university of durham

# Reliability Modelling in Software with Random Processes

*Jeremy Bradley and Nigel Thomas*

# Contents

- Motivation
- Randomness in system models
- Reliability with randomly timed execution
- Examples
- Conclusions



In this presentation, we are looking at how we might model software with stochastic processes and in particular stochastic process algebras (SPAs). We look at how random (stochastic) events can legitimately occur in software modelling and where approximation is necessary.

We present several possible definitions for measuring reliability over and above the standard definitions of performance in stochastic systems. Subsequently we demonstrate two examples of software systems and show how they can be manipulated to allow us to generate reliability information from them.

# Motivation

- Process algebra
  - Component–wise decomposition
- Reliability in random behaviour
  - Improve over mean–based previous metrics
  - Solve livelock problems
  - Tail–end behaviours
    - NOT possible with simulation
  - Asking the right questions to improve system knowledge



Our motivation for using process algebras to model software systems is that they offer a conceptually simple component decomposition with the emphasis on the inter–component communication. This gives us the ability to express a system succinctly within the algebra and then manipulate it to create abstractions and equivalent–functioning systems. Also crucially, a description can be augmented with performance and time information.

We look at the issue of reliability within random behaviour as previously the emphasis has concentrated on mean–based metrics, such as steady–state distributions and rewards. This would be fine accept that they offer no concept of how such systems vary over short time–scales and in many modelling situations there is a specific requirement to be able to ensure that an event happens within a certain period of time (as one example of what we mean when we talk about reliability). In contrast, a steady–state result could only give you a long time–scale proportion of how long the system spent doing a particular event.

By attributing random distributions to events in systems we are giving ourselves modelling freedom to represent rare events, for example, if a system event occurs very infrequently after a very short period of time. Importantly combinations of such rare events might put the system in a failure mode which would not be observable through simulation as they might only occur after many years or decades of simulation time.

## This is not Fault–tolerance

- **We model with:**
  - **Uncertain execution–time**
  - **Probabilistic program branching**
    - **e.g. conditional statements**
      - **if...then...else**
      - **while (...) {...}**
- **Not just faulty components**

wospSept2000

As has been eluded to, we add stochastic events to the execution times of actions within the process algebra description. At a basic level of description we can also have probabilistic branching at branch points of programs, e.g. If–then–else and while statements. This is one of the instances of approximation within the paradigm since a static probability is unlikely to represent the complete behaviour of a general while–loop.

Further enhancements could be made to have time–based or loop–based probabilities which would do a better modelling job – but these are beyond the scope of this presentation.

Fault–tolerance can be modelled within the framework but the paradigm is more than the traditional fault–tolerance techniques of the 1980s.

## How might randomness occur in software modelling?

- Statement: software is mostly deterministic
- Uncertain knowledge of execution
  - High level representation
  - Genuine stochastic event
    - user input
    - random seeding
    - hardware accessing
  - Because we put it there!



So how might software have random properties?

Many people assume that software execution is mainly a deterministic affair and at a very low level this used to be true, but with cache paging algorithms, predictive pipelining and such, even at a low level of execution, commands do not necessarily have a completely and unerringly deterministic (constant) execution time. At an application level virtual memory paging, pre–emptive multi–tasking, JIT compilers and garbage–collection add to the mess and ensure that something as high level as a line of Java may take between 0.1 and 0.5ms to execute according to the state of the system its running on.

It would be an overly complex task to model the entire state of the processor and operating system (and even virtual–machine), so even at a very basic level it may be necessary to express lines of code as having  stochastic properties (if your modelling requirement demands it, e.g. a safety–critical application).

What does a random execution time represent? It might represent the fact that we do not know how long something might take and so we use a probability distribution to model and allow us to reason with our incomplete knowledge.

It may be that an event is truly random in any case, for instance the time it takes for a user to respond with necessary user input, or the time it takes for a machine to retrieve a piece of data from a hard disk.

As we will see later a software event may be random because we injected deliberate randomness into the program!

## Random over deterministic events

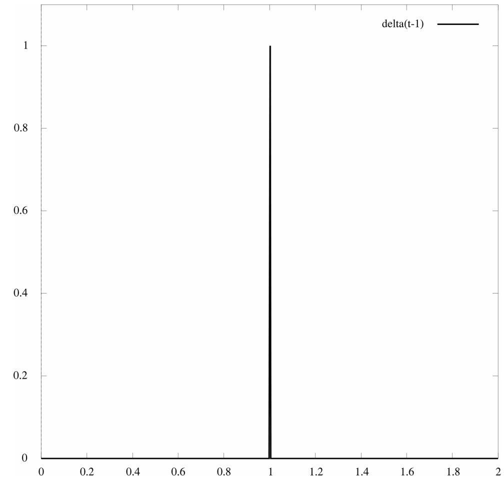
- **Deterministic**
  - Clock cycles
- **Random**
  - User key stokes
  - Mouse clicks
  - Web requests on web server



See previous slide notes for explanation.

# Random over deterministic events (II)

- Clock pulses



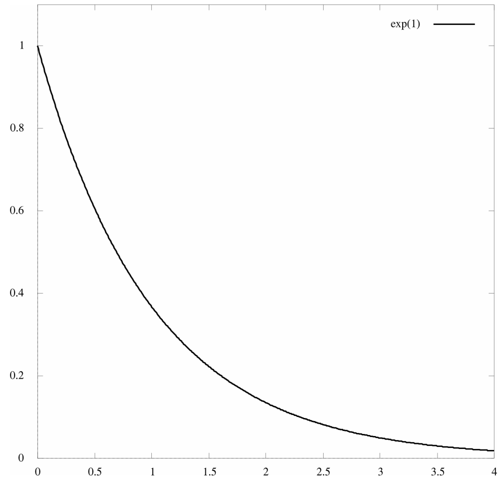



The fact that we are modelling with stochastic events does not preclude us being able to talk about deterministically timed actions if we need to. A deterministic event is modelled by a Dirac−delta function in a continuous space and it satisfies all the usual probabilistic conditions of a probability distribution.

The above graph represents an event that occurs at time t=1 with probability 1.

## Random over deterministic events (III)

- Web page accesses



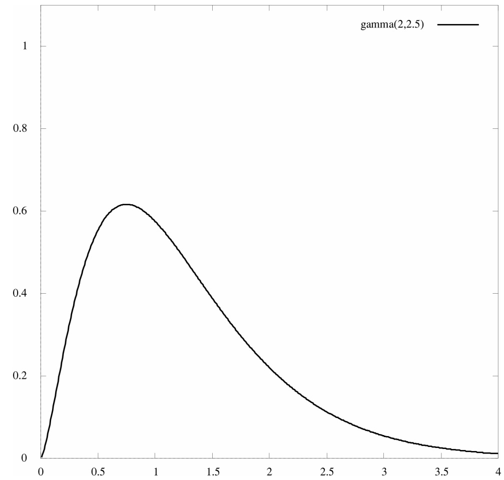



We might represent the time between web page accesses on a web server as having an exponential distribution (as above).

Later, when we talk about a system being Markovian, we mean that all the events are assumed to have an exponentially timed distribution.

# Random over deterministic events (IV)

- User key strokes




wospSept2000

The time between a user's key strokes might have a slightly different distribution which allows for the fact that a user cannot type in excess of a certain speed.

The above distribution is a Gamma distribution.

## Simple While–loop Example

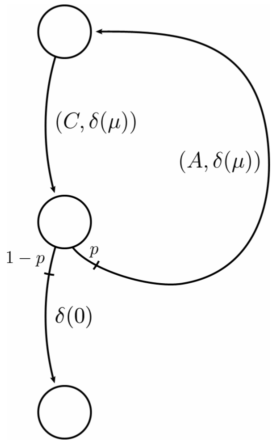
- **Code**
  - **while (C) {**
    **A;**
    **}**
    **exit(0);**
- **Process algebra**

$$
\begin{aligned}
Test1 &\stackrel{\text{def}}{=} (\mathrm{C}, \delta(\mu)).Test2 \\
Test2 &\stackrel{\text{def}}{=} [1-p]0 + [p]While \\
While &\stackrel{\text{def}}{=} (\mathrm{A}, \delta(\mu)).Test1
\end{aligned}
$$



We present a model of a while–loop which has a conditional test C and a set of instructions A that it will continually execute if C is true. We model the test on C as having a probability of p of success and assume that it takes exactly mu seconds to perform the test C and mu seconds for the instruction set A.

# While−loop Example (II)






A stochastic transition graph of the while−loop example.

## While–loop Example (III)

- Randomness inserted in the number of times the system cycles
- Overall distribution approximates an exponential evolution

wospSept2000

The randomness in this system is not explicitly in the execution time of the events, as we have modelled these with deterministic distributions. The randomness exists solely in the probabilistic branch at the point that the condition C gets tested for its truth value.

If we were to represent this while loop as a single aggregated transition, we would have to represent the fact that it might take 0 or mu or 2mu or 3mu, etc. to execute. This itself is representable as a single distribution. In fact in the limit it has exponential properties and could be well approximated by an exponential event.

## What is reliability within a stochastic system then?

- **Reliability can loosely be described as the amount of variation from the mean a particular system might have**
- **This can be represented by several metrics:**
  - **Variance**
  - **$\mathrm{Prob}(a<X<b)$**
  - **$\mathrm{Prob}(X>a)$**



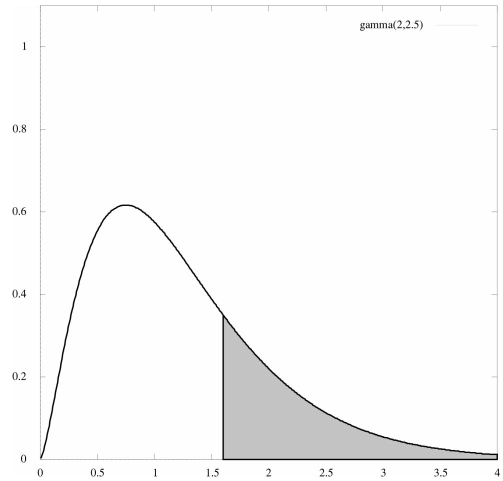
Now we turn to the issue of reliability within stochastic systems.

We might simply use the standard probability measure of variance to represent a notion of reliability. However we might like to be more specific and ask questions like: what’s the probability that an event occurs between two times (2–tailed test) or what’s the probability that a key event takes longer than a specific period (1–tailed test).

As we will see in a later example the key event may not directly be represented by the action we are studying, it may in fact be the distribution of time that the remainder of the system takes to return to perform the action.

## Stochastic Reliability (II)

- **Prob(X>a)**

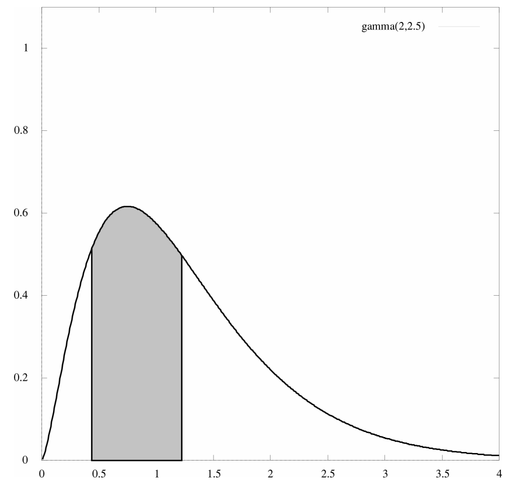



wospSept2000

A 1–tailed test as represented on a distribution graph.

# Stochastic Reliability (III)

- $\text{Prob}(a<X<b)$




wospSept2000

A 2−tailed test.

# Network Communication Example

- **Extracting a reliability measure from an MPA model**

$$
\begin{aligned}
Transmitter &\stackrel{\text{def}}{=} (\text{transmit}, \lambda_1).(\text{t-recover}, \lambda_2).Transmitter \\
Receiver &\stackrel{\text{def}}{=} (\text{receive}, \top).(\text{r-recover}, \mu).Receiver \\
Network &\stackrel{\text{def}}{=} (\text{transmit}, \top).(\text{delay}, \nu_1).(\text{receive}, \nu_2).Network \\
System &\stackrel{\text{def}}{=} (Transmitter || Receiver) \underset{\{\text{transmit,receive}\}}{\bowtie} Network
\end{aligned}
$$

wospSept2000

The above is a PEPA (a Markovian process algebra – which assumes exponential distributions) model of a Transmitter/Receiver system running over a network.

The parameters which proceed the action name represent the parameter of the exponential distribution which that action has attributed to it. Where a top symbol appears, this represents a passive action which will wait for its active counterpart (in another component) to complete before continuing.

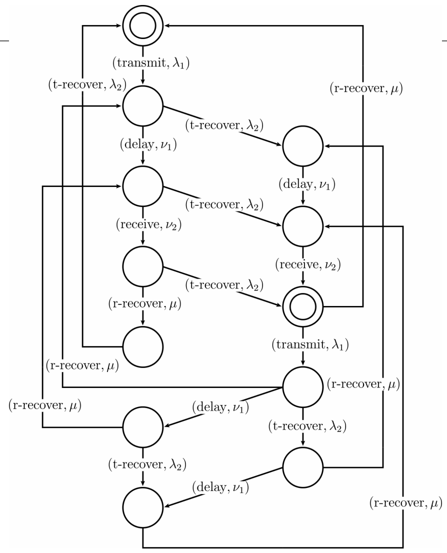
The Transmitter component performs a transmit and then a t–recover (say calculating checksums) before looping to perform a transmit again.

A Receiver waits for the network to send a receive then it goes into an r–recover before looping.

The Network waits for the transmit from the Transmitter, introduces a delay before finally sending the receive action.

We are going to concentrate on the transmit action.

# Network Example (II)






The global state space of the Network example. The highlighted states (concentric circles) represent the states that the system is about to perform a transmit from.

## Network Example (III)

- *Traditional analysis would solve Markov chain:*
  - *For $\lambda_1 = 1$, $\lambda_2 = \mu = 1/4$, $\nu_1 = 1/10$, $\nu_2 = 10$*
    - *P(1) = 0.0225, P(8) = 0.0556*
    - 
- *Apply reward vector*
  - *Get long–term average figure*



In a traditional analysis of such a system, we might assign figures to the parameters of the distributions (as above) and obtain the steady–state figures for being in states 1 and 8 (the states that the system would wait in to send the transmit token). We could then apply a reward vector and obtain a long–term average statistic for how long the system spent in the transmit position.

# Network Example (IV)

$X_1 \sim \exp(\lambda_1)$

$X_2 \sim \exp(\nu_1 + \lambda_2)$

$X_3 \sim \exp(\nu_2 + \lambda_2)$

$X_4 \sim \exp(\mu + \lambda_2)$

$X_5 \sim \exp(\lambda_2)$

$X_6 \sim \exp(\lambda_1 + \mu)$

$X_7 \sim \exp(\lambda_2 + \nu_1 + \mu)$

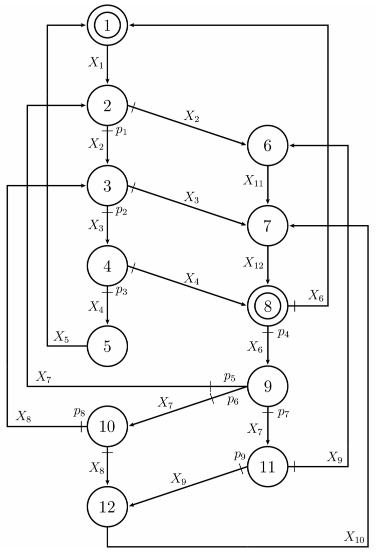
$X_8 \sim \exp(\lambda_2 + \mu)$

$X_9 \sim \exp(\nu_1 + \mu)$

$X_{10} \sim \exp(\mu)$

$X_{11} \sim \exp(\nu_1)$

$X_{12} \sim \exp(\nu_2)$




$p_1 = \frac{\nu_1}{\nu_1+\lambda_2}$

$p_2 = \frac{\nu_2}{\nu_2+\lambda_2}$

$p_3 = \frac{\mu}{\mu+\lambda_2}$

$p_4 = \frac{\lambda_1}{\lambda_1+\mu}$

$p_5 = \frac{\mu}{\lambda_2+\nu_1+\mu}$

$p_6 = \frac{\nu_1}{\lambda_2+\nu_1+\mu}$

$p_7 = \frac{\lambda_2}{\lambda_2+\nu_1+\mu}$

$p_8 = \frac{\mu}{\lambda_2+\mu}$

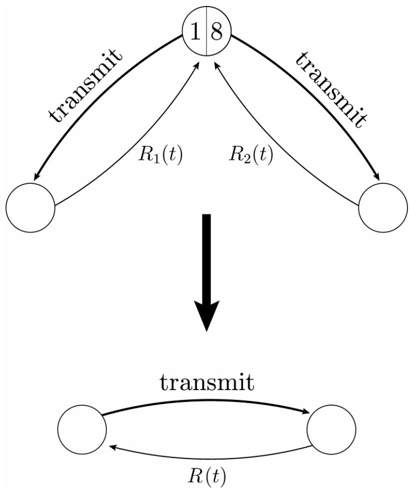
$p_9 = \frac{\nu_1}{\nu_1+\mu}$



In our reliability analysis (see references for details), we transform the Markovian system into a stochastic transition system, with full probabilistic branching and transform the distributions similarly. We can now aggregate the parts of the system (using passage–time distribution calculations) we are not interested in to obtain a reduced system which concentrates on the transmit actions.

# Network Example (V)

- *Require $F_R(t)$*
- *Prob(system recovers for retransmit in < 15ms) = 0.7332*
- *If packet size = 4Kbits*
  - *Prob(bandwidth > 250Kbits) = 0.7332*






The first diagram represents a combined 1/8 state which has the two possible transmit evolutions of the global state space with two aggregate corresponding return distributions representing the recovery of the rest of the system after each of the transmits.

This can in turn be transformed into a single cycle system with a single return distribution R(t). If we calculate $F_R(t)$, we can use our 1–tailed reliability test to find out the probability that the system recovers in less than 15ms (using the parameter values of slide 18). Given that we know how long the transmit itself takes we can give a reliability figure (i.e. not just an average) for the bandwidth, as above.

Of further use would be the general quantity Prob(bandwidth>t), which would yield a distribution for how the bandwidth can vary with time.

## Injecting deliberate random timing

- **Why?**
  - **If Markovian systems are so easy to reason about – why not make systems Markovian!**
  - **Expressiveness – lets us be honest when events really are random.**
- **More generally**
  - **Making systems that we are better able to reason about**



Finally we get to the point eluded to earlier: why might we want to inject random timing into a system?

As computer scientists we are in the enviable position of also being able to control the "universe" we model. We can write or modify the software, protocols or hardware, if necessary, that our models describe. The idea is that we insist, as part of the design criteria of a system, that individual execution elements of a system (at a particular level) have exponentially timed duration. One way of achieving this is by inserting random delay into the execution pattern of a program so that the individual elements might statistically appear to be exponential. What does this give us as modellers? It gives us the ability to construct a truly accurate model in a paradigm that we can reason about.

If the benefit we would get from better understanding the operation of a system (especially with issues like reliability of certain events) outweighs the usual requirement for raw performance then this may well be a possible way forward in designing parts of a complex system.

# Conclusions

- **Improve knowledge of a system**
- **Aggregation and abstraction techniques reduce conceptual complexity as well as computational complexity**
- **Describe rare–events and effects**

wospSept2000

In summary, reliability modelling is all about improving our understanding of a system's operation.

When considering random or deterministic timing, we can use powerful aggregation techniques to create higher–level abstractions which reduce conceptual and computational complexity.

Finally, we have demonstrated how reliability modelling can take into account rare events which standard metrics tend to "smooth" over.

# References


- **(Bohnenkamp, H C and Haverkort, B R)**
  - *Stochastic Event Structures for the Decomposition of Stochastic Process Algebra Models, pages 25–39, PAPM'99*
- **(Bradley, J T)**
  - *Towards Reliable Modelling with Stochastic Process Algebras, PhD Thesis, Department of Computer Science, University of Bristol, October 1999*
- **(Bradley, J T and Davies, N J)**
  - *Measuring Improved Reliability in Stochastic Systems, pages 121–130, UKPEW'99*

## References


- **(Bradley, J T and Davies, N J)**
  - *A Matrix–based Method for Analysing Stochastic Process Algebras, pages 579–590, PAPM 2000*
- **(Cox, D R and Miller, H D)**
  - *The Theory of Stochastic Processes, Methuen, 1965*
- **(Hillston, J)**
  - *A Compositional Approach to Performance Modelling, Cambridge University Press, 1996*