MODELLING TOOLS AND TECHNIQUES FOR THE PERFORMANCE ANALYSIS OF WIRELESS PROTOCOLS

ASHOK ARGENT-KATWALA, JEREMY T. BRADLEY*, NIL GEISWEILLER, STEPHEN T. GILMORE†, AND NIGEL THOMAS‡

Abstract. In this chapter, we show how a stochastic process algebra, PEPA, can be used to describe performance models of wireless network protocols. We highlight the existing body of tools and modelling techniques to demonstrate the different types of performance analysis that are possible on the underlying Markov model. We present a case study of the 802.11 family of protocols, which we analyse to obtain quality of service measurements in the form of passage time quantiles.

1. Introduction. Wireless communications protocols need to transport data reliably and securely while meeting QoS criteria related to throughput or latency and simultaneously satisfying service-level criteria which mandate that a given high percentage of messages must be delivered in a specified time. Well-designed protocols will operate reliably in harsh environments with intermittent connectivity of low capacity bandwidth. It is a steep engineering challenge to be able to deliver reliable data services in the wireless environment in a timely fashion and with good availability. For this reason, we view quantitative analysis techniques which consider measurable criteria such as throughput and response time as being as important as qualitative ones such as freedom from deadlock. The quantitative analysis of computer protocols through construction and solution of descriptive high-level models is a hugely profitable activity: brief analysis of a model can provide as much insight as hours of simulation and measurement [1].

Building performance models of realistic real-world communications protocols is an activity which requires careful attention to detail in order to model correctly the intended behaviour of the system. Proceeding with care during this part of the modelling process is a wise investment of effort. If the initial performance model contains errors then all of the computational expense incurred in solving the model and all of the intellectual effort invested in the analysis and interpretation of the results obtained would at best be wasted. In general interpreting a model with errors could lead to making flawed implementation or engineering decisions based on erroneous conclusions made from erroneous results. Formal modelling languages such as process algebras are helpful here because they allow unambiguous models to be developed which can be checked by software tools. These tools can be used to find errors which a human might overlook or to compute quantitative results which deliver insights into the dynamics of the functioning protocol.

Jane Hillston's Performance Evaluation Process Algebra (PEPA) [2] is an expressive formal language for modelling computer and communication systems. PEPA models are constructed by the composition of components which perform individual activities

*Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2BZ, United Kingdom. {ashok,jb}@doc.ic.ac.uk

†Laboratory for Foundations of Computer Science, The University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom. ngeiswei@informatics.ed.ac.uk, Stephen.Gilmore@ed.ac.uk

‡School of Computing Science, University of Newcastle-upon-Tyne, Newcastle-upon-Tyne NE1 7RU, United Kingdom. Nigel.Thomas@ncl.ac.uk

or cooperate on shared ones. To each activity is attached an estimate of the rate at which it may be performed. Using such a model, a protocol designer can determine whether a candidate design meets both the behavioural and the temporal requirements demanded of it. That is: the protocol may be behaviourally correct, but can it be executed quickly enough to complete the message exchange within a specified time bound, with a given probability of success?

The rest of the chapter has structure as set out below. Sect. 2 describes the basic modelling constructs behind the PEPA formalism. Sect. 3 shows how the Markovian system underlying the PEPA model can be analysed to produce steady-state, passage-time and transient quantities. Sect. 4 discusses different modelling constructs and issues that are encountered while capturing wireless protocol; i.e. queueing subsystems for buffers, phase-type distributions for timeouts and stochastic parameter estimation for accurate model parameterisation. Sect. 5 introduces a selection of popular tools for producing quantitative analysis of PEPA models; e.g. PEPA Workbench, ipc, MEERCAT and EMPEPA. Finally, Sect. 6 presents a PEPA model and analysis of a version of the IEEE 802.11 protocol from Sridhar and Ciobanu [3].

2. The PEPA Process Algebra. PEPA [4] as a performance modelling formalism has been used to study a wide variety of systems: multimedia applications [5], mobile phone usage [6], GRID scheduling [7], production cell efficiency [8] and web-server clusters [9] amongst others. The definitive reference for the language is [4].

As in all process algebras, systems are represented in PEPA as the composition of *components* which undertake *actions*. In PEPA the actions are assumed to have a duration, or delay. Thus the expression $(\alpha, r).P$ denotes a component which can undertake an α action at rate r to evolve into a component P. Here $\alpha \in \mathcal{A}$ where $\mathcal{A}$ is the set of action types and $P \in \mathcal{C}$ where $\mathcal{C}$ is the set of component types. The rate r represents the parameter of an exponential distribution, and the duration is assumed to be a random variable. The mean, or expected value, of an exponential distribution with rate r is $1/r$. Thus $(\alpha, r).P$ is a component where the average duration of the α activity is $1/r$.

PEPA has a small set of combinators, allowing system descriptions to be built up as the concurrent execution and interaction of simple sequential components. The syntax of the type of PEPA model considered in this paper may be formally specified using the following grammar:

$$S ::= (\alpha, r).S \mid S + S \mid C_S$$
$$P ::= P \underset{L}{\bowtie} P \mid P/L \mid C$$

where S denotes a *sequential component* and P denotes a *model component* which executes in parallel. C stands for a constant which denotes either a sequential component or a model component as introduced by a definition. C_S stands for constants which denote sequential components. The effect of this syntactic separation between these types of constants is to constrain legal PEPA components to be cooperations of sequential processes.

More information on PEPA can be found in [4]. The structured operational semantics are shown in Fig. 2.1. A brief discussion of the basic PEPA operators is given below:

Prefix The basic mechanism for describing the behaviour of a system with a PEPA

Prefix

$$\frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Competitive Choice

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E'} \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} F'}$$

Cooperation

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{S}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{S}{\bowtie} F} \ (\alpha \notin S) \qquad \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{S}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{S}{\bowtie} F'} \ (\alpha \notin S)$$

$$\frac{E \xrightarrow{(\alpha,r_1)} E' \quad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{S}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{S}{\bowtie} F'} \ (\alpha \in S)$$

$$\text{where } R = \frac{r_1}{r_a(E)} \frac{r_2}{r_a(F)} \min(r_a(E), r_a(F))$$

Hiding

$$\frac{E \xrightarrow{(\alpha,r)} E'}{E \backslash L \xrightarrow{(\alpha,r)} E' \backslash L} \ (\alpha \notin L) \qquad \frac{E \xrightarrow{(\alpha,r)} E'}{E \backslash L \xrightarrow{(\tau,r)} E' \backslash L} \ (\alpha \in L)$$

Constant

$$\frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'} \ (A \stackrel{def}{=} E)$$

Fig. 2.1. PEPA Structured Operational Semantics

model is to give a component a designated first action using the prefix combinator, denoted by a full stop. As explained above, $(\alpha, r).P$ carries out an α action with rate r, and it subsequently behaves as P.

Choice The component $P + Q$ represents a system which may behave either as P or as Q. The activities of both P and Q are enabled. The first activity to complete distinguishes one of them: the other is discarded. The system will behave as the derivative resulting from the evolution of the chosen component.

Constant It is convenient to be able to assign names to patterns of behaviour associated with components. Constants are components whose meaning is given by a defining equation. The notation for this is $X \stackrel{def}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for exam-

ple, $X \stackrel{def}{=} (\alpha, r).X$ performs α at rate r forever.

Hiding The possibility to abstract away some aspects of a component's behaviour is provided by the hiding operator, denoted P/L. Here, the set L identifies those activities which are to be considered internal or private to the component and which will appear as the distinguished unknown type τ.

Cooperation We write $P \underset{L}{\bowtie} Q$ to denote cooperation between P and Q over L. The set which is used as the subscript to the cooperation symbol, the *cooperation set* L, determines those activities on which the components are forced to synchronise. The set L cannot contain the unknown type τ. For action types not in L, the components proceed independently and concurrently with their enabled activities. We write $P \parallel Q$ as an abbreviation for $P \underset{L}{\bowtie} Q$ when L is empty.

In process cooperation, if a component enables an activity whose action type is in the cooperation set it will not be able to proceed with that activity until the other component also enables an activity of that type. The two components then proceed together to complete the *shared activity* with an appropriate rate. The capacity of a component P to perform an action α is denoted $r_\alpha(P)$, and is termed the *apparent rate*. PEPA respects the definition of *bounded capacity*: that is, a component cannot be made to perform an activity faster by cooperation, so the apparent rate of a shared activity in a cooperation is the minimum of the apparent rates of the activity in the cooperating components.

In some cases, when a shared activity is known to be completely dependent only on one component in the cooperation, then the other component will be made *passive* with respect to that activity. This means that the rate of the activity is left unspecified (denoted $\top$) and is determined upon cooperation, by the rate of the activity in the other component. All passive actions must be synchronised in the final model.

The structured operational semantics give a precise interpretation of all possible behaviours of any PEPA model, which can be represented as a *derivation graph* or *labelled multi-transition system*. In the derivation graph each possible global state of the model is represented by a node while the arcs record the (possibly multiple) transitions (*(action type, rate)* pairs) between states. This graph can then be interpreted as the state transition diagram of a continuous time Markov chain (CTMC) and subjected to the usual steady state and transient analysis applied to CTMCs. Here a global state or *derivative* is a syntactic form which can be reached by the evolution of the original PEPA expression according to the semantic rules.

2.1. PEPAinf. PEPAinf is an extension to PEPA to allow components with regular, infinite state spaces. By providing such an extension, we benefit from extra analysis possibilities for PEPA models by making use of potential product form solutions.

Briefly the extension to PEPA is described below. Component names may be augmented with a list of subscripts which range over the integers. This allows for unbounded, but regular state spaces. We also force finite branching in or out of any state.

The intention with PEPAinf is to formalise definitions like:

$$P_0 \stackrel{def}{=} (a, s_1).P_1$$

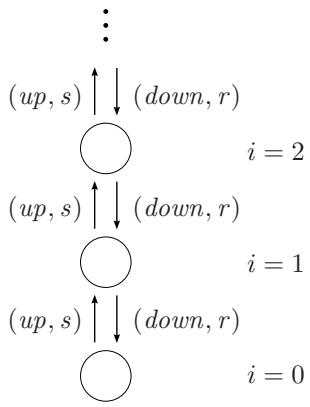

Fig. 2.2. A simple birth–death process

$$P_i \stackrel{def}{=} (a, s_1).P_{i+1} + (d, s_2).P_{i-1} \qquad \forall i > 0$$

P	::=	Name | Name(VarList)
VarList	::=	Var | VarList,Var
Var	::=	Varname[Arg][Guard]
Arg	::=	+Num | −Num
Guard	::=	>Num | <Num

For example, the simple birth–death process depicted in Fig. 2.2 is defined in PEPAinf as:

$$P_{i=0} \stackrel{def}{=} (up, s).P_{i+1}$$
$$P_{i>0} \stackrel{def}{=} (down, r).P_{i-1} + (up, s).P_{i+1}$$

So, for example, we can specify a tandem pair of $M/M/1$ queues as:

$$P_{i=0} \stackrel{def}{=} (arr, r).P_{i+1}$$
$$P_{i>0} \stackrel{def}{=} (arr, r).P_{i+1} + (int, s).P_{i-1}$$
$$Q_{i=0} \stackrel{def}{=} (int, \top).Q_{i+1}$$
$$Q_{i>0} \stackrel{def}{=} (int, \top).Q_{i+1} + (out, s').Q_{i-1}$$
$$Sys \stackrel{def}{=} P_{i=0} \underset{\{int\}}{\bowtie} Q_{i=0}$$

Any PEPA process is a valid PEPAinf process, as our only change is to allow some new, structured names. We retain many of PEPA's restrictions – in particular retaining a finite alphabet of action-types, finite branching and a fixed, finite number of processes in the system – while covering some new, useful processes, particularly with respect to product-form results.

3. Quantitative Analysis. In this section, we describe the underlying mathematical model for PEPA, namely the continuous-time Markov chain. Further, the analysis techniques steady-state, passage time and transient analysis are introduced. These are the core analysis techniques for PEPA models and will be used to provide performance results for the wireless protocol case study in Sect. 6.

3.1. Stochastic Processes. At the lowest level, the performance modelling of a system can be accomplished by identifying all possible states that the system can enter and describing the ways in which the system can move between those states. This is termed the *state-transition* level behaviour of the model, and the changes in state as time progresses describe a *stochastic process*. In this section, we focus on those stochastic processes which belong to the class known as *Markov processes*, specifically continuous-time Markov chains (CTMCs).

Consider a random variable X which takes on different values at different times t. The sequence of random variables $X(t)$ is said to be a stochastic process. The different values which members of the sequence $X(t)$ can take (also referred to as *states*) all belong to the same set known as the *state space* of $X(t)$.

A stochastic process can therefore be classified by the nature of its state space and of its time parameter. If the values in the state space of $X(t)$ are finite or countably infinite, then the stochastic process is said to have a *discrete state space* (and may also be referred to as a *chain*). Otherwise, the state space is said to be *continuous*. Similarly, if the times at which $X(t)$ is observed are also countable, the process is said to be a *discrete time* process. Otherwise, the process is said to be a *continuous time* process. In this chapter, all stochastic processes considered have discrete state spaces, and we focus mainly on those which evolve in continuous time.

A *Markov* process is a stochastic process in which the *Markov property* holds. Given that $X(t) = x_t$ indicates that the state of the process $X(t)$ at time t is x_t, this property stipulates that:

$$\begin{aligned}\mathbb{P}\big(X(t) = x \mid X(t_n) = x_n, X(t_{n-1}) = x_{n-1}, \ldots, X(t_0) = x_0\big) \\ = \mathbb{P}\big(X(t) = x \mid X(t_n) = x_n\big) \\ : \text{for } t > t_n > t_{n-1} > \ldots > t_0 \end{aligned} \quad (3.1)$$

That is, the future evolution of the system depends only on the current state and not on any prior states.

A Markov process is said to be *homogeneous* if it is invariant to shifts in time, that is:

$$\mathbb{P}\big(X(t+s) = x \mid X(t_n + s) = x_n\big) = \mathbb{P}\big(X(t) = x \mid X(t_n) = x_n\big)$$

3.1.1. Continuous-Time Markov Chains. There exists a family of Markov processes with discrete state spaces but whose transitions can occur at arbitrary points in time; we call these continuous-time Markov chains (CTMCs). An homogeneous N-state $\{1, 2, \ldots, N\}$ CTMC has state at time t denoted $X(t)$. Its evolution is described by an $N \times N$ generator matrix Q, where q_{ij} is the infinitesimal rate of moving from state i to state j $(i \neq j)$, and $q_{ii} = -\sum_{i \neq j} q_{ij}$.

The Markov property imposes the restriction on the distribution of the sojourn times in states in a CTMC that they must be *memoryless* – the future evolution of the system therefore does not depend on the evolution of the system up until the current state, nor does it depend on how long has already been spent in the current state. This means that the sojourn time ν in any state must satisfy:

$$\mathbb{P}(\nu \geq s + t \mid \nu \geq t) = \mathbb{P}(\nu \geq s) \quad (3.2)$$

A consequence of Eqn. (3.2) is that all sojourn times in a CTMC must be exponentially distributed.

The rate out of state i, and therefore the parameter of the sojourn time distribution, is μ_i and is equal to the sum of all rates out of state i, that is $\mu_i = -q_{ii}$. This means that the density function of the sojourn time in state i is $f_i(t) = \mu_i\, e^{-\mu_i t}$ and the average sojourn time in state i is $1/\mu_i$.

3.2. Steady-state. We now define the steady-state distribution for a CTMC. We denote the set of steady-state probabilities as $\{\pi_j\}$.

In a CTMC which has all states recurrent non-null and which is time-homogeneous, the *limiting* or *steady-state probability distribution* $\{\pi_j\}$ is given by:

$$\pi_j = \lim_{t\to\infty} \mathbb{P}(X(t) = j \mid X(0) = i)$$

For an finite, irreducible and homogeneous CTMC, the steady-state probabilities $\{\pi_j\}$ always exist and are independent of the initial state distribution. They are uniquely given by the solution of the equations:

$$-q_{jj}\pi_j + \sum_{k\neq j} q_{kj}\pi_k = 0 \quad \text{subject to} \quad \sum_i \pi_i = 1$$

Again, this can be expressed in matrix vector form (in terms of the vector $\vec{\pi}$ with elements $\{\pi_1, \pi_2, \ldots, \pi_N\}$ and the matrix Q defined above) as:

$$\vec{\pi}Q = \vec{0}$$

A CTMC also has an embedded discrete-time Markov chain (EMC) which describes the behaviour of the chain at state-transition instants, that is to say the probability that the next state is j given that the current state is i. The EMC of a CTMC has a one-step $N \times N$ transition matrix P where $p_{ij} = -q_{ij}/q_{ii}$ for $i \neq j$ and $p_{ij} = 0$ for $i = j$.

3.3. Passage-time. Consider a finite, irreducible CTMC with N states and generator matrix Q from Sect. 3.1.1. As $X(t)$ denotes the states of the CTMC at time $t \geq 0$ and $N(t)$ denotes the number of state transitions which have occurred by time t, the first passage time from a single source marking i into a non-empty set of target markings $\mathcal{J}$ is:

$$P_{i\mathcal{J}}(t) = \inf\{u > 0 : X(t+u) \in \mathcal{J}, N(t+u) > N(t), X(t) = i\}$$

When the CTMC is stationary and time-homogeneous this quantity is independent of t:

$$P_{i\mathcal{J}} = \inf\{u > 0 : X(u) \in \mathcal{J}, N(u) > 0, X(0) = i\} \tag{3.3}$$

That is, the first time the system enters a state in the set of target states $\mathcal{J}$, given that the system began in the source state i and at least one state transition has occurred. $P_{i\mathcal{J}}$ is a random variable with probability density function $f_{i\mathcal{J}}(t)$ such that:

$$\mathbb{P}(a < P_{i\mathcal{J}} < b) = \int_a^b f_{i\mathcal{J}}(t)\,\mathrm{d}t \quad : 0 \leq a < b$$

In order to determine $f_{i\mathcal{J}}(t)$ it is necessary to convolve the state holding-time density functions over all possible paths (including cycles) from state i to all of the states in $\mathcal{J}$.

The calculation of the convolution of two functions in t-space can be more easily accomplished by multiplying their Laplace transforms together in complex s-space and inverting the result. The calculation of $f_{i\mathcal{J}}(t)$ is therefore achieved by calculating the Laplace transform of the convolution of the state holding times over all paths between i and $\mathcal{J}$ and then numerically inverting this Laplace transform.

In a CTMC all state sojourn times are exponentially distributed, so the density function of the sojourn time in state i is $f_i(t) = \mu_i e^{-\mu_i t}$, where $\mu_i = -q_{ii}$ for $1 \leq i \leq N$ as defined in Sect. 3.1.1. The Laplace transform of an exponential, $f(t)$, density function with rate parameter λ is:

$$\mathcal{L}(f(t)) = \frac{\lambda}{s+\lambda}$$

Denoting the Laplace transform of the density function $f_{i\mathcal{J}}(t)$ of the passage time random variable $P_{i\mathcal{J}}$ as $L_{i\mathcal{J}}(s)$, we proceed by means of a first-step analysis. That is, to calculate the first passage time from state i into the set of target states $\mathcal{J}$, we consider moving from state i to its set of direct successor states $\mathcal{K}$ and thence from states in $\mathcal{K}$ to states in $\mathcal{J}$. This can be expressed as the following system of linear equations:

$$L_{i\mathcal{J}}(s) = \sum_{k \notin \mathcal{J}} p_{ik} \left(\frac{-q_{ii}}{s - q_{ii}} \right) L_{k\mathcal{J}}(s) + \sum_{k \in \mathcal{J}} p_{ik} \left(\frac{-q_{ii}}{s - q_{ii}} \right) \tag{3.4}$$

The first term (i.e. the summation over non-target states $k \notin \mathcal{J}$) convolves the sojourn time density in state i with the density of the time taken for the system to evolve from state k into a target state in the set $\mathcal{J}$, weighted by the probability that the system transits from state i to state k. The second term (i.e. the summation over target states $k \in \mathcal{J}$) simply reflects the sojourn time density in state i weighted by the probability that a transition from state i into a target state k occurs.

Given that $p_{ij} = -q_{ij}/q_{ii}$ in the context of a CTMC (cf. Sect. 3.1.1), Eqn. (3.4) can be rewritten as:

$$L_{i\mathcal{J}}(s) = \sum_{k \notin \mathcal{J}} \frac{q_{ik}}{s - q_{ii}} L_{k\mathcal{J}}(s) + \sum_{k \in \mathcal{J}} \frac{q_{ik}}{s - q_{ii}} \tag{3.5}$$

This set of linear equations can be expressed in matrix–vector form. For example, when $\mathcal{J} = \{1\}$ we have:

$$\begin{pmatrix} s - q_{11} & -q_{12} & \cdots & -q_{1n} \\ 0 & s - q_{22} & \cdots & -q_{2n} \\ 0 & -q_{32} & \cdots & -q_{3n} \\ 0 & \vdots & \ddots & \vdots \\ 0 & -q_{n2} & \cdots & s - q_{nn} \end{pmatrix} \begin{pmatrix} L_{1\mathcal{J}}(s) \\ L_{2\mathcal{J}}(s) \\ L_{3\mathcal{J}}(s) \\ \vdots \\ L_{n\mathcal{J}}(s) \end{pmatrix} = \begin{pmatrix} 0 \\ q_{21} \\ q_{31} \\ \vdots \\ q_{n1} \end{pmatrix} \tag{3.6}$$

Our formulation of the passage time quantity in Eqn. (3.3) states that we must observe at least one state-transition during the passage. In the case where $i \in \mathcal{J}$ (as for $L_{1\mathcal{J}}(s)$

in the above example), we therefore calculate the density of the cycle time to return to state i rather than requiring $L_{i\mathcal{J}}(s) = 1$.

Given a particular (complex-valued) s, Eqn. (3.6) can be solved for $L_{i\mathcal{J}}(s)$ by standard iterative numerical techniques for the solution of systems of linear equations in $A\vec{x} = \vec{b}$ form. In the context of the Laplace transform inversion algorithms both Euler and Laguerre can identify in advance the values of s at which $L_{i\mathcal{J}}(s)$ must be calculated in order to perform the numerical inversion. Therefore, if the algorithm requires m different values of $L_{i\mathcal{J}}(s)$ to calculate $f_{i\mathcal{J}}(t)$, Eqn. (3.6) will need to be solved m times.

The corresponding cumulative distribution function $F_{i\mathcal{J}}(t)$ of the passage time is obtained by integrating the density function. This integration can be achieved in terms of the Laplace transform of the density function with $F^*_{i\mathcal{J}}(s) = L_{i\mathcal{J}}(s)/s$. In practice, if Eqn. (3.6) is solved as part of the inversion process for calculating $f_{i\mathcal{J}}(t)$, the m values of $L_{i\mathcal{J}}(s)$ can be cached for future computation of $F_{i\mathcal{J}}(t)$ as required.

When there are multiple source markings, denoted by the vector $\mathcal{I}$, the Laplace transform of the response time density at equilibrium is:

$$L_{\mathcal{I}\,\mathcal{J}}(s) = \sum_{k \in \mathcal{I}} \alpha_k L_{k\mathcal{J}}(s)$$

where the weight α_k is the equilibrium probability that the state is $k \in \mathcal{I}$ at the starting instant of the passage. This instant is the moment of entry into state k; thus α_k is proportional to the equilibrium probability of the state k in the underlying embedded (discrete-time) Markov chain (EMC) of the CTMC with one-step transition matrix P as defined in Sect. 3.1.1. That is:

$$\alpha_k = \begin{cases} \pi_k / \sum_{j \in \mathcal{I}} \pi_j & : \text{if } k \in \mathcal{I} \\ 0 & : \text{otherwise} \end{cases} \tag{3.7}$$

where the vector $\vec{\pi}$ is any non-zero solution to $\vec{\pi} = \vec{\pi}P$. The row vector with components α_k is denoted by $\vec{\alpha}$.

3.4. Transient analysis. As well as the Laplace transform approach described above, passage time densities and quantiles in CTMCs may also be computed through the use of *uniformization* (also known as *randomization*). This transforms a CTMC into one in which all states have the same mean holding time $1/q$, by allowing *invisible* transitions from a state to itself. This is equivalent to a discrete-time Markov chain, after normalisation of the rows, together with an associated Poisson process of rate q. The one-step transition probability matrix P which characterises the one-step behaviour of the uniformized DTMC is derived from the generator matrix Q of the CTMC as:

$$P = Q/q + I \tag{3.8}$$

where the rate $q > \max_i |q_{ii}|$ ensures that the DTMC is aperiodic by guaranteeing that there is at least one single-step transition from a state to itself.

3.4.1. Uniformization for Transient Analysis of CTMCs. Uniformization has classically been used to conduct transient analysis of finite-state CTMCs [10, 11].

The transient state distribution of a CTMC is the probability that the process is in a state in $\mathcal{J}$ at time t, given that it was in state i at time 0:

$$\pi_{i\mathcal{J}}(t) = \mathbb{P}(X(t) \in j \mid X(0) = i)$$

where $X(t)$ denotes the state of the CTMC at time t.

In a uniformized CTMC, the probability that the process is in state j at time t is calculated by conditioning on $N(t)$, the number of transitions in the DTMC that occur in a given time interval [0,t]:

$$\pi_{i\mathcal{J}}(t) = \sum_{m=0}^{\infty} \mathbb{P}\big(X(t) \in \mathcal{J} \mid N(t) = m\big)\, \mathbb{P}\big(N(t) = m\big)$$

where $N(t)$ is given by a Poisson process with rate q and the state of the uniformized process at time t is denoted $X(t)$. Therefore:

$$\pi_{i\mathcal{J}}(t) = \sum_{n=1}^{\infty} \frac{(qt)^n e^{-qt}}{n!} \sum_{k \in \mathcal{J}} \pi_k^{(n)}$$

where:

$$\vec{\pi}^{(n+1)} = \vec{\pi}^{(n)} P \quad : \text{for } n \geq 0$$

and $\vec{\pi}^{(0)}$ is the initial probability distribution from which the transient measure will be measured (typically, for a single initial state i, $\pi_k = 1$ if $k = i$, 0 otherwise).

4. Modelling Constructs and Techniques.

4.1. Queues. Queues are important structures in performance models of network architectures and wireless network architectures especially. Queues are used to model any instance where there is the potential for competition for a resource – and thus also capture the delay in awaiting that resource's availability. This might be contention for a wireless channel, contention for a wireless service, or contention at an application level in a protocol.

As such queues have appeared in various forms in a large number of papers concerning stochastic process algebra. A small number of papers have specifically considered modelling queues [12, 13]. In addition a number of efficient solution techniques have been applied from queueing theory to stochastic process algebra. These include reversibility [14, 15], quasi-reversibility [16] and spectral expansion [17]. It is important to note that the tools supporting PEPA only allow analysis of models with a finite state space, i.e. bounded queues. However PEPA can be used to specify models of unbounded queues and alternative solution methods can be applied on an *ad hoc* basis (as in [17]).

In this section, we will introduce some basic techniques for modelling queues in PEPA. There are two basic approaches to modelling queues in PEPA; the state-based and the place-based approaches. The state-based is by far the most commonly used and generally leads to the smaller state space in the underlying CTMC. For example, consider the following model of an $M/M/1/K$ queue.

$$Queue_0 \stackrel{def}{=} (arrive, \top).Queue_1$$

$$Queue_i \stackrel{def}{=} (arrive, \top).Queue_{i+1} + (service, \top).Queue_{i-1} \quad : 1 \leq i < K$$
$$Queue_K \stackrel{def}{=} (service, \top).Queue_{K-1}$$

$$Server \stackrel{def}{=} (service, \mu).Server$$

$$Arrivals \stackrel{def}{=} (arrive, \lambda).Arrivals$$

$$Server \underset{\{service\}}{\bowtie} Queue_0 \underset{\{arrive\}}{\bowtie} Arrivals$$

In this representation the subscript associated with the queue component depicts the number of jobs in the queue, and the *Arrivals* and *Server* components control the arrivals and departures from the queue. A simple modification to the *Server* component can be made to consider breakdowns (without job loss on failure).

$$Server \stackrel{def}{=} (service, \mu).Server + (breakdown, \xi).Server'$$
$$Server' \stackrel{def}{=} (repair, \eta).Server$$

An alternative representation of the queue components is possible whereby each place in the queue is modelled separately and the whole queue is a parallel composition of all the places. This is illustrated below, without the *Server* and *Arrivals* components as these are unchanged.

$$Queue_0 \stackrel{def}{=} (arrival, \top).Queue_1$$
$$Queue_1 \stackrel{def}{=} (service1, \top).Queue_0$$

$$Queue \stackrel{def}{=} (Queue_0 \parallel \cdots \parallel Queue_0)$$

These two specifications of the $M/M/1/K$ queue give rise to different underlying CTMCs. The state-based representation gives rise to K states, but the place-based representation gives rise to 2^K states. For this reason one would not normally choose to use the place-based representation, however there are some advantages in certain situations. Sereno [18] used such a representation in a model that had a product form solution; the state-based approach would not have had such a solution in that case. In addition, this style of model is *potentially* amenable to a form of analysis based on ordinary differential equations that has recently been applied to PEPA [19] and is supported by the Dizzy tool [20]. This analysis effectively counts the number of components in a given derivative without recourse to deriving the underlying CTMC. Thus, it is possible to count the number of components behaving as derivative $Queue_1$ to calculate the number of jobs in the queue. This form of analysis is extremely efficient and so even though the underlying CTMC is larger, it is still possible to analyse models with far more derivatives (in this case larger queues). Much work remains to be done to facilitate this analysis in PEPA, not least in the acceptance of the solution methods for performance modelling and applicability to a wider range of models. As such we will not describe this analysis any further here, but the interested reader is encouraged to consult the PEPA home page and related bibliography for the latest developments.

Using either of these approaches it is clearly possible to model a wide range of single queue behaviours (see [12] for a fairly comprehensive survey of these), however the

more interesting models arise when multiple queues interact. Such interaction can be classified in terms of queues in parallel or in sequence; networks of queues obviously include both situations. Queues in parallel are found in many routing and service problems. For example, consider the distribution of jobs between $M/M/1/K$ queues according to the operational state of the servers where in the PEPA description below we use an indexing notation in component definitions such as

$$Arrivals \stackrel{def}{=} (arrive_j, \lambda_j).Arrivals \quad : j = 1, 2$$

as an abbreviation for:

$$\begin{aligned} Arrivals \stackrel{def}{=} & (arrive_1, \lambda_1).Arrivals \\ & + (arrive_2, \lambda_2).Arrivals \end{aligned}$$

The definition of such queues is the following:

$$\begin{aligned} Queue(j)_0 &\stackrel{def}{=} (arrive_j, \top).Queue(j)_1 \quad : j = 1, \ldots, n \\ Queue(j)_i &\stackrel{def}{=} (arrive_j, \top).Queue(j)_{i+1} + (service_j, \top).Queue(j)_{i-1} \\ &\qquad : 1 \leq i < K,\ j = 1, \ldots, n \\ Queue(j)_K &\stackrel{def}{=} (service_j, \top).Queue(j)_{K-1} \quad : j = 1, \ldots, n \end{aligned}$$

$$\begin{aligned} Server_j &\stackrel{def}{=} (service_j, \mu_j).Server_j + (breakdown_j, \xi_j).Server'_j \quad : j = 1, \ldots, n \\ Server'_j &\stackrel{def}{=} (repair_j, \eta_j).Server_j \end{aligned}$$

$$\begin{aligned} Arrivals &\stackrel{def}{=} (arrive_j, \lambda_j).Arrivals + (breakdown_j, \top).Arrivals_j \quad : j = 1, \ldots, n \\ Arrivals_j &\stackrel{def}{=} (arrive_{j\oplus 1}, \lambda_1 + \lambda_2).Arrivals_j + (repair_j, \top).Arrivals \\ &\qquad + (breakdown_{j\oplus 1}, \top).Arrivals_\emptyset \quad : j = 1, \ldots, n \\ Arrivals_\emptyset &\stackrel{def}{=} (repair_{j\oplus 1}, \top).Arrivals_j \quad : j = 1, \ldots, n \end{aligned}$$

$$((Server_1 \parallel \ldots \parallel Server_n) \underset{L_1}{\bowtie} Arrivals) \underset{L_2}{\bowtie} (Queue(1) \parallel \ldots \parallel Queue(n))$$

where $j = 1, \ldots, n$, $L_1 = \{arrive, repair_j, breakdown_j\}$, $L_2 = \{arrive_j, service_j\}$, $i \oplus 1 = i + 1$ when $1 \leq i < n$ and $n \oplus 1 = 1$.

It is similarly possible to specify random, round robin and shortest queue routing strategies (see [12]). Priority service is also an important consideration. For example, consider a model where high priority jobs (type 2) are always served before low priority jobs (type 1).

$$\begin{aligned} Queue(j)_0 &\stackrel{def}{=} (arrive_j, \top).Queue(j)_1 \quad : j = 1, 2 \\ Queue(1)_1 &\stackrel{def}{=} (arrive_1, \top).Queue(1)_2 + (service_1, \top).Queue(1)_0 \\ Queue(2)_1 &\stackrel{def}{=} (arrive_2, \top).Queue(2)_2 + (last_2, \top).Queue(2)_0 \\ Queue(j)_i &\stackrel{def}{=} (arrive_j, \top).Queue(j)_{i+1} + (service_j, \top).Queue(j)_{i-1} \\ &\qquad : 2 \leq i < K, j = 1, 2 \end{aligned}$$

$$Queue(j)_K \stackrel{def}{=} (service_j, \top).Queue(j)_K \quad : j = 1, 2$$

$$Arrivals \stackrel{def}{=} (arrive_j, \lambda_j).Arrivals \quad : j = 1, 2$$

$$Server_1 \stackrel{def}{=} (arrive_2, \top).Server_2 + (service_1, \mu_1).Server_1$$
$$Server_2 \stackrel{def}{=} (arrive_2, \top).Server_2 + (service_2, \mu_2).Server_2 + (last_2, \mu_2).Server_1$$

$$Server_1 \underset{M_1}{\bowtie} (Queue(1) \parallel Queue(2)) \underset{M_2}{\bowtie} Arrivals$$

where $M_1 = \{arrive_2, service_1, service_2, last_2\}$, $M_2 = \{arrive_1, arrive_2\}$. This model illustrates an important technique in modelling with process algebra, namely the trigger action. In this case the $service_2$ action is replaced in $Queue(2)_1$ with $last_2$. This allows the server behaviour to alter when the last high priority job has been served. An additional component could be added to this model to ensure that the total number of jobs is capped (at k).

$$Queue_0 \stackrel{def}{=} (arrive_j, \top).Queue_1 \quad : j = 1, 2$$
$$Queue_1 \stackrel{def}{=} (arrive_j, \top).Queue_2 + (service_1, \top).Queue_0 + (last_2, \top).Queue_0$$
$$Queue_i \stackrel{def}{=} (arrive_j, \top).Queue_{i+1} + (service_j, \top).Queue_{i-1} \quad : 2 \leq i < k$$
$$Queue_k \stackrel{def}{=} (service_j, \top).Queue_{k-1}$$

$$QEntry \stackrel{def}{=} Arrivals \underset{M_2}{\bowtie} Queue_0$$

$$Server_1 \underset{M_1}{\bowtie} (Queue(1)_0 \parallel Queue(2)_0) \underset{M_3}{\bowtie} QEntry$$

where $M_3 = \{arrive_1, arrive_2, last_2, service_1, service_2\}$. Finally in this section we turn our attention to models where job departure from one queue corresponds to an arrival at another. For example, consider two $M/M/1/K$ queues in tandem.

$$Queue1_0 \stackrel{def}{=} (arrive_1, \top).Queue1_1$$
$$Queue1_i \stackrel{def}{=} (arrive_1, \top).Queue1_{i+1} + (service_1, \top).Queue1_{i-1} \quad : 1 \leq i < K$$
$$Queue1_K \stackrel{def}{=} (service_1, \top).Queue1_{K-1}$$

$$Queue2_0 \stackrel{def}{=} (service_1, \top).Queue2_1$$
$$Queue2_i \stackrel{def}{=} (service_1, \top).Queue2_{i+1} + (service_2, \top).Queue2_{i-1} \quad : 1 \leq i < K$$
$$Queue2_K \stackrel{def}{=} (service_2, \top).Queue2_{K-1}$$

$$Server1 \stackrel{def}{=} (service_1, \mu_1).Server1$$

$$Server2 \stackrel{def}{=} (service_2, \mu_2).Server2$$

$$Arrivals \stackrel{def}{=} (arrive_1, \lambda_1).Arrivals$$

$$(Arrivals \underset{\{arrive_1\}}{\bowtie} Queue1_0 \underset{\{service_1\}}{\bowtie} Server1) \underset{\{service_1\}}{\bowtie} (Queue2_0 \underset{\{service_2\}}{\bowtie} Server2)$$

In this model jobs are blocked (awaiting service) at the first node if the second queue is full. The model is simple to adapt by changing $Queue2_K$ to specify the case where jobs are lost after service at node 1 when the second queue is full.

$$Queue2_K \stackrel{def}{=} (service_1, \top).Queue2_K + (service_2, \top).Queue2_{K-1}$$

4.2. Phase-type distributions. The exponential distribution is not always the most realistic event duration distribution when considering models of wireless networks. In particular, it is not always an accurate representation of an arrival process.

Although there exists a version of PEPA – semi-Markov PEPA [21] – that can be used to model activities with general distributions, the existence of general distributions in a model restricts the use of true concurrency at time of analysis. One alternative strategy is to make use of phase type distributions to approximate general distributions in a standard PEPA model. Phase type distributions are distributions constructed by combining multiple exponential random variables. These can be used to approximate most general distributions and approximations can be constructed using tools such as *EMpht* [22].

The Erlang distribution is a commonly used example of a phase type distribution which consists of an exponential distribution repeated k times. The *probability density function* (PDF) for the Erlang distribution is as follows:

$$f(x) = \frac{\lambda^k x^{k-1} e^{-\lambda x}}{(k-1)!}$$

In PEPA this is generally modelled as a ticking clock:

$$\begin{aligned} Clock_i &\stackrel{def}{=} (tick, t).Clock_{i-1} \quad : 1 < i \leq k \\ Clock_1 &\stackrel{def}{=} (event, t).Clock_k \end{aligned}$$

where $t = k\lambda$. The Erlang distribution is generally used to approximate deterministic events; the greater the value of k (i.e. the more ticks) then the closer to a deterministic delay the Erlang distribution becomes. However, it should also be noted that the larger the value of k, the more states there will be in the underlying CTMC. Hence, although we may wish to have 40 or 50 ticks to generate a nearly deterministic process, in practice a typical value of k is in the range $[5, 10]$. When studying network protocols the Erlang distribution is very useful for modelling timeouts.

The Erlang distribution is a special case of a more general phase type distribution known as the Cox distribution. The Cox distribution consists of an initial exponential, followed by a second exponential with probability α_1 (or finish with probability $1-\alpha_1$), followed by another exponential with probability α_2 and so on up to some maximum number of phases. The two phase Cox distribution, Cox_2, can easily be modelled in PEPA as follows:

$$Cox_2 \stackrel{def}{=} (phase_1, \alpha\lambda_1).(phase_2, \lambda_2) \ldots + (phase_1, (1-\alpha)\lambda_1) \ldots$$

Clearly the Cox distribution allows a greater range of conditions to be approximated. The variance is low if k is large and α_i is close to 1 for all i (i.e. it is nearly Erlang),

but the variance can be large if α_i is smaller and λ_i is quite different (as in the hyper-exponential below).

Another important phase type distribution is the hyper-exponential, or H_k, distribution, which is a random choice between k exponential distributions. The most commonly used hyper-exponential is the H_2-distribution, which has three parameters, α, μ_1 and μ_2 and the following *cumulative distribution function* (CDF).

$$F_{H_2} = 1 - \alpha e^{-\mu_1 t} - (1-\alpha)e^{-\mu_2 t} \quad : t \geq 0.$$

In PEPA this branching cannot be modelled explicitly except by introducing a new pair of actions over a choice operator. However, in practice branching made be represented implicitly at the preceding action. Thus if a job arrives into an empty queue and gets a hyper-exponential service, it could be represented thus:

$$\begin{aligned} Queue_0 &\stackrel{def}{=} (arrival, \alpha\lambda).Queue_{1a} + (arrival, (1-\alpha)\lambda).Queue_{1b} \\ Queue_{1a} &\stackrel{def}{=} (service, \mu_1).Queue_0 \\ Queue_{1b} &\stackrel{def}{=} (service, \mu_2).Queue_0 \end{aligned}$$

In this representation the probabilistic branch is made by the choice of one of two *arrival* actions. The arrivals themselves will occur at the rate λ, i.e. the sum of the two branches under the race condition. Alternatively a service component can be constructed as follows:

$$\begin{aligned} Server &\stackrel{def}{=} (service, \alpha\mu_1).Server + (service, (1-\alpha)\mu_1).Server' \\ Server' &\stackrel{def}{=} (service, \alpha\mu_2).Server + (service, (1-\alpha)\mu_2).Server' \end{aligned}$$

It is important to note that in this representation the *Server* component will always perform a *service* action at rate μ_1 first before having the opportunity to branch. Subsequent *service* actions will occur at either rate according to the branching probability α. This behaviour would not affect the overall steady state solution (since the start state is irrelevant in steady state when the model is irreducible), however it should be noted that inconsistent results may arise when transient or passage time analysis is naively applied to such a model.

An important feature of the hyper-exponential distribution is that it has a greater variance than an exponential distribution of the same mean (as long as $\mu_1 \neq \mu_2$ obviously). This is in contrast to the Erlang distribution; thus by a choosing between Erlang, exponential and hyper-exponential it is possible to consider a wide range of behaviours.

It is important to note that some care is required when using phase type distributions which can be interrupted by other events, but will resume from the point of interruption. The memoryless property of the exponential means that if all the actions are exponential then we do not need to worry about residual functions (as they too will all be exponential). Because phase type distributions are constructed from exponentials, it is generally a simple matter to calculate residual functions. For instance, the residual life of an H_2 random variable following an Erlang is easily calculated. As might be expected the result has an H_2-distribution, although with parameters α', μ_1 and μ_2. The branching probability in the residual H_2, α', is calculated by considering the weighted outcomes of the races of pairs of exponential distributions.

An important class of distributions for network modelling concerns *bursty* arrival processes. The most common way of modelling this is to use Markov modulated Poisson process (MMPP). In PEPA this is simple to model as follows:

$$Arrivals_{off} \stackrel{def}{=} (turnOn, \eta).Arrivals_{on}$$
$$Arrivals_{on} \stackrel{def}{=} (arrival, \lambda).Arrivals_{on} + (turnOff, \xi).Arrivals_{off}$$

There are many other phase type distributions that can be used and good approximations to most general distributions can be made. It is worth noting however that the more phases considered, the greater the impact on the size of the state space, which can be a limiting factor for some forms of analysis.

4.3. Parameter estimation. PEPA captures both behavioural and temporal aspects of a communication model. The behavioural model can be derived from observed software or hardware events, however the question arises of how to choose the numerical values which represent the exponential rates associated with the behavioural events.

Part of the answer can be obtained by looking at the technical specification of the system, the throughput of the signals, the speed of the machines or the protocols used. But that might not be adequate. Firstly because the technical features of some hardware (or software) can be theoretically right (or right in some particular conditions) but practically wrong. And secondly because a model is not a full description of the system but an abstraction of it. For instance, a single rate of some event in the model can encompass several rates of a chain of events in the system. Also, it is not always clear what part of the model matches a given part of the real system and so which performance features from the technical specification match the rates of the model.

4.3.1. Estimating the rates of the model according to observations. Another way to answer this question is to choose the rates according to measurements obtained from the real system, for instance, benchmarks, response distributions. While the first inconvenience expressed above vanishes, the second one still remains, that is how to correlate the observations with the rates of the model. If a rate models the occurrence of a single observable event then it may be straightforward to choose it. For instance, if the model describes a message which has been measured to be sent on average every 20ms the rate of such event is $\frac{1}{0.02} = 50s^{-1}$. But when the rates are part of some intricate and unobservable chain of events it becomes far more difficult to choose them.

For this reason an algorithm has been developed and implemented in the tool EMPEPA (see Sect. 5.4) to find the rates inside a PEPA model such that it maximises (possibly locally) the likelihood of a set of measured observations [23, 24].

The set of parameters to estimate are some rates inside the model, say $r_1, \ldots, r_n$. A mapping from those rates to $\mathbb{R}_+^n$ is called an estimate, denoted $\theta = (r_1 = r_1^\theta, \ldots, r_n = r_n^\theta)$ or simply $\theta = (r_1^\theta, \ldots, r_n^\theta)$. The goal is to find the best rate estimate, that is the one such that the model maximises the likelihood of the observations, which can be, in our current version of the algorithm:

1. A set of absorption times, in this case the PEPA model must describe a distribution.

2. A set of *partially observable executions* (POEs). Each POE is a sequence of events interleaved by the times spent between the observations of each pair of consecutive events. A POE is denoted:

$$init \xrightarrow{t_1} a_1 \dots a_k \xrightarrow{t_{stop}} stop$$

It starts in the initial state, which is represented by the keyword *init*, and stops when the observer stops the observation, which this is represented by the keyword *stop*. The events a_1 to a_k correspond to some visible actions in the PEPA model, that is, $a_i \neq \tau$, $i = 1, \dots, k$ (see Sect. 2).

The algorithm implemented in EMPEPA is based on the EM algorithm [25, 26] and is inspired by the work of Soren Asmussen, Olle Nerman and Marita Olsson for fitting phase-type distributions [22].

4.3.2. Principle of the EM algorithm. The EM (*Expectation-Maximisation*) algorithm is a general method that maximises the likelihood of a set of incomplete observations. The observations are incomplete in the sense that they reveal only a part of the internal state of the system at a given time. Usually the likelihood function, denoted $L(Y;\theta)$, is defined as the probability (or the probability density) of observing some data Y, according to some estimate θ. The algorithm computes iteratively a sequence of estimates, $\theta_0, \theta_1, \dots, \theta_i, \dots$ where θ_0 is the initial guess provided either by the user or by a random function. At each iteration the likelihood function increases:

$$L(Y;\theta_0) < L(Y;\theta_1) < \cdots < L(Y;\theta_i) < \cdots$$

making the next estimate better than the previous one. To perform that, each iteration executes the two following steps:

1. The E step: that completes the partial observations by their theoretical expectations. So the E step guesses what the observer cannot see.
2. The M step: that finds the estimate that maximises the likelihood of the observations once completed by the E step.

Computing these expectations requires a model of the system which is, at the current iteration (provided by θ_i), incorrect or inaccurate. Consequently the M step does not find the best model but, however, succeeds to improve it [25]. The maximisation (local or global) is then achieved by convergence.

Since the EM algorithm is a general method, it can be applied in a large variety of contexts, as it is in speech recognition [27], neural networks [28] and many others. It has, however, two drawbacks:

1. There are in general several local maxima and the algorithm does not necessarily converge to the best one or even a good one. Also, it is advisable to try the algorithm several times with different initial guesses and choose the best result.
2. It converges slowly. There are however recent results that accelerate its convergence by using Fisher's information [29].

4.3.3. The EM algorithm with PEPA. In the context of using the EM algorithm with PEPA the visible part of the observations are the sequences of visible

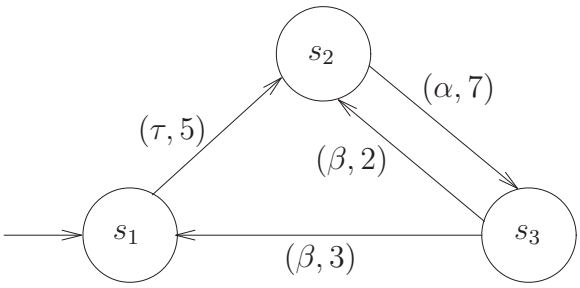

Fig. 4.1. The derivation graph G

actions and the time spent between their observations. The invisible parts are everything that happens between two visible actions. We present an example below.

Let G be the derivation graph of some PEPA model. G is depicted in Fig. 4.1. Suppose that an observer can observe completely the system between its initialisation and a certain time. An example of such an execution is given below:

$$s_1 \xrightarrow{0.05} (\tau, 5), s_2 \xrightarrow{0.07} (\alpha, 7), s_3 \xrightarrow{0.54} (\beta, 3), s_1 \xrightarrow{0.4} (\tau, 5), s_2 \xrightarrow{0.62} (\alpha, 7), s_3 \xrightarrow{0.37} \mathit{stop}$$

The execution starts in s_1, stays 0.05 unit of time, then takes the transition $(\tau, 5)$ to reach s_2, and so on, until the observer stops the observation, denoted at the end of the execution by the key word *stop*. The total time of the execution is $T_{stop} = 0.05 + 0.07 + \cdots + 0.37 = 2.05$. And the partially observable execution of it is:

$$\mathfrak{y} = \mathit{init} \xrightarrow{0.12} \alpha \xrightarrow{0.54} \beta \xrightarrow{1.02} \alpha \xrightarrow{0.37} \mathit{stop}$$

where the time between two visible actions is the sum of the time spent in all states between those two actions. There are of course many, and actually infinitely many, complete observations that match a POE.

The E step. As explained earlier the E step of the algorithm consists of completing, by theoretical expectations, the partially observable executions. A great simplification can be made here because the E step does not need to know the expectations of all details of some execution to calculate or maximise its likelihood. It actually only needs to know the *number of times every transition has been taken* and the *total time spent in every state*, as those quantities form a *sufficient statistic* for the process. That is, it is sufficient to know these statistics to calculate the probability density for any execution. The expectation of these quantities can be defined as the solution of a set of coupled ODEs (see [24] for more details). So the E step consists of solving an ODE system.

The M step. Once these expectations are computed, the maximisation of the complete observations is then given by setting each rate with the number of times its associated transition is taken divided by the time spent in the state that precedes it (this comes directly from the definition of the transition rate).

5. Tools. The wireless communications protocols which are used in present-day practice are complex and sophisticated. High-level process algebra models of these

protocols still contain considerable detail making the use of software tools mandatory for effective performance modelling of wireless communications protocols. In this section, we review analysis and model parameterisation tools for the PEPA language. These tools are used to compute performance analysis results for PEPA models such as steady-state behaviour, time-dependent transient behaviour, and the computation of passage-time quantiles (used for quality-of-service criteria). Software tools are also used to extract high-level model parameters from low-level measurement data in order to ensure that the performance results which are computed from a process algebra model closely mirror physical reality. The tools which we consider here are the PEPA Workbench, the Imperial PEPA Compiler (ipc), MEERCAT and EMPEPA.

5.1. PEPA Workbench. The PEPA Workbench [30] is a development environment for PEPA models which facilitates convenient solution of models for performance properties which can be derived from the steady-state probability distribution over the state space of the model. Such properties include *utilisation* of components and *throughput* of protocols but not time-dependent properties such as latency.

The PEPA Workbench will automatically generate a CTMC from the PEPA model and solve it for its equilibrium probability distribution using procedures of numerical linear algebra such as the pre-conditioned biconjugate gradient method (BCG) or successive over-relaxation (SOR). The relationship between the process algebra model and the CTMC representation is the following. The process terms (P_i) reachable from the initial state of the PEPA model by applying the operational semantics of the language form the states of the CTMC (X_i). For every set of labelled transitions between states P_i and P_j of the model $\{(\alpha_1, r_1), \ldots, (\alpha_n, r_n)\}$ add a transition with rate r between X_i and X_j where r is the sum of $r_1, \ldots, r_n$. The activity labels (α_i) are necessary at the process algebra in order to enforce synchronisation points, but are no longer needed at the Markov chain level.

In addition to the computation of quantitative results, the PEPA Workbench also facilitates qualitative model analysis to detect modelling errors such as mismatched synchronisations or out-of-order errors in sequences of activities. The quantitative results computed from a PEPA model will only be valid if the model itself is an accurate representation of the physical reality of the protocol and thus it is important to find and correct modelling errors before computing quantitative results. To this end the PEPA Workbench contains a range of debugging aids such as single-step animators and visual debuggers which can be useful components of the model analysis tool chain even if the performance results are computed from another modelling tool such as ipc or MEERCAT.

5.2. The Imperial PEPA Compiler (ipc). The Imperial PEPA Compiler (ipc) [9, 31] enables additional solution procedures and response time analysis capabilities over existing PEPA tool sets such as PEPA Workbench [32], PRISM [33] and Möbius [34].

ipc performs the translation from PEPA to a stochastic Petri net formalism that the HYDRA Markov chain analyser [35] employs. ipc converts each PEPA component submodel into a stochastic Petri net. The models are combined using shared net transitions to represent the shared actions of the PEPA synchronisation. ipc calculates the apparent rate of each component as it is combined in order to represent the complete PEPA synchronisation rate.

5.2.1. Stochastic Probes. ipc uses the stochastic probe measurement mechanism from [36] to specify how measures relate to models. A stochastic probe is a fragment of process algebra (say a component Q) that expresses a stochastic measure over a PEPA model, say M. By synchronising the probe with the model, $M \underset{L}{\bowtie} Q$, specified model behaviour (from set L) can be observed before turning a measurement on (having the probe change to a *running state*) and further user-defined behaviour (also in L) is looked for before switching the measurement off (as the probe switches to an *off state*). As HYDRA is a state-oriented formalism, ipc encodes the measurement (response time, transient, or steady-state) in terms of the observed state of the probe.

While normal PEPA components can affect both the temporal and functional execution of the process that they are cooperating with, a stochastic probe is designed not to interfere with the model it is measuring in any way. If it did, it would affect the measurement result and this is not desirable. To achieve this passive observation property, care is taken to ensure that any actions in the cooperation set, L, that the probe is not currently interested in are enabled. Additionally, all probe actions are assigned passive rate $\top$.

As specified in [37], a stochastic probe definition, R, has the following syntax:

$$\begin{array}{lcl} R & ::= & A \mid T, T \mid S \\ S & ::= & T | S \mid T \\ T & ::= & R \mid R\{n\} \mid R\{m,n\} \mid R^{+} \mid R^{\star} \mid R? \mid R/act \\ A & ::= & act \mid act{:}\mathsf{start} \mid act{:}\mathsf{stop} \end{array} \tag{5.1}$$

act is an action label that matches a label in the system being measured. Any action specified in the probe has to be observed in the model before the probe can advance a state. An action, act, can also be distinguished or tagged as a *start* or *stop* action in a probe and signifies an action which will start or stop a measurement, respectively.

R_1, R_2 is the **sequential** operation. R_1 is matched against the system's operation, then R_2 is matched.

$R_1 \mid R_2$ is the **choice** operation. Either R_1 or R_2 is matched against the system being probed.

$R^{\star}$ is the **closure** operation, where zero or more copies of R are matched against the system.

$R?$ is the **optional** operation, matching zero or one copy of R against the system.

$R\{n\}$ is the **iterative** operation. A fixed number of sequential copies of R are matched against the system e.g. $R\{3\}$ is simply shorthand for R, R, R.

$R\{m,n\}$ is the **range** operation. Between m and n copies of R are matched against the system's operation. $R\{m,n\}$ is equivalent to $R\{n,m\}$, and we consider the canonical form to have the smaller index first.

R^{+} is the **positive closure** operation, where one or more copies of R are matched against the system. It is syntactic sugar for $R, R^{\star}$.

R/act is the **without** operation. R must begin again whenever the probe sees an act action that is not matched by R.

5.2.2. Example: Wireless active badge sensor network. In the original active badge model, described in [38], there are 4 rooms on a corridor, all installed

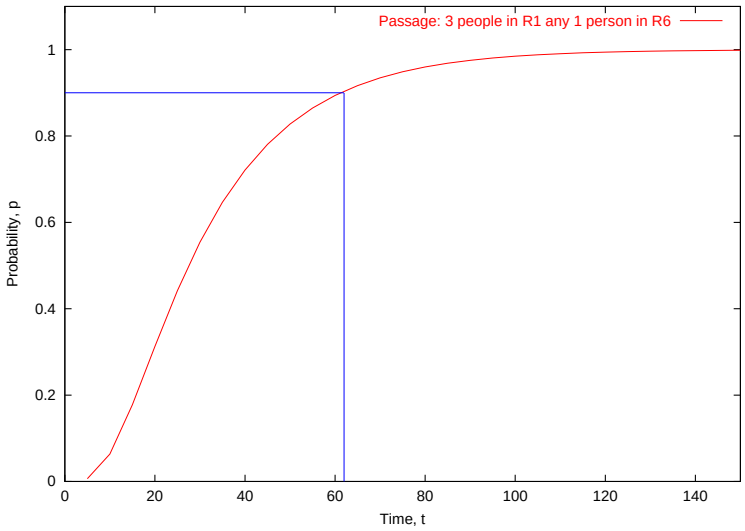

Fig. 5.1. Wireless active badge model: cumulative response time distribution and quantile for any 3 people activating the wireless sensor in room 6.

with wireless active badge sensors, and a single person who can move from one room to an adjacent room. The wireless sensors are linked to a database which records which sensor has been activated last. In the PEPA model of the wireless active badge system below, we have M people in N rooms with sensors and a database that can be in one of N states.

$$
\begin{aligned}
Person_1 &\stackrel{def}{=} (reg_1, r).Person_1 + (move_2, m).Person_2 \\
Person_i &\stackrel{def}{=} (move_{i-1}, m).Person_{i-1} + (reg_i, r).Person_i + (move_{i+1}, m).Person_{i+1} \\
&\qquad : \text{for } 1 < i < M \\
Person_M &\stackrel{def}{=} (move_{M-1}, m).Person_{M-1} + (reg_N, r).Person_M \\
WSensor_i &\stackrel{def}{=} (reg_i, \top).(rep_i, s).WSensor_i \quad : \text{for } 1 \leq i \leq N \\
Dbase_i &\stackrel{def}{=} \sum_{j=1}^{N} (rep_j, \top).Dbase_j \quad : \text{for } 1 \leq i \leq N \\
Sys &\stackrel{def}{=} \left(\prod_{j=1}^{M} Person_j \underset{R_1}{\bowtie} \prod_{j=1}^{N} WSensor_j \right) \underset{R_2}{\bowtie} Dbase_1
\end{aligned}
$$

where $R_1 = \{reg_i : 1 \leq i \leq N\}$ and $R_2 = \{rep_i : 1 \leq i \leq N\}$ and $\prod_{i=1}^{N} A_i \equiv A_1 \parallel A_2 \parallel \cdots \parallel A_n$.

In the model, $Person_i$ represents a person in room i, $WSensor_i$ is the sensor in room i and $Dbase_i$ is the state of the database. A person in room i can either move to room $i-1$ or $i+1$ or, if they remain there long enough, set off the sensor in room i, which registers its activation with the database. To maintain a reasonable state space, this is a simple database which does not attempt to keep track of every individual's location, rather it remembers the last movement that was made by any person in the system.

We study an instance of the model with $M = 3$ people and $N = 6$ rooms, to give a global state space of $82,944$ states. To specify the measure that represents the act of 3 people triggering the sensor in room 6, we use the probe:

$$reg_1 : \mathsf{start}, rep_6\{2\}, rep_6 : \mathsf{stop}$$

This starts the clock on seeing a reg_1 action and requires three rep_6 actions before stopping the measurement. In this case, we require a cumulative distribution function (CDF) of the time to see 3 people trigger sensor in room 6, as shown in Fig. 5.1. From this, we can derive the performance quantile that with probability 0.9, three people will have triggered the sensors in room 6 by time $t = 62$.

5.3. MEERCAT**.** MEERCAT classifies models where we can apply certain compositional analysis techniques for steady-state analysis. It applies the Reversed Compound Agent Theorem (RCAT) [39] to a Markovian process algebra model of two cooperating components to generate alternate components which can be analysed separately.

We may apply the RCAT in a system that satisfies several conditions, given in [39, 40]. We take a pair of cooperating components, and examine their reverse processes, that is as though they were moving backwards in time. Then, if the cooperation satisfies RCAT's conditions, we can build two separate components, based on the reverse processes, whose steady-state behaviour allows us to calculate the steady-state occupation probabilities of the original cooperating model. In this manner we can avoid examining the global state space of the joint cooperation.

For example, consider a pair of tandem queues, which is the simplest form of a Jackson network, and thus has a well-known product-form at equilibrium, as pictured in Fig. 5.2.

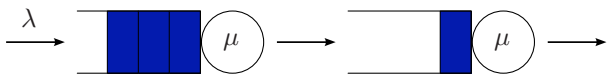

Fig. 5.2. A tandem pair of queues

In PEPAinf, as defined in Sect. 2.1, the system in Fig. 5.2 is specified as:

$$\begin{aligned}
P_{i=0} &\stackrel{def}{=} (arr, r).P_{i+1} \\
P_{i>0} &\stackrel{def}{=} (arr, r).P_{i+1} + (int, s).P_{i-1} \\
Q_{i=0} &\stackrel{def}{=} (int, \top).Q_{i+1} \\
Q_{i>0} &\stackrel{def}{=} (int, \top).Q_{i+1} + (out, s').Q_{i-1} \\
Sys &\stackrel{def}{=} P_{i=0} \underset{\{int\}}{\bowtie} Q_{i=0}
\end{aligned}$$

By convention, we name the reverse process of X as $\overline{X}$, and an action a corresponds to the reversed action $\overline{a}$ in the reverse process.

Now, to apply RCAT to this we look first at the top-level composition, $P_{i=0} \underset{\{int\}}{\bowtie} Q_{i=0}$, and check that the action int is passively enabled in all the derivative states of $Q_{i=0}$ and $\overline{int}$ is passively enabled in all the derivative states of $\overline{P_{i=0}}$.

If required, we could make the whole reverse process by reversing cooperating active actions to become passive actions. We instead build two separate components (that is, with no passive actions) introducing new, unknown rates. Due to the conditions of RCAT – these processes, which we call $\overline{P'}$ and $\overline{Q'}$, will have the same steady-state behaviour as the cooperation of the whole reversed process, given in Eqn. (5.2).

The rate of $\overline{out}$ in $\overline{Q'}$ is x_{int}, since this is the rate into the Q component in the forward process. $\overline{int}$ in $\overline{Q'}$ proceeds at rate s' and in $\overline{P}$ as rate r by a similar argument. Thus, we define our $\overline{P_{i=0}}$ and $\overline{Q'_{i=0}}$ as:

$$\begin{aligned}
\overline{P'_{i=0}} &\stackrel{def}{=} (\overline{int}, r).\overline{P'_{i+1}} \\
\overline{P'_{i>0}} &\stackrel{def}{=} (\overline{arr}, s).\overline{P'_{i-1}} + (\overline{int}, r).\overline{P'_{i+1}} \\
\overline{Q'_{i=0}} &\stackrel{def}{=} (\overline{out}, x_{int}).\overline{Q'_{i+1}} \\
\overline{Q'_{i>0}} &\stackrel{def}{=} (\overline{out}, x_{int}).\overline{Q'_{i+1}} + (\overline{int}, s').\overline{Q'_{i-1}}
\end{aligned}$$

where $x_{int} = r$ from the traffic equations.

The unnormalised steady-state probability that the joint, forward process is in state P_i, Q_j is the product-form:

$$\pi_{ij} = \pi_P(i)\pi_Q(j)$$

where $\pi_P(i)$ is the steady-state probability that queue P is in state i, and similarly $\pi_Q(j)$ is the steady-state probability that Q is in state j.

Should it be needed, the actual reverse process:

$$\begin{aligned}
\overline{P_{i=0}} &\stackrel{def}{=} (\overline{int}, \top).\overline{P_{i+1}} \\
\overline{P_{i>0}} &\stackrel{def}{=} (\overline{arr}, x_{int}).\overline{P_{i-1}} + (\overline{int}, \top).\overline{P_{i+1}} \\
\overline{Q_{i=0}} &\stackrel{def}{=} (\overline{out}, x_{int}).\overline{Q_{i+1}} \\
\overline{Q_{i>0}} &\stackrel{def}{=} (\overline{out}, x_{int}).\overline{Q_{i+1}} + (\overline{int}, s').\overline{Q_{i-1}} \\
Sys &\stackrel{def}{=} \overline{P_{i=0}} \underset{\{\overline{int}\}}{\bowtie} \overline{Q_{i=0}}
\end{aligned} \tag{5.2}$$

$\overline{P}$ and $\overline{Q}$ are identical to $\overline{P'}$ and $\overline{Q'}$, replacing the explicit rates we inserted for the cooperation with a proper cooperating action.

5.3.1. Tandem queue with feedback and external interactions. As depicted in Fig. 5.3, this is the most general form of a pair of $M/M/1$ queues with positive customers. Without loss of generality, it is equally likely to depart as it is to transfer to the other queue. Note also that this system is completely symmetric, *int* is no more internal than *repeat*, it is merely a matter of how you look at the system. In PEPAinf:

$$\begin{aligned}
P_{i=0} &\stackrel{def}{=} (arr, r).P_{i+1} + (repeat, \top).P_{i+1} \\
P_{i>0} &\stackrel{def}{=} (arr, r).P_{i+1} + (repeat, \top).P_{i+1} + \\
&\qquad (int, s/2).P_{i-1} + (extdep, s/2).P_{i-1} \\
Q_{i=0} &\stackrel{def}{=} (extarr, r').Q_{i+1} + (int, \top).Q_{i+1} \\
Q_{i>0} &\stackrel{def}{=} (int, \top).Q_{i+1} + (repeat, s'/2).Q_{i-1} +
\end{aligned}$$

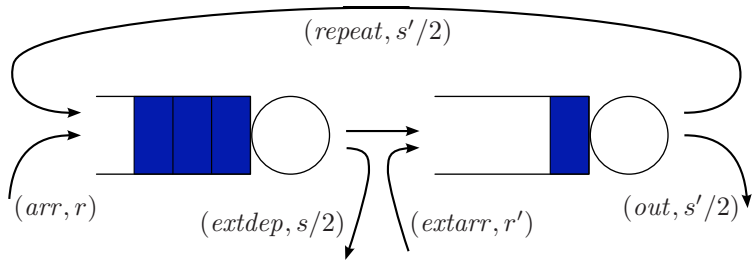

Fig. 5.3. Tandem queue with feedback and external arrivals and departures to both queues.

$$
\begin{aligned}
&(out, s'/2).Q_{i-1}\\
Sys \stackrel{def}{=} P_{i=0} &\underset{\{int,repeat\}}{\bowtie} Q_{i=0}
\end{aligned}
$$

To build the two reversed processes we again replace the passive rates as described above and form two independently operating queues:

$$
\begin{aligned}
\overline{P'_{i=0}} &\stackrel{def}{=} (\overline{extdep}, \frac{r + x_{repeat}}{2}).\overline{P'_{i+1}} + (\overline{int}, \frac{r + x_{repeat}}{2}).\overline{P'_{i+1}}\\
\overline{P'_{i>0}} &\stackrel{def}{=} (\overline{arr}, \frac{sr}{r + x_{repeat}}).\overline{P'_{i-1}} + (\overline{repeat}, \frac{sx_{repeat}}{r + x_{repeat}}).\overline{P'_{i-1}}\\
&\quad +(\overline{extdep}, \frac{r + x_{repeat}}{2}).\overline{P'_{i+1}} + (\overline{int}, \frac{r + x_{repeat}}{2}).\overline{P'_{i+1}}\\
\overline{Q'_{i=0}} &\stackrel{def}{=} (\overline{repeat}, \frac{x_{int} + r'}{2}).\overline{Q'_{i+1}} + (\overline{out}, \frac{x_{int} + r'}{2}).\overline{Q'_{i+1}}\\
\overline{Q'_{i>0}} &\stackrel{def}{=} (\overline{int}, \frac{s'x_{int}}{x_{int} + r'}).\overline{Q'_{i-1}} + (\overline{extarr}, \frac{s'r'}{x_{int} + r'}).\overline{Q'_{i-1}} +\\
&\quad (\overline{repeat}, \frac{x_{int} + r'}{2}).\overline{Q'_{i+1}} + (\overline{out}, \frac{x_{int} + r'}{2}).\overline{Q'_{i+1}}
\end{aligned}
$$

Where the rate equations are:

$$x_{int} = \frac{r + x_{repeat}}{2} \quad \text{and} \quad x_{repeat} = \frac{r + x_{int}}{2}$$

Thus $x_{int} = r$ and $x_{repeat} = r$.

The MEERCAT tool may be used interactively on the PerformDB web site:

`http://performdb.org/tools/meercat/`

or downloaded from there to be run locally.

5.4. EMPEPA. EMPEPA is a tool that allows the user to find the rates in a PEPA model so that they maximise (possibly locally) the likelihood of a set of observations obtained from executions of a real system. EMPEPA is useful when it is not possible to calculate directly the transition rates by using the simple formula:

$$\text{transition rate} = \frac{\text{number of times the transition is taken}}{\text{time spent in the state preceding the transition}} \tag{5.3}$$

Indeed, when the model is only partially observable these quantities can be unknown.

5.4.1. Example: Extracting rates from observations. Let P be the following PEPA model:

$$\begin{array}{rcl} P & \stackrel{def}{=} & (\alpha, x).P + (\tau, 4).P' \\ P' & \stackrel{def}{=} & (\beta, y).P' + (\alpha, z).P \end{array}$$

The goal of this example is to show how EMPEPA can find unknown rate values inside a model according to incomplete observations. The unknown values are represented by the variables x, y, z in the PEPA model. It is worth noting that simply observing the actions α and β is not sufficient to deduce the internal state of the system (P or P') nor the time spent in them, and so the designer cannot use the formula of Eqn. (5.3) to calculate the rates of the model due to this lack of information.

We now describe step by step a procedure that allows the user to recover the rates x, y, z based on previously generated observations.

1. Firstly these variables are initialised with the following values:

$$x = 1, y = 2, z = 3$$

2. Then, the model is simulated using the simulation functions of EMPEPA. The result is a set of partially observable executions (POEs) written in a file. This is obtained by typing the following command:

```
empepa --simulate -n 100 -t 0.08 tinyExample.pepa -o POEs.data
```

   Option `--simulate` launches EMPEPA in simulation mode. Option `-n` indicates the number of POEs to generate. Option `-t` defines the stop rate of the POEs, the value 0.08 means that in average a POE will last $\frac{1}{0.08} = 12.5$ seconds. `tinyExample.pepa` contains the PEPA model with the initialisation of the unknown values and `POEs.data` is the file containing the generated POEs after simulation. A piece of the file `POEs.data` is displayed below:

```
init -5.019937e-02> a -1.342709e+00> b -2.160762e-01> a[...]stop
init -4.008800e-01> a -1.622834e-01> stop
init -1.762623e+00> a -3.723575e-01> a -2.002015e+00> b[...]stop
                              ⋮
```

   where `init -5.019937e-02> a` stands for $init \xrightarrow{5.019937e-02} \alpha$.
3. Secondly the variables are changed by:

$$x = 0.2, y = 4, z = 1$$

   The model is saved in the new file `tinyExample-MODIF.pepa`.
4. EMPEPA is asked to find the previous values of the model on the basis of the POEs contained in the file `POEs.data`. EMPEPA is invoked with the following command:

```
empepa -i 10 --algo=rk4 tinyExample-MODIF.pepa POEs.data
```

   Option `-i` gives the number of iterations of the algorithm. Option `--algo` defines the resolution method of the ODEs, `rk4` stands for 4^{th} order Runge-Kutta.

5. After executing the 10 iterations (taking a couple of seconds) EMPEPA returned the following values, close to the initial ones as listed in Step 1 above.

$$x = 1.02, y = 2.04, z = 2.95$$

The results are not exact due to the finite number of observations.

EMPEPA has been successfully used on models of hundreds of states and thousands of observations (see [23]), but the total amount time to converge to a good solution could take many days. Also for the moment EMPEPA supports only PEPA models with synchronisations between active and passive transitions. However, optimisations and extensions are possible and further research is ongoing to improve its efficiency and its usability.

5.4.2. The EMPEPA tool. EMPEPA is an open source project coded in C and compatible with UNIX (the installation is simple and performed by the tools Automake and Autoconf). The project is distributed under the terms of the GPL and located at the following address:

`http://empepa.sourceforge.net/`

It contains an implementation of the algorithm described in Sect. 4.3 plus some simulation functions. The observations can be absorption durations or partially observable executions (POEs) (as described in Sect. 4.3). The resolution of the ODEs, required during the E step, is performed by the ODE solver of the GNU Scientific Library (GSL) offering numerous solving methods such as, 4^{th} order Runge-Kutta, Runge-Kutta Prince-Dormand (8,9) method and implicit 4^{th} order Runge-Kutta at Gaussian points, amongst others.

The syntax describing a PEPA model in EMPEPA is based on the one used in the model checker PRISM [41] with the additional keyword `var` to declare the set of variables of the model to estimate. The user invokes EMPEPA with the command `empepa` with a set of options. All the options will not be detailed here, full detail can be obtained in the documentation of EMPEPA located at the website or by invoking EMPEPA with the option `--help`. Instead, a simple example will be given below to enlighten its functionalities.

6. Wireless Protocol Modelling Example. This example[1] is based on a model of IEEE 802.11 developed by Sridhar and Ciobanu in [3]. IEEE 802.11 uses a *carrier sense multiple access collision avoidance* (CSMA/CA) access mechanism. A node wishing to send listens to the medium; if it is quiescent for some period (the *DCF InterFrame Space*, or DIFS, typically $128\mu s$) it transmits a control packet (following a short delay, VLUN) called a *Request To Send* (RTS). The receiving node responds with a *Clear To Send* (CTS) control packet if it is available. If the sending node does not receive the CTS packet within a given timeout period, or it detects a collision with an RTS from another sender, it will enter a back-off behaviour before attempting to send again. Assuming the CTS has arrived successfully, the sender will wait (the *Short InterFrame Space*, or SIFS) and then send the data.

[1]The PEPA model of the IEEE 802.11 protocol together with results presented in this section can be found at `http://performdb.org/models/ieee-802.11/` – any future improvements to this model will also be posted here.

The model used here is of two sender-receiver pairs ($i \in \{1,2\}$) with a single communication medium. All messages on the medium are assumed to be transmitted with a certain delay (mean $1/delay$). Thus a sender ($Sender_i$) will generate a message and listen to the media before sending the RTS packet (action rts_i). If the medium is available it will then forward this RTS to the receiver (as action r_i). The corresponding receiver ($Timer_i$) will acknowledge the RTS with its own CTS (action cts_i) which will be forwarded by the medium after a delay (as action c_i). On receipt of the CTS (c_i), sender i will send the message data (action $data_i$) which is forwarded to the receiver by the medium (as action d_i). When the receiver receives the data it will send an acknowledgement (action ack_i) which is forwarded to the sender by the medium (as action a_i). The sender will return to its start state once it has received the acknowledgement.

The sender can adopt an alternative behaviour after sending the RTS, corresponding to a collision. A collision will only occur if more than one sender attempts an RTS at the same time. In the model this is represented within the medium, where an r_i action has yet to complete (the forwarding of the RTS from sender i) and another sender j initiates a rts_j action. In such a situation both senders will detect the collision after some (short) delay and the medium will return to being idle. Following the collision each sender will attempt an RTS again, after a back-off delay. The duration of this delay is crucial to the performance of the system, and was the focus of the study by Sridhar and Ciobanu in [3]. The RTS may be successful (hence a CTS is received), or a further collision occurs and another back-off cycle is started.

In [3], the model erroneously went back to the initial RTS state ($Sender_{i1}$), rather than going through another back-off/acknowledge cycle. Another slight difference in our model is we model the senders as detecting the collision simultaneously, whereas in [3] $Sender_1$ would always detect the collision before $Sender_2$. As such the results here differ slightly from those in [3].

$$
\begin{aligned}
Sender_{i0} &\stackrel{def}{=} (gm, difs).Sender_{i1} \\
Sender_{i1} &\stackrel{def}{=} (rts_i, nav).Sender_{i2} \\
Sender_{i2} &\stackrel{def}{=} (c_i, \top).Sender_{i3} + (coll, \top).Sender_{i4} \\
Sender_{i3} &\stackrel{def}{=} (data_i, sifs).Sender_{i5} \\
Sender_{i4} &\stackrel{def}{=} (rts_i, backoff).Sender_{i6} \\
Sender_{i5} &\stackrel{def}{=} (a_i, \top).Sender_{i0} \\
Sender_{i6} &\stackrel{def}{=} (c_i, \top).Sender_{i3} + (coll, \top).Sender_{i4} \\
&\qquad : \text{for } i = 1, 2 \\
Timer_{i0} &\stackrel{def}{=} (r_i, \top).Timer_{i1} \\
Timer_{i1} &\stackrel{def}{=} (cts_i, sifs).Timer_{i2} \\
Timer_{i2} &\stackrel{def}{=} (d_i, datato).Timer_{i3} \\
Timer_{i3} &\stackrel{def}{=} (ack_i, sifs).Timer_{i0} \\
&\qquad : \text{for } i = 1, 2 \\
Medium_0 &\stackrel{def}{=} (rts_1, \top).Medium_1 + (cts_1, \top).Medium_2 + (data_1, \top).Medium_3 \\
&\quad + (ack_1, \top).Medium_4 + (rts_2, \top).Medium_5 + (cts_2, \top).Medium_6
\end{aligned}
$$

$$+ (data_2, \top).Medium_7 + (ack_2, \top).Medium_8$$

$$Medium_1 \stackrel{def}{=} (r_1, delay).Medium_0 + (rts_2, \top).Medium_9$$

$$Medium_2 \stackrel{def}{=} (c_1, delay).Medium_0$$

$$Medium_3 \stackrel{def}{=} (d_1, delay).Medium_0$$

$$Medium_4 \stackrel{def}{=} (a_1, delay).Medium_0$$

$$Medium_5 \stackrel{def}{=} (r_2, delay).Medium_0 + (rts_1, \top).Medium_9$$

$$Medium_6 \stackrel{def}{=} (c_2, delay).Medium_0$$

$$Medium_7 \stackrel{def}{=} (d_2, delay).Medium_0$$

$$Medium_8 \stackrel{def}{=} (a_2, delay).Medium_0$$

$$Medium_9 \stackrel{def}{=} (coll, delay).Medium_0$$

$$Senders \stackrel{def}{=} Sender_{10} \underset{L_3}{\bowtie} Medium_0 \underset{L_4}{\bowtie} Sender_{20}$$

$$Timer_{10} \underset{L_5}{\bowtie} Senders \underset{L_6}{\bowtie} Timer_{20}$$

where $L_3 = \{rts_1, data_1, c_1, a_1, coll\}$, $L_4 = \{rts_2, data_2, c_2, a_2, coll\}$, $L_5 = \{r_1, d_1, cts_1, ack_1\}$, $L_6 = \{r_2, d_2, cts_2, ack_2\}$. It is important to note that this model does not include all the behaviour of the IEEE 802.11 protocol. Even at this fairly high level of abstraction it is far from complete. In particular it is missing the timeout mechanism employed by the sender to see if the CTS has arrived within a given limit. It would be a simple matter to add an Erlang timeout in the manner described earlier in this chapter, as follows.

$$Timer_i(0) \stackrel{def}{=} (timeout_i, t).Timer_i(N) + (c_i, \top).Timer_i(N) + (coll, \top).Timer_i(N)$$

$$Timer_i(j) \stackrel{def}{=} (tick_i, t).Timer_i(j-1) + (c_i, \top).Timer_i(N) + (coll, \top).Timer_i(N)$$

$$: \text{for } 1 \leq j < N$$

$$Timer_i(N) \stackrel{def}{=} (rts_i, \top).Timer_i(N-1)$$

$$: \text{for } i = 1, 2$$

$$Sender_{i2} \stackrel{def}{=} (c_i, \top).Sender_{i3} + (coll, \top).Sender_{i4} + (timeout_i, \top).Sender_{i4}$$

$$Sender_{i6} \stackrel{def}{=} (c_i, \top).Sender_{i3} + (coll, \top).Sender_{i4} + (timeout_i, \top).Sender_{i4}$$

$$: \text{for } i = 1, 2$$

$$Sender_1 \stackrel{def}{=} Sender_{10} \underset{L_7}{\bowtie} Timer_1(N)$$

$$Sender_2 \stackrel{def}{=} Sender_{20} \underset{L_8}{\bowtie} Timer_2(N)$$

$$Senders \stackrel{def}{=} Sender_1 \underset{L_3}{\bowtie} Medium_0 \underset{L_4}{\bowtie} Sender_2$$

$$Timer_{10} \underset{L_5}{\bowtie} Senders \underset{L_6}{\bowtie} Timer_{20}$$

where $L_7 = \{rts_1, timeout_1, c_1, coll\}$, $L_8 = \{rts_2, timeout_2, c_2, coll\}$ and all other components remain as previously specified. N is the number of phases in the Erlang distribution used to approximate the deterministic timeout.

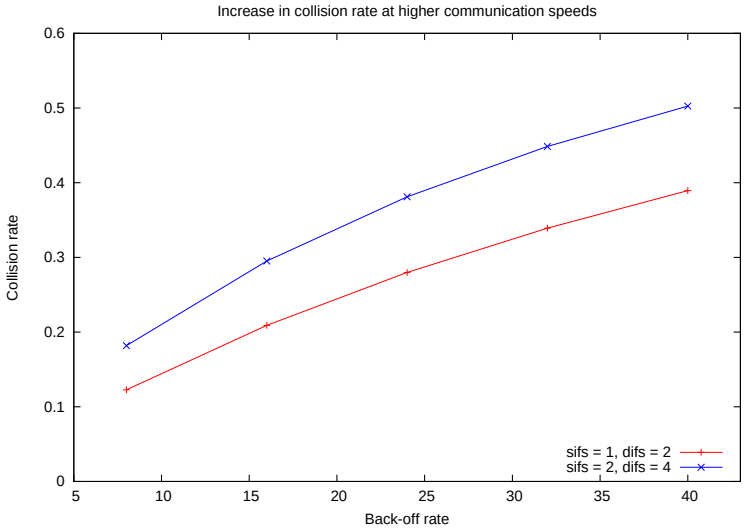

Fig. 6.1. Rate of collision varied against the back-off rate ($nav = 10$, $datato = 1$ and $delay = 1$).

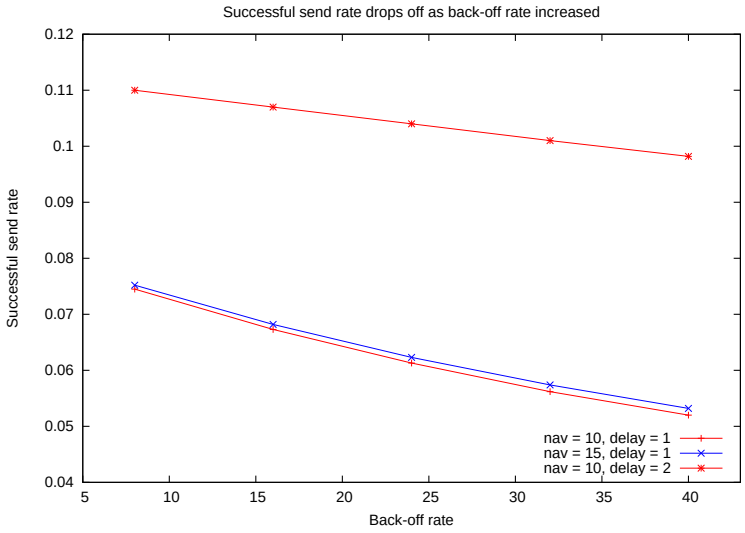

Fig. 6.2. Successful send rate varied against the back-off rate ($datato = 1$, $sifs = 1$ and $difs = 2$).

6.1. Analysis. In [3], results for the *collision probability* were presented. The authors define this as the probability that the medium is in state $Medium_9$.[2] In fact, this is more accurately termed the average rate of collisions, as is shown in Fig. 6.1. The parameters used in these figures are based on those in [3].

Fig. 6.1 shows the rate of collision varied against the back-off rate for two different

[2]In the model, as originally presented in [3], this is not an entirely accurate definition as the medium would be blocked until the collision is detected by both senders, rather this defines the *collision rate*.

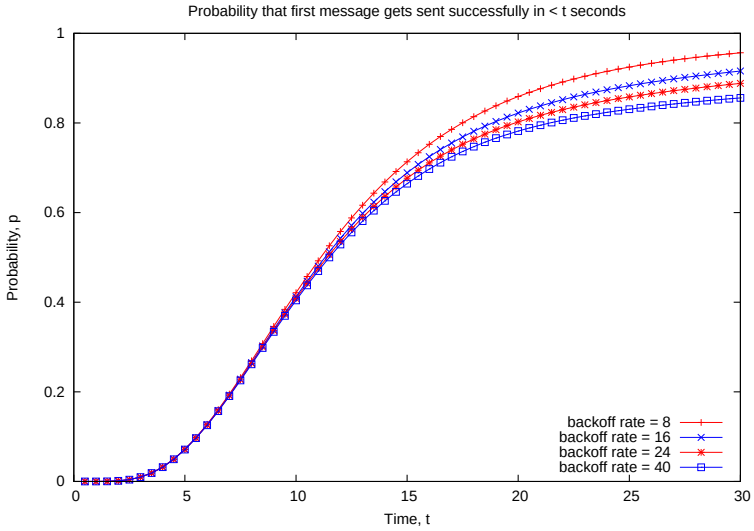

Fig. 6.3. Cumulative response time distribution of time that first message gets sent successfully, as back-off rate, *backoff*, is varied between 8 and 40.

transmission rates. The larger the back-off rate (corresponding to a shorter back-off period), the greater the collision rate. This is due to the increased probability of successive collisions if the back-off period is too short, i.e. the senders are both going to try again quickly and will therefore more likely collide again. The faster communication ($difs = 4$, $sifs = 2$) also gives rise to an increase in the collision rate. Naively this may seem to be counter-intuitive, as a quick communication means there is less time when the transmission can be interrupted by the other sender. However, in this model the senders are continuously going through a sending cycle. The consequence for the model is that the amount of time the sender spends in non-conflicting behaviour decreases, whereas the amount of time the sender spends dealing with each collision remains the same. Hence, the proportion of time spent in collision related behaviour actually increases.

The success rate is controlled by the ability of the sender to send its *rts* and for this to be delivered. This corresponds to the rate at which requests to send are transmitted (nav) and the network latency ($delay$). Intuitively, if nav increases (for the same latency) then there will be more collisions, as the sender is waiting less time to check if the network is quiescent. Conversely if the latency is decreased (essentially if the senders and receivers are closer together) then one would expect the collision rate to fall. This is evident in Fig. 6.2, which shows the rate of successful message transmission for different values of request rate (nav) and latency ($delay$).

Clearly the change in latency has a greater impact than the change in request rate. This is due to the latency having a greater role in the model (it occurs in many places) and the fact that the request rate is already significantly faster than any other rate, and so increasing it has less overall effect. In fact, increasing the request rate increases the successful send rate slightly. This is because the medium will be less likely to be idle (a request is more likely to be made) and so more messages are able to be processed.

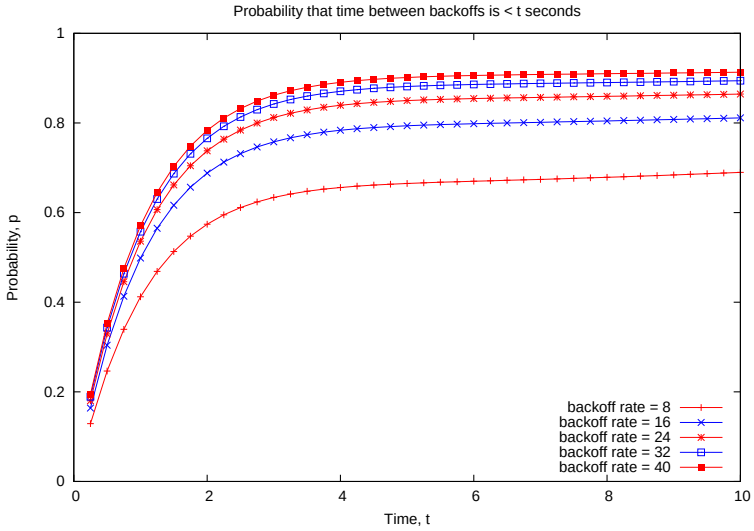

Fig. 6.4. Cumulative response time distribution of time between consecutive back-off events, as back-off rate, *backoff*, is varied between 8 and 40.

Fig. 6.3 and Fig. 6.4 show response time analysis as generated by the ipc/HYDRA toolkit. A particularly powerful aspect of this style of analysis is that probabilities from these results can be used as quality-of-service metrics in a commercial setting. In this case we also augment the response time CDF with a parameter sweep of the *backoff* parameter space.

In Fig. 6.3, we plot the time to first message being successfully sent. We see that as we increase the *backoff* parameter from 8 to 40, we get a consistent worsening in the quality of the response time – that is, as rate of back-off increases, the probability that we will have completed a successful first message transmission decreases.

In Fig. 6.4, we plot the time between consecutive back-off events. This time, as we increase the *backoff* parameter from 8 to 40, we see a consistent decrease in the time between back-off events, with the probability of having seen a back-off event by 10 seconds increasing from 0.6 to 0.9.

7. Conclusions. To gain the maximum possible benefit from the opportunities for new interaction experiences offered by consumer devices equipped for wireless communication the software and hardware architecture which powers that communication capability must be engineered to provide the performance demanded. Performance modelling is not the study of the obvious: there are many situations where replacing hardware components with faster devices can reduce the performance measure of greatest importance. For this reason it is essential to comprehensively investigate the performance of the system via quantitative modelling.

High-level modelling languages allow models to be concise and powerful modelling tools allow analysis to be swift. We have surveyed in the present chapter a range of modelling tools for high-level modelling languages based on Performance Evaluation Process Algebra and presented applications of these languages in the context of

wireless communications protocols. In the evaluation of the models we have used a range of strong modelling tools which help modellers to detect errors in their models and after debugging to help to securely compute analysis results which deliver rich insights into the functioning of the system under study.

Acknowledgements. Ashok Argent-Katwala and Jeremy Bradley are supported in part by EPSRC under the PerformDB grant, EP/D054087/1. Stephen Gilmore and Nil Geisweiller are supported by the SENSORIA project (EU FET-IST Global Computing 2 project 016004).

REFERENCES

[1] I. Mitrani, *Probabilistic Modelling*. Cambridge University Press, 1998.
[2] "PEPA resource site." `http://www.dcs.ed.ac.uk/pepa/`.
[3] K. N. Sridhar and G. Ciobanu, "Describing IEEE 802.11 wireless mechanisms by using the π-calculus and Performance Evaluation Process Algebra," in *EPEW'04, Proceedings of the European Performance Evaluation Workshop* (M. T. Nunez et al., ed.), vol. 3236 of *Lecture Notes in Computer Science*, pp. 233–247, Springer-Verlag, October 2004.
[4] J. Hillston, *A Compositional Approach to Performance Modelling*, vol. 12 of *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1996.
[5] H. Bowman, J. W. Bryans, and J. Derrick, "Analysis of a multimedia stream using stochastic process algebras," *The Computer Journal*, vol. 44, no. 4, pp. 230–245, 2001.
[6] J. Forneau, L. Kloul, and F. Valois, "Performance modelling of hierarchical cellular networks using PEPA," *Performance Evaluation*, vol. 50, pp. 83–99, Nov. 2002.
[7] N. Thomas, J. T. Bradley, and W. J. Knottenbelt, "Stochastic analysis of scheduling strategies in a grid-based resource model," *IEE Software Engineering*, vol. 151, pp. 232–239, September 2004.
[8] D. R. W. Holton, "A PEPA specification of an industrial production cell," in Gilmore and Hillston [42], pp. 542–551.
[9] J. T. Bradley, N. J. Dingle, S. T. Gilmore, and W. J. Knottenbelt, "Derivation of passage-time densities in PEPA models using ipc: the Imperial PEPA Compiler," in *MASCOTS'03, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems* (G. Kotsis, ed.), (University of Central Florida), pp. 344–351, IEEE Computer Society Press, October 2003.
[10] W. Grassman, "Means and variances of time averages in Markovian environments," *European Journal of Operational Research*, vol. 31, no. 1, pp. 132–139, 1987.
[11] A. Reibman and K. S. Trivedi, "Numerical transient analysis of Markov models," *Computers and Operations Research*, vol. 15, no. 1, pp. 19–36, 1988.
[12] N. Thomas and J. Hillston, "Using a Markovian process algebra to specify interactions in queueing systems," LFCS Report ECS–LFCS–97–373, Department of Computer Science, University of Edinburgh, Edinburgh EH9 3JZ, UK, 1997.
[13] M. Bernardo, L. Donatiello, and R. Gorrieri, "Describing queueing systems with MPA," Tech. Rep. UBLCS-94-11, University of Bologna, 1994.
[14] M. Bhabuta, P. Harrison, and K. Kanani, "Detecting reversibility in Markovian process algebra," in *Performance Engineering of Computer and Telecommunication Systems*, (Springer Verlag), 1995.
[15] J. Hillston and N. Thomas, "A syntactical analysis of reversible PEPA models," in Priami [43], pp. 37–49.
[16] P. G. Harrison and J. Hillston, "Exploiting quasi-reversible structures in Markovian process algebra models," in Gilmore and Hillston [42], pp. 510–520.
[17] N. Thomas and S. Gilmore, "Applying quasi-separability to Markovian process algebras," in Priami [43], pp. 27–36.
[18] M. Sereno, "Towards a product form solution for stochastic process algebras," in Gilmore and Hillston [42], pp. 622–632.
[19] J. Hillston, "Fluid flow approximation of PEPA models," in *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, (Torino), pp. 33–42, IEEE Computer Society Press, September 2005.
[20] S. Ramsey, D. Orrell, and H. Bolouri, "Dizzy: stochastic simulation of large-scale genetic regula-

tory networks," *Journal of Bioinformatics and Computational Biology*, vol. 3, pp. 415–436, 2005.

[21] J. T. Bradley, "Semi-Markov PEPA: Modelling with generally distributed actions," *International Journal of Simulation*, vol. 6, pp. 43–51, January 2005.

[22] M. O. Soren Asmussen, Olle Nerman, "Fitting phase-type distribution via the EM algorithm," *Scandinavian Journal of Statistics*, vol. 23, pp. 419–441, 1996.

[23] N. Geisweiller, "Fitting phase type distributions within PEPA model," in *Process Algebra and Stochastically Timed Activities*, 2005.

[24] N. Geisweiller, "Finding the most likely values inside a PEPA model according to partially observable executions," tech. rep., 2006.

[25] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. Roy. Soc. Ser. B.*, vol. 39, pp. 1–38, 1977.

[26] C. Wu, "On the convergence properties of the EM algorithm," *Ann. Statist.*, vol. 11, pp. 95–103, 1983.

[27] L. Rabiner and B. Juang, *Fundamentals of Speech Recognition*. Prentice Hall Signal processing Series, 1993.

[28] C. M. Bishop, *Neural Networks for Pattern Recognition*. Clarendon Press, Oxford, 1995.

[29] S. Ikeda, "Acceleration of the EM algorithm," *Systems and Computers in Japan, John Wiley & Sons, Inc.*, vol. 31, no. 2, pp. 10–18, 2000.

[30] S. Gilmore and J. Hillston, "The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling," in *Proceedings of the Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, no. 794 in Lecture Notes in Computer Science, (Vienna), pp. 353–368, Springer-Verlag, May 1994.

[31] J. T. Bradley and W. J. Knottenbelt, "The ipc/HYDRA tool chain for the analysis of pepa models," in *QEST'04, Proceedings of the 1st IEEE Conference on the Quantitative Evaluation of Systems* (B. Haverkort et al., ed.), (University of Twente, Enschede), pp. 334–335, IEEE Computer Society Press, September 2004.

[32] S. Gilmore and J. Hillston, "The PEPA workbench: A tool to support a process algebra-based approach to performance modelling," in *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation* (G. Haring and G. Kotsis, eds.), vol. 794 of *Lecture Notes in Computer Science*, pp. 353–368, Springer-Verlag, Vienna, May 1994.

[33] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," in *TACAS'02, Proceedings of Tools and Algorithms for Construction and Analysis of Systems*, vol. 2280 of *Lecture Notes in Computer Science*, (Grenoble), pp. 52–66, Springer-Verlag, April 2002.

[34] G. Clark and W. Sanders, "Implementing a stochastic process algebra within the Möbius modeling framework," in *Proceedings of the first joint PAPM-PROBMIV Workshop* (L. de Alfaro and S. Gilmore, eds.), vol. 2165 of *Lecture Notes in Computer Science*, (Aachen, Germany), pp. 200–215, Springer-Verlag, Sept. 2001.

[35] N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison, "HYDRA: HYpergraph-based Distributed Response-time Analyser," in *PDPTA'03, Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications* (H. R. Arabnia and Y. Man, eds.), vol. 1, (Las Vegas, NV), pp. 215–219, June 2003.

[36] A. Argent-Katwala, J. T. Bradley, and N. J. Dingle, "Expressing performance requirements using regular expressions to specify stochastic probes over process algebra models," in *WOSP 2004, Proceedings of the 4th International Workshop on Software and Performance* (V. Almeida and D. Lea, eds.), (Redwood City, California), pp. 49–58, ACM, January 2004.

[37] A. Argent-Katwala and J. T. Bradley, "Functional performance specification with stochastic probes," in *EPEW'06, Proceedings of the 3rd European Performance Evaluation Workshop* (M. Telek, ed.), vol. 4054 of *Lecture Notes in Computer Science*, (Budapest), pp. 31–46, Springer-Verlag, June 2006.

[38] S. Gilmore, J. Hillston, and G. Clark, "Specifying performance measures for PEPA," in *Proceedings of the 5th International AMAST Workshop on Real-Time and Probabilistic Systems*, vol. 1601 of *Lecture Notes in Computer Science*, (Bamberg), pp. 211–227, Springer-Verlag, 1999.

[39] P. Harrison, "Turning back time in Markovian process algebra," *Theoretical Computer Science*, no. 290, pp. 1947–1968, 2003.

[40] P. Harrison, "Reversed Processes, product forms and some non-product forms," *Linear Algebra and Its Applications*, pp. 359–381, 2004.

[41] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: Probabilistic symbolic model checker,"

in *Computer Performance Evaluation / TOOLS*, pp. 200–204, 2002.

[42] S. Gilmore and J. Hillston, eds., *PAPM'95, Proceedings of the 3rd International Workshop on Process Algebra and Performance Modelling*, vol. 38(7) of *Special Issue: The Computer Journal*, CEPIS, Edinburgh, June 1995. ISSN 0010–4620.

[43] C. Priami, ed., *PAPM'98, Proceedings of the 6th International Workshop on Process Algebra and Performance Modelling*, Università Degli Studi di Verona, Nice, September 1998.