if $t < 3$. We should note here that the optimal strategies depend on A , and that changing the payoff function may result in a change in the the optimal strategy, even when the underlying *System* is unchanged.

The above facts may be applied to system design in the following way. Suppose that the design of a protocol, besides satisfying some qualitative specification, is also required to achieve some level of performance specified by (2), and that the performance may vary depending on how the identified parameter is set. We may determine the optimal value of the parameter relative to particular payoff A as follows. First, we model the protocol as an action system, using angelic choice to set the identified parameter. Next we compute the the optimal cost relative to A by evaluating (2) for the various definitions, and the optimal value for the identified parameter may then be determined by, for example, using the "policy iteration method" [25].These ideas are illustrated in our case study below.

Thus far our comments only apply when A is a bounded expectation. In our case study however we are obliged to consider unbounded expectations A, to express expected times. In the next section we consider a variation of the game which applies to unbounded payoffs as well.

2.5 Discounted games with unbounded payoffs

In some cases early payoffs are valued more highly over later ones, and in this case a *discount* factor is usually applied at each iteration [4]. In wireless networks for example degrading battery lives mean that the results from later payoffs become insignificant.

Definition 3. *Given an action system System, the discounted game with discount factor $p \in (0,1)$, is defined to be*[7]

$$\diamond_p A \quad \hat{=} \quad (\mu X \cdot A \sqcup p \times Wp.System.X) \ . \tag{5}$$

Discounted games enjoy a number of nice properties [17], and in particular they remain well-defined even when the payoff function is unbounded. We say that A *grows linearly* relative to *System* provided that there is a fixed constant $K \geq 0$ such that for all pairs of states (s, s') related via *System*, $|A.s - A.s'| \leq K$. In the remainder of this section we shall show that, provided A grows linearly, the game is well-defined and the players still have optimal memoryless strategies. The proofs of all the results may be found in the appendix.[8]

Lemma 1. *Suppose that A grows linearly with respect to* System *in the expression $\diamond_p A$ set out at (5), and is bounded below. The least fixed point on the right hand side at (5) is well-defined (i.e. non-infinitary).*

It turns out that the optimal strategies of the players may be computed from the solution of the game at (5). We denote the optimal strategies of a single step of *System* relative to an immediate post-expectation E as follows. We define $System_E$ to be *System* with each nondeterministic choice (both angelic and demonic) is replaced by a Boolean choice such that

$$Wp.System.E \quad = \quad Wp.System_E.E \ .$$

(The revised *Guesser* given by (4) defines such a Boolean choice for expectation (2) and (3).) Note that such Boolean choices always exist if the nondeterminism has only a finite range of possibilities [27, 16]. With that notation we may now state a corollary of Lem. 1, that both players in (5) have optimal strategies.

Lemma 2. *The players of the game $\diamond_p A$ both have optimal strategies whenever A grows linearly with respect to System, and is bounded below.*

Computing the optimal strategies can be done using the well-known "value (policy) iteration method" [25]. The idea is that the optimal strategies may be improved at each iteration by solving the equations

$$A \sqcup Wp.System_{E_n}.F^n.A \quad = \quad F^{n+1}.A \ ,$$

[7] We shall also consider the slightly less general $(\mu X \cdot A \lhd G \rhd p \times Wp.System.X)$, where G is any predicate. We note that all the lemmas are still valid for this variation.

[8] Found at http://www.comp.mq.edu.au/∼anabel/ICTAC06.pdf.

where $F \mathrel{\hat{=}} (\lambda X \cdot A \sqcup Wp.System.X)$. Here $System_{E_0}$, $System_{E_1} \ldots$ define a sequence of strategies, which converges in finite state spaces [25], so that there is some $N > 0$ such that for all $n \geq N$ we have the equality $System_{E_n} = System_{E_N}$, and that $System_{E_N}$ is optimal. It turns out that the same idea works even for unbounded expectations, as the next lemma shows.

Lemma 3. *The value (policy) iteration method is a valid method for computing the optimal strategies under the assumptions of Lem. 2.*

In this section we have indicated how angelic nondeterminism may be used to optimise system design relative to a payoff function, even when that function is unbounded. In the next section we illustrate these ideas in an application for wireless communication.

3 A sleep/awake protocol for wireless networks

A promising approach to conserving battery power in wireless networks is to allow nodes periodically to enter a "low power mode" at times when the network traffic is low. During a low power phase, nodes do not actively listen for network activity, thus power is conserved that would otherwise have been wasted on so-called "idle listening" [28]. The disadvantge of this approach however is that the throughput is (almost certainly) decreased since messages may be blocked temporarily en route as they are forced to wait until sleeping nodes wake up.

In this section we describe a protocol designed to manage the scheduling of low/high power modes in a wireless network, with the goal of analysing the sleeping time so that the optimal sleep schedule may be determined.

Our study is based on a protocol suggested by Wei Ye *et al.* [28] and is intended to reduce the effects of message delay by arranging neighbouring nodes to wake up at approximately the same time. Nodes do this by broadcasting a *synch packet* just before they go to sleep announcing when they will next wake. Any neighbour in receipt of the synch packet then knows when it should also be awake and listening for traffic.

The main events in the sleep/awake protocol are set out at Fig. 4; the general scheme is as follows. After waking from a low power mode, a node listens for a while, sending and receiving any data as necessary. It then broadcasts and/or recieves a *synch* packet, before setting its internal timer to its next scheduled waking time; it then goes to sleep.

Although the overall scheme of the protocol is fixed, there are still some opportunities for fine-tuning. For example the synch packet informs a node only *approximately* (rather than exactly) when its neighbour will next wake, and therefore it may be better for the neighbour to become active at some slightly different time, one which takes into account the uncertainties caused by natural delays involved in wireless communication and "clock drift", both of which diminish the reliability of the information received in the *synch* packet. Here *clock drift* is a common phenomenon in distributed systems that cannot rely on a global clock. Wireless nodes, for instance, use their own internal clocks, and

after some time (the order of tens of seconds [28]) nodes' internal timers will have drifted relative to others' in the network. The problem may be corrected by periodic time synchronisation, or (in some applications) the use of distributed global clocking schemes.

- *Wake*: If local timer exceeds the wake up deadline, then the mode changes from low to high power;
- *Listen*: The node listens for a signal in the network;
- *Send/Receive(data)*: The node receives any data, and replies with an acknowledgement if necessary;
- *Send/Receive(synch)*: The node either receives or sends a "synch packet" announcing a sleep schedule.
- *Choose*(w_r) : If a synch packet has been received from a neighbour, it sets its own wake-up deadline by setting parameter w_r;
- *Sleep*: It reinitialises its local timer and changes the mode from awake to sleeping.

Fig. 4. The major events in the sleep/awake protocol

Our formal model is constructed from a number of *modules*, each one consisting of a set of labelled *actions* describing the underlying state changes incurred after each event. In Fig. 5 we set out an action system modelling the main events at Fig. 4 for a *Receiver* node. The $Sender_i$ nodes are similar except that we use variables s_i, t_i and w_i instead of (respectively) r, t_r and w_r, and the commands labelled † and ‡ are replaced as follows. We replace the †'s respectively by

$$\begin{array}{l} send!i : \ (s_i = listen) \rightarrow \ s_i := ack_i \\ ack_i : \ (s_i = ack) \rightarrow \ s_i = listen \end{array}$$

and ‡ by

$$tick : \ (s_i = listen \wedge (r = sleep)) \rightarrow \ \mathsf{skip} \ .$$

Thus the exchange of data is indicated by the senders and receivers synchronising on $send_i$ and ack_i events. Additional delays on throughput can be due to messages colliding (event *clash* handled by module *Channel* set out in Fig. 6), or the time spent waiting for a node's neighbour to wake. The latter may routinely happen because of clock drift, and here we model that behaviour using a probabilistic increment for the internal timer variable. In the case of message collisions, we note that the *Channel* may be blocked or clear, but may only be blocked for a finite time (with probability 1). In a more detailed model of the message-passing protocol [14] collisions are resolved by a probabilistic backoff — here we only model the overall effect of that backoff making *clash* a probabilistic event.

The whole system is defined by the parallel composition of the senders, the receivers, the channel and a module *time* which provides a notion of the real passage of time so that statistics on the time spent in the sleep state and the

delays in throughput may be gathered. In this small example, we only have two senders and a receiver.

$$Network \;\;\hat{=}\;\; Sender_1 \;||\; Sender_2 \;||\; Channel \;||\; Receiver \;||\; RealTime$$

$$Receiver \mathrel{\hat{=}} \left(\begin{array}{ll}
\textbf{var}\ r : \{listen, sleep, ack_i\}, t_r, w_r : \mathbb{N} & \\
\textbf{initially}\ r := listen; & \\
[\!]_i\, send?i : \ (r = listen) \rightarrow \ r := ack_i & \dagger \\
[\!]_i\, ack_i : \ (r = ack_i) \rightarrow \ r := listen & \dagger \\
clash : \ (r = listen) \rightarrow \mathsf{skip} & \\
 & \\
tick : \ (r = sleep \wedge t_r < w_r) \rightarrow \ t_r := t_r + 1 \ {}_d\!\oplus \mathsf{skip} & \\
tick : \ (r = sleep \wedge t_r \geq w_r) \rightarrow \ r := listen & \\
tick : \ (r = listen \wedge (\forall i \cdot s_i = sleep)) \rightarrow \ \mathsf{skip} & \ddagger \\
synch : \ (r = listen) \rightarrow \ r := sleep \ ; \ t_r := 0 \ ; \ Choose(w_r) &
\end{array}\right)$$

The function $Choose(w_r) \mathrel{\hat{=}} (w_r := 1) \sqcup \ldots \sqcup (w_r := T)$, for some fixed T; the probabilistic assignment to t_r indicates some clock drift with rate relative to d.

Fig. 5. Specification of a Receiver node.

Our aim is to investigate how to choose the optimal sleeping schedule w_r, namely to resolve the angelic nondeterminism in $Choose(w_r)$, where w_r is chosen from some finite range of values on the firing of action *synch*. That is the topic of the next section.

3.1 Optimal behaviour of the sleep/awake protocol

In this section we formalise a game in order to optimise $Choose(w_r)$ in Fig. 5 relative to an appropriate payoff function. The idea here is to optimise the payoff for saving battery power in the context of message delivery; thus our payoff function optimises time spent in the sleep state offset by the cost of possibly delaying messages. To estimate the latter cost, we assign a value to the messages according to the length of time they took to be delivered. For example if the contents of the messages quickly become out of date, then the longer they take to be delivered the less valuable they are. This is a common concern in the gathering of environmental data such as temperatures or moisture levels. The only contraint we need for our game to be well-defined is that the payoff should be grow linearly relative to *Network*.

An example of such a function is set out at Fig. 7, where *totalSleep* and *totalDelay* are statistics defining respectively the overall time spent in the low power mode, and the total time it takes to deliver messages. [9] We can collect

[9] We include the termination condition in Fig. 7 rather than use $\sqcup$ since we are only intending to optimise the selection of w_r.

$$Time \mathrel{\hat{=}} \left(\begin{array}{l} \textbf{var}\ t \colon \mathbb{N} \\ \textbf{initially}\ t \colon = 0 \\ [\![_{(evt \colon \alpha)}\ evt : (t \geq 0) \rightarrow t \colon = t + 1 \end{array} \right) \qquad Channel \mathrel{\hat{=}} \left(\begin{array}{l} \textbf{var}\ c \colon \{block, clear\} \\ \textbf{initially}\ c \colon = block\ ;\ t \colon = 0 \\ clash : (c = block) \rightarrow \mathsf{skip}\ {}_q\!\oplus c \colon = clear \end{array} \right)$$

The module *Time* is used to allow us to collect statistics for the performance analysis. Here α is the union of all the timing events defined by the *sender*s and the *receiver*s; specifically $\alpha \mathrel{\hat{=}} \{tick, synch, clash\}$. The event *clash* signifies a collision between messages.

Fig. 6. Modelling the behaviour of real time and the channel.

$$(\mu X \ \cdot\ (a \times totalSleep - b \times totalDelay) \ \triangleleft G \triangleright\ (p \times Wp.System\ ;\ X)) \qquad (6)$$

totalSleep is the total time that the receiver has spent in the sleeping mode; *totalDelay* is the total message delay, and G is the termination condition, that all messages have been delivered. Constants a and b are determined by the application.

Fig. 7. The game for optimising the cost of sleeping offset by the throughput.

those statistics from the formal model of *Network* as follows. To define the overall time spent in the *Receiver*'s low power state we augment the definition at Fig. 5 with variables *SleepTime* and *totalSleep*. Thus on waking *SleepTime* is set to the current value of the global time variable t, and just before going to sleep again *totalSleep* is set to $totalSleep + t - SleepTime$. We measure the total additional time it takes to send a message, and store it in a variable called *totalDelay* by similarly augmenting the $Sender_i$'s.

When scalars a and b are set to 1, the problem does not articulate any constraint on message delay (and thus the optimal sleep schedule would be never to wake up!) A better variation is to include scalars a and b to express the relative importance between saving power and tolerance of message delay, depending on the application [28]. Thus a high value of b relative to a means that message delay costs more than power savings; for example if $a/b = 1/2$ then a delay of 1 second costs twice as much as the power saved by a low power interval of 1 second.[10]

Finally from Fig. 6 we see that time is incremented linearly with each execution of *Network*, and so (6) also only grows linearly with respect to *Network*, thus Lem. 1 and Lem. 2 implies that (6) is well-defined and the player *Max* choosing the schedule must have an optimal strategy.

3.2 Experimental results

We used the PRISM model checker [23] to model *Network*, and to compute the various probability distributions over *totalSleep* and *totalDelay*. From those re-

[10] Note that the apparent unboundness (below) of (6) is not a problem — see *Lem. 6* in the appendix.

sults we computed the expected optimal productive sleeping time. We assumed that the *Sender*s always announced a fixed sleep schedule of 2 time steps, and based on that we computed *Receiver*'s optimal assignment to w_r. In our experiments we used parameters $a, b := 1$ and also $a := 1$ and $b := 2$. As expected, the results show that in the former case, it is better for the *Receiver* to set its wait deadline greater than 2 time steps, whereas in the latter case, where throughput is more essential (comparatively) than saving power it is better for *Receiver* to wake up after 2.

4 Conclusions

We have proposed a stochastic parity game to investigate the tradeoff between power savings and message throughput in a wireless network. Our formalisation allows the use of standard value and policy iteration algorithms to compute the optimal sleep schedule, moreover having access to model checking allows the overall expected delays to be computed once the optimal sleep schedule has been determined.

An alternative approach would be to formalise the problem as a standard optimisation problem *viz.* to optimise the time spent in the low power mode constrained by an upper bound on throughput. There are several practical and theoretical drawbacks to this however. The first is that both the objective function, and the constraints in such a formalisation must be determined by expressions of the form (6), and need themselves to be computed using (for example) non-trivial probabilistic model-checking techniques. Furthermore there is an additional dependency between the contraint and the objective function due to the inherent nondeterminism, and this cannot be expressed straightforwardly. The use of our proposed game addresses the latter problem, as there is no separate constraint expression; moreover as there is only a single expression of the form (6) involved in the problem fomalisation its solution can be obtained more efficiently. A drawback is that it might be difficult to determine appropriate parameters a and b.

We are not the first to give a game interpretation of Action Systems [3]. Others have explored similar two-player "Markov Games" [13], and de Alfaro *et al.* have studied the use of discounting in systems theory [7]. Formal models of power management have be investigated in other contexts [4, 22].

References

1. A. Aziz, V. Singhal, F. Balarinand R.K. Brayton, and A.L. Sangiovanni-Vincentelli. It usually works: The temporal logic of stochastic systems. In *Computer-Aided Verification, 7th Intl. Workshop*, volume 939 of *LNCS*, pages 155–65. Springer Verlag, 1995.
2. R.-J.R. Back. A calculus of refinements for program derivations. *Acta Informatica*, 25:593–624, 1988.
3. R.-J.R. Back and J. von Wright. *Refinement Calculus: A Systematic Introduction*. Springer Verlag, 1998.

4. Luca Benini, Alessandro Bogliolo, Giuseppe A. Paleologo, and Giovanni De Micheli. Policy optimization for dynamic power management. *IEEE Transactions of Computer-Aided Design of Integrated Systems and Circuits*, 18(6):813–834, 1999.
5. M.J. Butler and C.C. Morgan. Action systems, unbounded nondeterminism and infinite traces. *Formal Aspects of Computing*, 7(1):37–53, 1995.
6. K. Chatterjee, M. Jurdzinski, and T. Henzinger. Quantitative stochastic parity games.
7. Luca de Alfaro, Thomas A. Henzinger, and Rupak Majumdar. Discounting the future in systems theory. In *Proceedings of the 30th International Colloquium on Automata, Languages, and Programming (ICALP)*, LNCS, pages 1022–1037, 2003.
8. S. Hallerstede and M. Butler. Performance analysis of probabilistic action systems. 2005.
9. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1985.
10. Joe Hurd, A.K. McIver, and C.C. Morgan. Probabilistic guarded commands mechanised in HOL. To appear in *Proc. QAPL '04 (ETAPS)*, 2004.
11. G. Jones. *Programming In occam*. Prentice Hall International, 1988.
12. D. Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–54, 1983.
13. M.G. Lagoudakis and R. Parr. Value function approximation in zero-sum markov games. 2002.
14. A.K. McIver. Quantitaive refinement and model checking for the analysis of probabilistic systems. Accepted for FM 2006.
15. A.K McIver and C.C. Morgan. Games, probability and the quantitative μ-calculus qMu. In *Proc. LPAR*, volume 2514 of *LNAI*, pages 292–310. Springer Verlag, 2002. Revised and expanded at [16].
16. A.K. McIver and C.C. Morgan. Results on the quantitative μ-calculus $qM\mu$. To appear in *ACM TOCL*, 2004.
17. AK McIver and CC Morgan. Memoryless strategies for stochastic games via domain theory. *ENTCS*, 130:23–37, 2005.
18. Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Technical Monographs in Computer Science. Springer Verlag, New York, 2004.
19. C.C. Morgan. Of probabilistic Wp and CSP. *25 years of CSP*.
20. C.C. Morgan. *Programming from Specifications*. Prentice-Hall, 1994.
21. C.C. Morgan and A.K. McIver. *pGCL*: Formal reasoning for random algorithms. *South African Computer Journal*, 22, March 1999.
22. G. Norman, D. Parker, M. Kwiatkowska, S. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. 2005.
23. PRISM. Probabilistic symbolic model checker. `www.cs.bham.ac.uk/~dxp/prism`.
24. Kaisa Sere and Elena Troubitsyna. Probabilities in action systems. In *Proc. of the 8th Nordic Workshop on Programming Theory*, 1996.
25. RS Sutton and AG Barto. *Reinforcement Learning*. MIT Press, 1998.
26. A. Tarski. A lattice-theoretic fixpoint theorem and its applications. *Pacific Journal of Mathematics*, 5:285–309, 1955.
27. J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, second edition, 1947.
28. 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.