Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models

Jeremy T. Bradley[*] Stephen T. Gilmore[†] Jane Hillston[†]

April 30, 2007

Abstract

Internet worms are classically described using SIR models and simulations, to capture the massive dynamics of the system. Here we are able to generate a differential equation-based model of infection based solely on the underlying process description of the infection agent model. Thus, rather than craft a differential equation model directly, we derive this representation automatically from a high-level process model expressed in the PEPA process algebra. This extends existing population infection dynamics models of internet worms by explicitly using frequency-based spread of infection. Three types of worm attack are analysed which are differentiated by the nature of recovery from infection and vulnerability to subsequent attacks.

To perform this analysis we make use of continuous state-space approximation, a recent breakthrough in the analysis of massively parallel stochastic process algebra models. Previous explicit state-representation techniques can only analyse systems of order 10^9 states, whereas continuous state-space approximation can allow analysis of models of 10^{10000} states and beyond.

1 Introduction

Internet worms are malicious programs that exploit operating system security weaknesses to propagate themselves over the internet. While the security flaws go unpatched, the worm spreads epidemic-like and uses large

[*]Department of Computing, Imperial College London, 180 Queen's Gate, London SW7 2BZ, United Kingdom. Email: jb@doc.ic.ac.uk

[†]Laboratory for Foundations of Computer Science, The University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom. Email: {Stephen.Gilmore,Jane.Hillston}@ed.ac.uk

amounts of available bandwidth. It has been shown, however, that far more destructive is the worms' effect on the internet routing infrastructure [16], as the worms tend to overload the connecting routers with nonexistent IP lookups.

In recent years, worms like Nimbda, Slammer, Code Red, Sasser [21] and Code Red 2 have caused the internet to become unusable for many hours at a time until security patches can be applied and routers fixed. Models of such infections which try to capture explicitly the overall behaviour of routing protocols as well as the infection behaviour of the worm itself run the risk of exploring an unfeasibly large state-space.

To counteract this explosion, several papers [15, 14] have used hybrid, or multi-scale modelling techniques, which use ordinary differential equations (ODEs) to capture the macro-scale infection dynamics of the worm, and explicit state-space representation to pick out the interaction of a single router, say, within such an environment.

We propose using a stochastic process algebra model to describe individual worm behaviour and using a continuous state-space approximation to provide insight into the emergent behaviour of the worm as it spreads to thousands and then hundreds-of-thousands of hosts.

In order to perform this analysis, we present a method for analysing huge stochastic process algebra models. By transforming a stochastic process algebra model into a set of ODEs, we can obtain a plot of model behaviour against time for models with global state spaces in excess of 10^{10000} states. Existing explicit state-based methods for calculating steady-state, transient or passage-time measures are limited to state-spaces of the order of 10^9.

In order to capture the mass infection dynamics in a population, it is common to construct a set of ODEs which capture the global behaviour of the infection. For an epidemic, these ODEs can be based around the standard SIR model which represents the numbers of susceptible, infective and removed individuals within a population. We extend the standard SIR model to take into account the *frequency-based* nature of infection[1]. More significantly, our modified SIR worm models are derived from the process model of an individual worm, rather than by empirical observation of global dynamics.

In this paper, we analyse three types of worm attack: an SIR over a network model where host inoculation prevents a host from being reinfected; an SIR over a network model where host inoculation is reversible (i.e. a host can be made insecure by a subsequent improperly applied security patch);

[1]Infection as spread through an intermediary medium, in this case the Internet, as opposed to by direct pairwise host infection.

and a SIR/network model which results in a distributed denial-of-service (DDOS) attack.

The remainder of this paper is organised as follows. We review the related work in Section 2. The use of PEPA descriptions to generate an associated set of ordinary differential equations (ODEs) is described in Section 3. This includes a summary of the previous results presented in [11], followed by a description of a generalisation to this result [4]. This technique is then applied to derive ODE models of internet worm attacks in Section 5, where we show that throttling the backbone bandwidth at an appropriately early stage of an infection can severely impede the spread of infection. The paper finishes with some conclusions in Section 6. The use of the PEPA modelling formalism may be familiar to the reader but for the benefit of those readers who do not know it we include an introduction in Appendix A.

2 Related Work

Modelling internet worm attacks with large scale, continuous state-space models based on ordinary differential equations was recently proposed by Liljenstam *et al.* [15, 14] and Nicol *et al.* [16], and this has been an inspiration for our paper. However, in their approach the population-level system of ODEs must be crafted by hand with simulations used to give individual scale behaviour under such attacks. Using the process algebra description systematically to generate the coupled ODEs allows us to focus the population-level model more closely on the individual behaviours of the elements within the system. Thus we may readily experiment with different reaction strategies and examine their impact on the population level statistics. Moreover, our models are able to capture a frequency-based model of infection spread, in contrast to the population infection dynamics of the standard SIR model. By this we mean that we assume that the rate of spread of infection is based on the frequency of interactions between susceptible and infective agents. The most common alternative is to assume that the rate of spread of infection is based on the density of infective agents within the total population [17]. Since infection is not passed on casually in our case (cf. the spread of the common cold through sneezing) the frequency-based models are more appropriate.

Previously, process algebra models have been used to model disease spread and SIR models in an abstract context [17, 18]. In these models the deterministically timed, synchronous process algebra, WSCCS [20], whereas we use the stochastic process algebra PEPA. Furthermore, in this previous work the underlying mathematical models, *mean field equations* and ODEs, were derived informally. In contrast we have developed a systematic mapping which allows a system of coupled ODEs to be derived automatically from a general PEPA model. This extends previous work by Hillston [11] which established a mapping from a restricted class of PEPA models to ODEs. Other work exploring the relationship between process algebra models and ODE models has been published in [1, 5, 6].

3 ODE Generation from PEPA Models

In this section, we summarise the numerical vector form representation and ODE analysis of PEPA models. In [11], Hillston shows how a class of PEPA models can be analysed using coupled ordinary differential equations (ODEs). We will need an extension of this to capture the dynamics of internet worms.

The insight in [11] is to show that cooperating models of identical components of the form, for example:

$$\underbrace{P \parallel P \parallel \cdots \parallel P}_{n}$$

might be better represented by a vector which describes the *number* of components in a given derivative state. That is to say, suppose P has two other derivative states, P' and P'', in its component description. A triple (v_1, v_2, v_3) could be used to represent there being v_1 components in state P, v_2 in state P' and v_3 in state P'' in the cooperation above. Clearly $v_1 + v_2 + v_3 = n$, the total number of components in the cooperation. The ordering of the derivative states within the expression above makes no difference to the observable behaviour. Thus there is no loss of information in simply counting derivatives in this way rather than recording their relative positions. Moreover it has the effect of reducing the state space representation to an aggregated form (described in [9]) which requires a vector representation of size $|D(P)|$, the number of derivatives of P, rather than one of size n, in the unaggregated form.

In the generic example of a n-processor/m-resource system given in [11]:

$$\begin{aligned}
Proc_0 &\overset{def}{=} (task1, r_1).Proc_1 \\
Proc_1 &\overset{def}{=} (task2, r_2).Proc_0 \\
Res_0 &\overset{def}{=} (task1, r_1).Res_1 \\
Res_1 &\overset{def}{=} (reset, s).Res_0
\end{aligned}$$

$$(Proc_0 \parallel \cdots \parallel Proc_0) \underset{\{task1\}}{\bowtie} (Res_0 \parallel \cdots \parallel Res_0)$$

An aggregate state $((n-1, 1), (m, 0))$ would represent a possible state where there were $n-1$ processor components in state $Proc_0$, one in state $Proc_1$, m resource components in state Res_0, and none in state Res_1.

Hillston [11] further goes on to show how a set of ODEs can be constructed which can approximate the discrete number of components in a given state with a continuous state space approximation. This is not significant for PEPA models with only a few replicated components. However, agent models typically have many thousands or millions or even hundreds of millions of similar components and for this style of modelling, this type of aggregation and analysis is very necessary.

Since much of the analytical value of the formalisation is obtained through numerical or symbolic evaluation of the system of ODEs one might ask why we should not begin our modelling working directly with differential equations, perhaps coded in a numerical evaluation platform such as Matlab. One motivation for this is that we have reasoning apparatus available to us at the process algebra level. We can determine that models are free from deadlock — even for very large state spaces — using the MTBDD-based representation of the state space supported by the PRISM modelling tool. We can use the bisimulation relation of the language to do component-wise simplification of the model, if needed. Finally, because we are modelling in a high-level language it is possible to apply these different numerical evaluation procedures to compute performance results from the same model. For example, we can also evaluate PEPA models by stochastic simulation. This is a freedom which we would not have if we had coded a differential equation-based representation of the model directly in a numerical computing platform such as Matlab.

The original work by Hillston placed a number of structural restrictions on the PEPA models which may be used to generate the continuous state space approximation [11]. These restrictions are listed below.

1. Cooperation within groups of components of the same type is not allowed; i.e. counted component derivatives for the purposes of continuous approximation, must occur within a parallel component group such as:
$$P \parallel P \parallel \cdots \parallel P.$$
2. The cooperation set, L, between interacting groups of components, as in:
$$(P \parallel P \parallel \cdots \parallel P) \underset{L}{\bowtie} (Q \parallel Q \parallel \cdots \parallel Q)$$
is restricted to be the set of common action labels between P and Q.
3. Cooperation, where it does occur, is assumed to be of an active nature with a common view of the rate. For instance α actions that occur at

rate λ can only cooperate with other α actions that occur at the same rate, λ.

4. Each action name must appear only in one derivative within the definition of a sequential component, and must only occur once within that derivative definition.

5. Action hiding is not considered.

In the SIR models which we consider in this paper it is necessary to consider some synchronisations that involve both active and passive partners. In the following section we show how the approach can be extended to models with this feature.

4 Extending Coverage of the PEPA Language

In this section, we extend the previous mapping from PEPA to the numerical vector form, and to a set of ODEs, to allow synchronisations between active and passive actions (that is, restriction (3) above is relaxed). Additionally, with the formulation below, we can also relax the requirement that cooperation is restricted to the common set of actions between two cooperating groups of components, thus dealing with restriction (2).

4.1 Time-based System Equation

As a first step, we introduce a time-based system equation for a PEPA model, which extends the *numerical vector form* of [11]. The state of a PEPA model at time t can be represented by $P(t)$, which has the grammar:

$$P(t) ::= P(t) \underset{L}{\bowtie} P(t) \;\mid\; (N(S,t), \ldots, N(S,t))_{\emptyset}$$

where S is the derivative state of a sequential component. The *derivatives tuple* $(N(S_1,t), \ldots, N(S_{N_s},t))_{\emptyset}$ is used to count the number of derivatives of S in the current state of the cooperation:

$$S \parallel S \parallel \cdots \parallel S$$

At this time, we do not consider action hiding or cooperation between components within the derivatives tuple.

The derivatives tuple uses the function $N(X,t)$ to represent the number of components that are in state X at time t within the environment expressed by the overall PEPA system formula, $P(t)$. There are $N_s = |D(S)|$ elements in this tuple to represent the total number of derivative states of component S, as discussed in Section 3.

The time-based system equation can be systematically generated from the static system equation that is used to specify the initial state of co-operation in a PEPA model. Groups of identical components within an empty cooperation structure are mapped onto the derivatives tuple structure $(N(C_1, t), \ldots, N(C_{N_c}, t))_\emptyset$. Whereas, heterogeneous cooperation is mapped onto the equivalent time-based cooperation structure, $P_1(t) \underset{L}{\bowtie} P_2(t)$, where importantly the cooperation set information is maintained. For example, the system equation from the Processor/Resource model of Section 3:

$$Sys \overset{\mathit{def}}{=} (Proc_0 \mid\mid \cdots \mid\mid Proc_0) \underset{\{task1\}}{\bowtie} (Res_0 \mid\mid \cdots \mid\mid Res_0)$$

would be turned into the following time-based equivalent:

$$\begin{aligned} Sys(t) &\overset{\mathit{def}}{=} P_1(t) \underset{\{task1\}}{\bowtie} P_2(t) \\ P_1(t) &\overset{\mathit{def}}{=} (N(Proc_0, t), N(Proc_1, t))_\emptyset \\ P_2(t) &\overset{\mathit{def}}{=} (N(Res_0, t), N(Res_1, t))_\emptyset \end{aligned}$$

As can be seen we define the state of the model in terms of the local states of the sequential components of the model. The function $\mathcal{I}(P)$ identifies this set of components as follows:

$$\begin{aligned} \mathcal{I}(P_1(t) \underset{L}{\bowtie} P_2(t)) &= \mathcal{I}(P_1(t)) \cup \mathcal{I}(P_2(t)) \\ \mathcal{I}((N(C_1, t), \ldots, N(C_{N_c}, t))_L) &= \{\, C_i \;:\; 1 \le i \le N_c \,\} \end{aligned}$$

By restriction (4) above, for an arbitrary action type α, within each sequential component there is at most one local state enabling an action of type α. We use the function $\mathcal{V}_\alpha$ to identify the derivative which enables the action of type α, if one exists. The set of component derivatives returned by this function will either contain one element or it will be empty.

$$\begin{aligned} \mathcal{V}_\alpha(P_1(t) \underset{L}{\bowtie} P_2(t)) &= \mathcal{V}_\alpha(P_1(t)) \cup \mathcal{V}_\alpha(P_2(t)) \\ \mathcal{V}_\alpha((N(C_1, t), \ldots, N(C_{N_c}, t))_L) &= \{\, C_i \;:\; 1 \le i \le N_c \wedge r_\alpha(C_i) > 0 \,\} \end{aligned}$$

Clearly, $\mathcal{V}_\alpha(P) \subseteq \mathcal{I}(P)$.

4.2 Apparent Rate Calculation

In order to generate a set of ODEs for a more generic PEPA model, we will need to show how the apparent rate function, $r_\alpha(\cdot)$, can be calculated over the time-based system equation, $P(t)$. It is important to understand that this is not a redefinition of PEPA's apparent rate formula, just a use of the existing apparent rate rules over the new representation:

$$r_\alpha(P_1(t) \underset{L}{\bowtie} P_2(t)) = \begin{cases} \min(r_\alpha(P_1(t)), r_\alpha(P_2(t))) & : \text{if } \alpha \in L \\ r_\alpha(P_1(t)) + r_\alpha(P_2(t)) & : \text{if } \alpha \notin L \end{cases}$$

$$\begin{aligned} r_\alpha(\,(N(C_1,t),\ldots,N(C_{N_c},t))_\emptyset) \ &= N(C_i,t)\, r_\alpha(C_i) \\ &\quad \text{where } \mathcal{V}_\alpha((N(C_1,t),\ldots,N(C_{N_c},t))_L) = \{\, C_i \,\} \end{aligned}$$

$$\begin{aligned} r_\alpha(\,(N(C_1,t),\ldots,N(C_{N_c},t))_\emptyset) \ &= 0 \\ &\quad \text{where } \mathcal{V}_\alpha((N(C_1,t),\ldots,N(C_{N_c},t))_L) = \emptyset \end{aligned}$$

We also need to consider a slightly more general rate set from that originally considered [10], to take into account the continuous approximation of the ODE system. For this purpose a PEPA action rate, λ, is drawn from the extended set:

$$\lambda \in \mathbb{R}^+ \cup \{n\top \mid n \in \mathbb{R}, n \geq 0\}$$

where $0\top$ will mean that there are no current passive components capable of performing a particular shared action. Thus for $r_1 \in \mathbb{R}^+$, $r_2 \in \mathbb{R}, r_2 \geq 0$:

$$\min(r_1, r_2\top) = \begin{cases} r_1 & : \text{if } r_2 > 0 \\ 0 & : \text{if } r_2 = 0 \end{cases}$$

Since the new rate set is a strict superset of the original rate definition, this in no way impinges on previous PEPA results.

When it comes to capturing the influence of the passive component on the rate of an interaction in the ODEs we use an indicator function to reflect whether the passive component is ready to participate in the shared action or not.

4.3 Component Rate Calculation

The inclusion of passive actions means that the rate at which a component evolves is not wholly defined by that component. Thus we need a function which records the rate of evolution of a component via actions of a particular type, within a given context. This is the component rate function. It is defined for a sequential component derivative C, within a time-based system equation $P(t)$, over a given action type α:

$$\rho_\alpha(C, P_1(t) \underset{L}{\bowtie} P_2(t)) = \begin{cases} \min(\rho_\alpha(C, P_i(t)), r_\alpha(P_j(t))) & \\ \qquad\qquad\qquad : \text{if } \alpha \in L, C \in \mathcal{V}_\alpha(P_i) & \\ \rho_\alpha(C, P_i(t)) \quad : \text{if } \alpha \notin L, C \in \mathcal{V}_\alpha(P_i) & \\ 0 \qquad\qquad\quad : \text{otherwise} & \end{cases}$$
$$\text{for } 1 \leq i \leq 2 \text{ and } j = 3 - i.$$

$$\rho_\alpha(C_i, (N(C_1, t), \ldots, N(C_{N_c}, t))_\emptyset) = N(C_i, t)\, r_\alpha(C_i)$$

In the first case of $\rho_\alpha(C, P_1(t) \underset{L}{\bowtie} P_2(t))$ the component rate calculation captures the local effect of the cooperation on the component, C. The overall synchronised rate is $\min(\rho_\alpha(C, P_i(t)), r_\alpha(P_j(t)))$ reflecting that the locally enabled action may have to wait for other slower components to perform their version of the same action if the action is to proceed both locally and globally.

The second case $\rho_\alpha(C, P_1(t) \underset{L}{\bowtie} P_2(t))$ shows that if the action type α is not in the cooperation set only this $P_j(t)$ does not impact on the rate of α in a sequential component derivative within $P_i(t)$

For $\rho_\alpha(C_i, (N(C_1, t), \ldots, N(C_{N_c}, t))_\emptyset)$, the component rate relies solely on the product $N(C, t)\, r_a(C)$, since there is no cooperation in operation.

4.4 ODE Generation

We are now in a position to describe how to generate a system of coupled ODEs over a PEPA model. As in [11], let us assume that we have n distinct component types $C_1, C_2, \ldots, C_n$ in our PEPA model M. Each component type C_i has N_i derivatives referenced by C_{ij} for $1 \leq j \leq N_i$.

Using $P(t)$ as the time-based system equation for M and recalling that $N(S, t)$ defines the number of components in state S at time t within the equation $P(t)$, we can write down a small-time approximation representing the change in value of $N(C_{ij}, t)$ within $P(t)$:

$$N(C_{ij}, t + \delta t) - N(C_{ij}, t) = \underbrace{- \sum_{k : C_{ij} \xrightarrow{(\alpha, \cdot)} C_{ik}} \rho_\alpha(C_{ij}, P(t))\, \delta t}_{\text{exit activities}} + \underbrace{\sum_{k : C_{ik} \xrightarrow{(\beta, \cdot)} C_{ij}} \rho_\beta(C_{ik}, P(t))\, \delta t}_{\text{entry activities}} \quad (1)$$

Dividing by δt and letting $\delta t \to 0$ and with $v_{ij}(t) = N(C_{ij}, t)$, we get the

ODE system:

$$\frac{\mathrm{d}v_{ij}(t)}{\mathrm{d}t} = -\sum_{k\,:\,C_{ij}\xrightarrow{(\alpha,\cdot)}C_{ik}} \rho_\alpha(C_{ij}, P(t)) \;\; + \sum_{k\,:\,C_{ik}\xrightarrow{(\beta,\cdot)}C_{ij}} \rho_\beta(C_{ik}, P(t)) \tag{2}$$

for $j = 1, 2, \ldots, N_i$ and $i = 1, 2, \ldots, n$. From our discussion of the component rate function, we know that $\rho_\alpha(C, P(t)) = f(v_{ij}(t) \; : \; 1 \leq j \leq N_i, 1 \leq i \leq n)$ for some f which will encapsulate the structure of cooperation over the action α within the system in question.

4.5 Worked Example

In this section, we demonstrate ODE generation over a process algebra model using a simple Resource-Processor example. It is often useful to have PEPA components synchronise passively in cooperations, which can be studied with the extensions presented in Section 4. In this example we have processors synchronising passively with the resources, i.e. they wait for the resources to return their data. We describe this scenario in the model:

$$\begin{aligned} \mathit{Proc}_0 &\stackrel{\mathit{def}}{=} (\mathit{task}1, \top).\mathit{Proc}_1 \\ \mathit{Proc}_1 &\stackrel{\mathit{def}}{=} (\mathit{task}2, r_2).\mathit{Proc}_0 \\ \mathit{Res}_0 &\stackrel{\mathit{def}}{=} (\mathit{task}1, r_1).\mathit{Res}_1 \\ \mathit{Res}_1 &\stackrel{\mathit{def}}{=} (\mathit{reset}, s).\mathit{Res}_0 \end{aligned}$$

$$\mathit{Sys} \stackrel{\mathit{def}}{=} (\underbrace{\mathit{Proc}_0 \mid\mid \cdots \mid\mid \mathit{Proc}_0}_{100}) \underset{\{\mathit{task}1\}}{\bowtie} (\underbrace{\mathit{Res}_0 \mid\mid \cdots \mid\mid \mathit{Res}_0}_{40})$$

In the above model, a processor synchronises with a resource on a *task*1 action before it can independently perform a *task*2 action and return to its original Proc_0 state. The resource also synchronises on the *task*1 action before being able to reset and become Res_0 once more. We configure the initial system with 100 Proc_0 components cooperating with 40 Res_0 components.

We first need to create a time-based system equation for the system before we can employ the ODE generation methods. By letting:

$$\begin{aligned} P_1(t) &= (N(\mathit{Proc}_0, t), N(\mathit{Proc}_1, t))_\emptyset \\ P_2(t) &= (N(\mathit{Res}_0, t), N(\mathit{Res}_1, t))_\emptyset \end{aligned}$$

we can define a time-based version of *Sys* to be:

$$\mathit{Sys}(t) = P_1(t) \underset{\{\mathit{task}1\}}{\bowtie} P_2(t)$$

which represents the number of different types of component in P_1 and P_2 at time, t. We also use the component mapping:

$$\begin{array}{ll} C_{11} \to \mathit{Proc}_0 & C_{21} \to \mathit{Res}_0 \\ C_{12} \to \mathit{Proc}_1 & C_{22} \to \mathit{Res}_1 \end{array}$$

for $v_{ij} = N(C_{ij}, t)$. We are now in a position to apply Eq. (2) and generate the following system of equations:

$$\begin{aligned}
\frac{\mathrm{d}v_{11}(t)}{\mathrm{d}t} &= -\rho_{task1}(\mathit{Proc}_0, \mathit{Sys}(t)) + \rho_{task2}(\mathit{Proc}_1, \mathit{Sys}(t)) \\
&= -r_{task1}(\mathit{Sys}(t)) + \rho_{task2}(\mathit{Proc}_1, P_1(t)) \\
&= -\min(r_{task1}(P_1(t)), r_{task1}(P_2(t))) \\
&\quad + r_{task2}(\mathit{Proc}_1) N(\mathit{Proc}_1, t) \\
&= -\min(v_{11}(t)\top, v_{21}(t) r_1) + r_2 v_{12}(t) \\
&= -r_1 I_{11}(t) v_{21}(t) + r_2 v_{12}(t) \\
\frac{\mathrm{d}v_{12}(t)}{\mathrm{d}t} &= -\rho_{task2}(\mathit{Proc}_1, \mathit{Sys}(t)) + \rho_{task1}(\mathit{Proc}_0, \mathit{Sys}(t)) \\
&= r_1 I_{11}(t) v_{21}(t) - r_2 v_{12}(t) \\
\frac{\mathrm{d}v_{21}(t)}{\mathrm{d}t} &= -\rho_{task1}(\mathit{Res}_0, \mathit{Sys}(t)) + \rho_{reset}(\mathit{Res}_1, \mathit{Sys}(t)) \\
&= -\min(v_{11}(t)\top, v_{21}(t) r_1) + s v_{22}(t) \\
&= -r_1 I_{11}(t) v_{21}(t) + s v_{22}(t) \\
\frac{\mathrm{d}v_{22}(t)}{\mathrm{d}t} &= -\rho_{reset}(\mathit{Res}_1, \mathit{Sys}(t)) + \rho_{task1}(\mathit{Res}_0, \mathit{Sys}(t)) \\
&= r_1 I_{11}(t) v_{21}(t) - s v_{22}(t)
\end{aligned}$$

where $I_{11}(t)$ is a specific indicator function on the value of $v_{11}(t)$ defined in general as follows:

$$I_{ij}(t) = \begin{cases} 1 & : v_{ij}(t) > 0 \\ 0 & : v_{ij}(t) = 0 \end{cases}$$

We demonstrate the ODE solution that is obtained from such a system in Fig. 1. This graph shows the number of Proc_0 and Res_0 components that exist in the system at time t. In this example, $r_1 = 4.0$, $r_2 = 2.8$, $s = 1.0$.

We compared the time taken to solve the system of ODEs with the time taken to solve the Markov chain representation for steady-state. We used the PRISM probabilistic model checker [13] to solve the Markov chain representation. As the number of copies of the processor and the resource grow the state space of the system doubles every time that another processor or resource is added. This cannot continue much past ten processors and ten resources (2^{20} states).

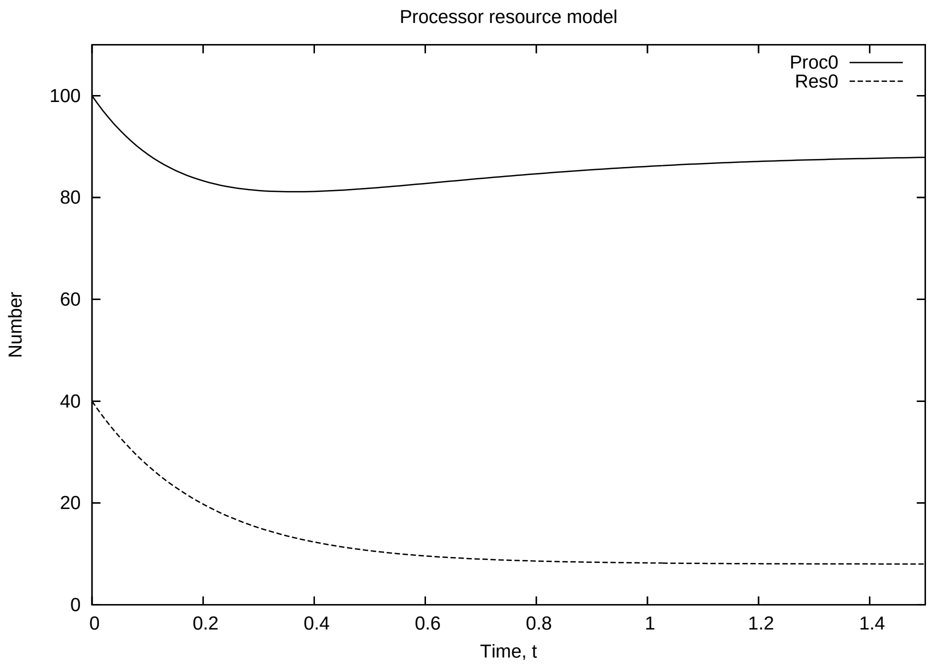

Figure 1: An example ODE solution for the passive processor/resource model

The ODE integrator which we used is a Java implementation of the Dormand-Prince fifth-order Runge-Kutta solver [7]. The version of PRISM which we used is 3.1. The runtimes which are reported are elapsed (wall clock) times as reported by GNU time version 1.7. All timings were made on a 1.60GHz Pentium IV with 1Gb RAM running Fedora Core 5 Linux, averaged over a number of runs. These repeated runs are simply to allow us to compute average timings to tell us about the efficiency of our analysis procedure: the results from each run of the Runge-Kutta integrator are identical.

Processors	Resources	States	Time to solve CTMC (secs)	Time to integrate ODEs (secs)
10	10	2^{20}	237.58	2.36
20	10	2^{30}	–	2.42
20	20	2^{40}	–	2.40
100	100	2^{200}	–	2.42
1000	1000	2^{2000}	–	2.46
10000	10000	2^{20000}	–	2.46

It is important to understand what the ODE solution represents compared to say, traditional transient analysis of a Markov chain: an ODE represents a deterministic view of a system, that is, a particular mean trajectory. This compares to a transient Markov model solution which maintains the stochastic information in the solution and shows a particular trajectory's probability of occurring at a time t.

Specifically, Fig. 1 shows that initially, the populations of Proc_0 and Res_0 decrease together as they synchronise with each other. As the number of Proc_0 components exceeds the number of Res_0 components initially, there is always a high contention for Res_0 components to synchronise with. After plateauing at around 80 competing Proc_0 components, a steady state of around 90 is reached. This high contention has the effect of suppressing the population of Res_0 components to a steady-state value less than 10.

This does not mean that it is not possible for more that 10 resources to ever perform a *reset* after a certain length of time. Rather this means that, in the steady state, there is a large demand for the resources and those that do become available above 10 are quickly absorbed by the population of 90 or so unsynchronised Proc_0 components.

5 Models of Internet Worms

In this section, we apply a version of an SIR model of infection to various computer worm attack models. An SIR model explicitly represents the total number of **susceptible**, **infective** and **removed** hosts in a system and is more commonly used to model disease epidemics. The following shows the standard deterministic model for an SIR infection, s is the number of susceptibles, i, the number of infectives and r, the size of the removed population at time t:

$$\begin{aligned} \frac{\mathrm{d}s(t)}{\mathrm{d}t} &= -\beta s(t)i(t) \\ \frac{\mathrm{d}i(t)}{\mathrm{d}t} &= \beta s(t)i(t) - \gamma i(t) \\ \frac{\mathrm{d}r(t)}{\mathrm{d}t} &= -\gamma i(t) \end{aligned} \qquad (3)$$

where β is the infection parameter and γ is the removal parameter. It is usual when coming up with tailored versions of the SIR model, to modify the ODEs directly to capture any individual requirements. For instance in Liljenstam *et al.* [14], they capture spatial differences in worm infection in a network by using m^2 β-parameters to specify the infection rates between m autonomous systems.

Rather than directly modify the ODEs, we take the different (and unusual) approach of letting an underlying process description dictate the overall structure of the deterministic model. We consider:

1. Basic SIR over a network
2. SIR with reinfection over a network
3. SIR model with second phase distributed denial-of-service attack

In all three cases, the transmission of an infection has to occur over a possibly congested network. This model, where a transmission population (in this case the available ability to route traffic in the network) is used as a conduit for infection, is termed a *frequency-based* infection and is distinct from the more usual *density-based* assumption of the standard SIR model, Eq. (3). Again, it is common for such a distinction to be factored into the governing equations by means of some generalising coefficients.

The benefit of having an explicit representation for the network, is that we can see how the worm infection's progress interferes with network dynamics. It also turns out that manipulating the network availability can be a useful factor in the controlling the spread of infection.

5.1 Susceptible-Infective-Removed model over a Network

This is our most basic infection model and is used to verify that we get recognisable qualitative results.

Initially, there are N susceptible computers and one infected computer. As the system evolves more susceptible computers become infected from the growing infective population. An infected computer can be patched so that it is no longer infected or susceptible to infection. This state is termed *removed* and is an absorbing state for that component in the system.

The capacity of the network is dictated by the parameter N, the number of concurrent, independent connections that the network can sustain. Additionally, an attempted network connection can fail or timeout as indicated by the *fail* action. This might be due to network contention or the lack of availability of a susceptible machine to infect. As large scale worm infections tend not to waste time determining whether a given host is already infected or not, we assume that a certain number of infections will attempt to reinfect hosts; in this instance, the host is unaffected.

$$\begin{aligned}
S &= (\mathit{infectS}, \top).I \\
I &= (\mathit{infectI}, \beta).I + (\mathit{infectS}, \top).I + (\mathit{patch}, \gamma).R \\
R &= \mathit{stop} \\
\mathit{Net} &= (\mathit{infectI}, \top).\mathit{Net}' \\
\mathit{Net}' &= (\mathit{infectS}, \beta).\mathit{Net} + (\mathit{fail}, \delta).\mathit{Net} \\
\mathit{Sys} &= (S[M] \mid\mid I) \underset{L}{\bowtie} \mathit{Net}[N]
\end{aligned}$$

where $L = \{\mathit{infectI}, \mathit{infectS}\}$. Throughout this section, we use the shorthand $C[N]$ to represent the parallel composition of N components of type C:

$$\underbrace{C \mid\mid C \mid\mid \cdots \mid\mid C}_{N}$$

As with the worked example, we first derive the time-based system equation for this model:

$$\begin{aligned}
\mathit{Sys}(t) &= P_1(t) \underset{L}{\bowtie} P_2(t) \\
P_1(t) &= (N(S,t), N(I,t), N(R,t))_{\emptyset} \\
P_2(t) &= (N(\mathit{Net},t), N(\mathit{Net}',t))_{\emptyset}
\end{aligned}$$

We use the following mapping for Eq. (2):

$$\begin{array}{ll}
C_{11} \to S & C_{21} \to \mathit{Net} \\
C_{12} \to I & C_{22} \to \mathit{Net}' \\
C_{13} \to R &
\end{array}$$

The ODE system for the SI-over-Network model is:

$$\begin{aligned}
\frac{\mathrm{d}v_{11}(t)}{\mathrm{d}t} &= -\beta I_{11}(t) v_{22}(t) \\
\frac{\mathrm{d}v_{12}(t)}{\mathrm{d}t} &= -\gamma v_{12}(t) + \beta I_{11}(t) v_{22}(t) \\
\frac{\mathrm{d}v_{13}(t)}{\mathrm{d}t} &= \gamma v_{12}(t) \\
\frac{\mathrm{d}v_{21}(t)}{\mathrm{d}t} &= -\beta I_{21}(t) v_{12}(t) + \beta I_{11}(t) v_{22}(t) \\
&\quad + \beta I_{12}(t) v_{22}(t) + \delta v_{22}(t) \\
\frac{\mathrm{d}v_{22}(t)}{\mathrm{d}t} &= -\beta I_{11}(t) v_{22}(t) - \beta I_{12}(t) v_{22}(t) - \delta v_{22}(t) \\
&\quad + \beta I_{21}(t) v_{12}(t)
\end{aligned}$$

where as before $v_{ij}(t) = N(C_{ij}, t)$.

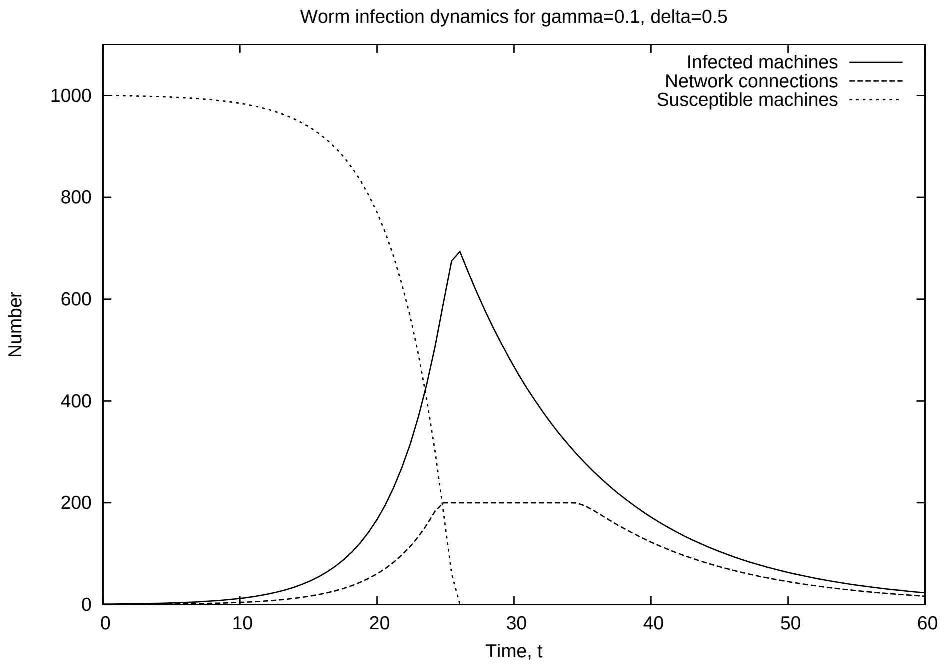

Figure 2: Plot of infectives, susceptibles and network connections over time for patch rate, $\gamma = 0.1$ and network connection failure rate, $\delta = 0.5$.

In Figs. 2–4, we take a susceptible population of $M = 1000$ computers and an interconnect network capable of sustaining up to $N = 200$ simultaneous independent connections.

Fig. 2 and Fig. 3 show the different population dynamics for different values of the patch parameter, γ. In Fig. 2, we see that the network reaches saturation as a direct result of infection growth.

Fig. 4 shows how the number of infected machines varies as the patch rate is increased. As we might expect for a higher patch rate, the infection takes longer to take hold and then does not peak at the sort of values as seen from a, possibly more realistic, patch rate. With a patch rate of 1/10th that of the infection rate, we see the infection peak earlier at nearly 80% of the susceptible population.

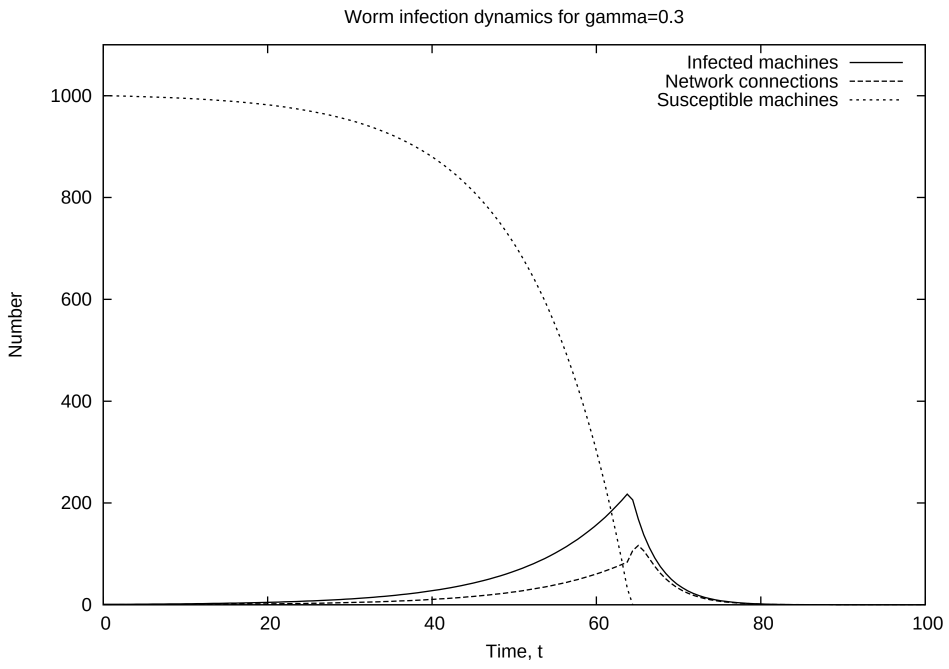

Figure 3: Plot of infectives, susceptibles and network connections over time for patch rate, $\gamma = 0.3$

5.2 Susceptible-Infective-Removed-Reinfection model over a Network

The Susceptible-Infective-Removed-Reinfection (SIRR) model is set out below. As with the SIR model of Section 5.1, we constrain infection to occur over a limited network resource, constrained by the number of independent network connections in the system, N. A small modification in the process model of infection allows for removed computers to become susceptible again after a delay. We use this to model a faulty or incomplete security upgrade or the mistaken removal of security patches which had previously defended the machine against attack.

$$\begin{aligned}
S &= (\mathit{infectS}, \top).I \\
I &= (\mathit{infectI}, \beta).I + (\mathit{infectS}, \top).I + (\mathit{patch}, \gamma).R \\
R &= (\mathit{unsecure}, \mu).S \\
\mathit{Net} &= (\mathit{infectI}, \top).\mathit{Net}' \\
\mathit{Net}' &= (\mathit{infectS}, \beta).\mathit{Net} + (\mathit{fail}, \delta).\mathit{Net} \\
\mathit{Sys} &= (S[1000] \mid\mid I) \underset{L}{\bowtie} \mathit{Net}[N]
\end{aligned}$$

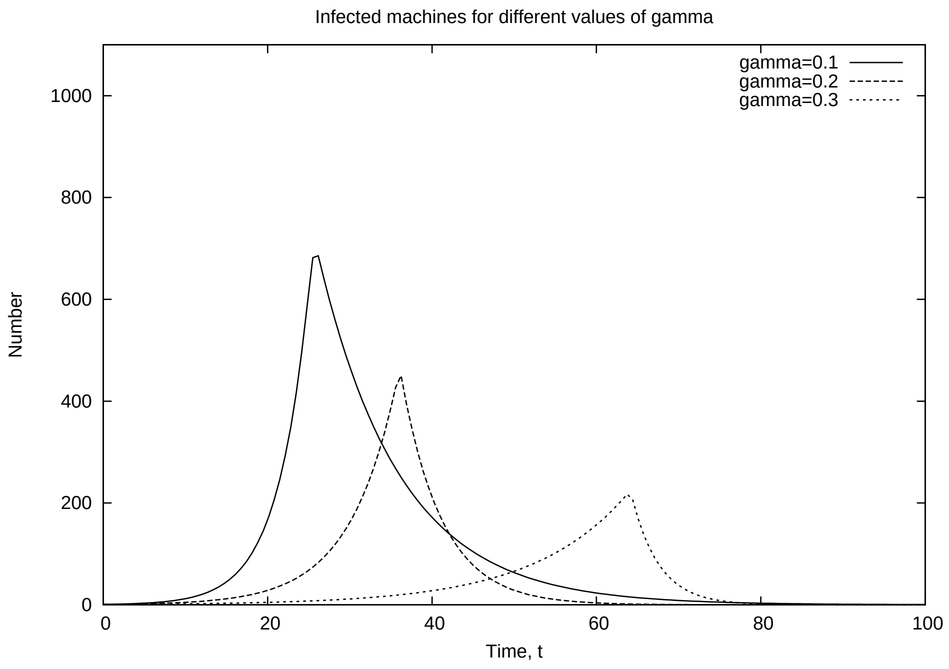

Figure 4: Plot of number of infected machines as patch rate γ is increased from 0.1 to 0.3

where $L = \{infectI, infectS\}$.

The definition of the quantities $Sys(t)$, $P_1(t)$, $P_2(t)$ and $v_{ij}(t)$ remain unaltered with respect to the model since the number of components has not changed.

The modified ODEs, generated from the new model, are presented below:

$$\begin{aligned}
\frac{\mathrm{d}v_{11}(t)}{\mathrm{d}t} &= -\beta I_{11}(t)v_{22}(t) + \mu v_{13}(t) \\
\frac{\mathrm{d}v_{12}(t)}{\mathrm{d}t} &= -\gamma v_{12}(t) + \beta I_{11}(t)v_{22}(t) \\
\frac{\mathrm{d}v_{13}(t)}{\mathrm{d}t} &= -\mu v_{13}(t) + \gamma v_{12}(t) \\
\frac{\mathrm{d}v_{21}(t)}{\mathrm{d}t} &= -\beta I_{21}(t)v_{12}(t) + \beta I_{11}(t)v_{22}(t) \\
&\quad +\beta I_{12}(t)v_{22}(t) + \delta v_{22}(t) \\
\frac{\mathrm{d}v_{22}(t)}{\mathrm{d}t} &= -\beta I_{11}(t)v_{22}(t) - \beta I_{12}(t)v_{22}(t) - \delta v_{22}(t) \\
&\quad +\beta I_{21}(t)v_{12}(t)
\end{aligned}$$

where as before $v_{ij}(t) = N(C_{ij}, t)$.

Solutions of the ODEs for the SIRR system gives rise to Figs. 5, 6 and 7. As a result of the addition of the possibility of a patched machine becoming susceptible once more, the infection level does not die away to zero but instead a residual level remains in the system.

Here, we vary the network capacity N of the interconnecting network. We observe that a reduced network capacity in Figs. 6 and 7 prevents the sharp peak in infection and instead restricts the infection spread to a much slower monotonic rise. Although the ultimate residual infection level in the system is the same (this depends on the ratio of infection rate to patch and unsecure rates), preventing the catastrophic infection growth and peak could be significant in preventing the damage to the router infrastructure seen in recent worm infections.

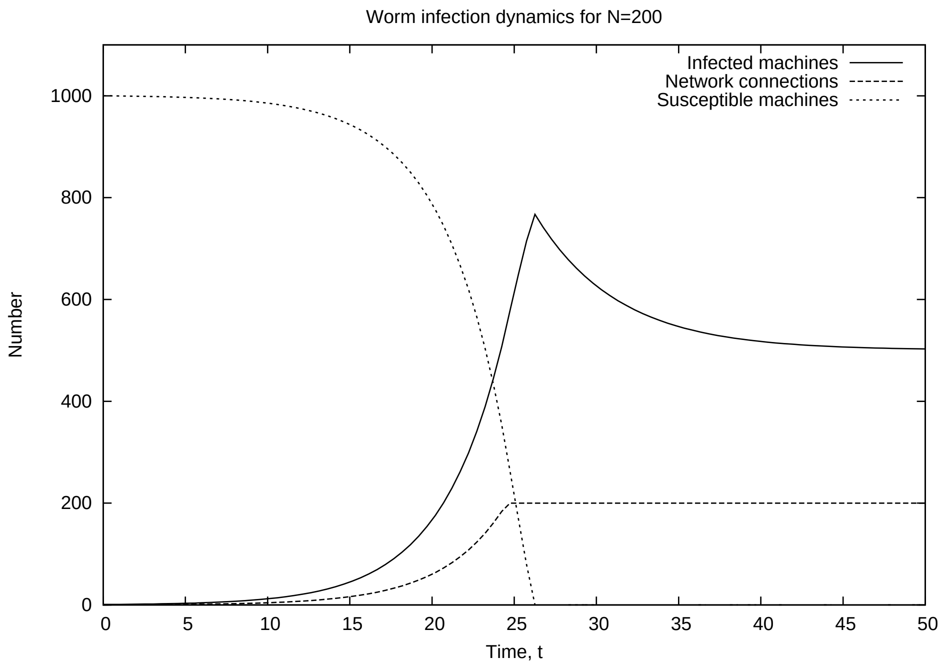

Figure 5: Plot of infection dynamics for unsecured SIR model with network capacity of $N = 200$ channels

5.3 Susceptible-Infective-Removed-Attack model

This example describes a modified SIR-Attack model. This simulates a possible *distributed denial-of-service* (DDOS) attack mode of an Internet

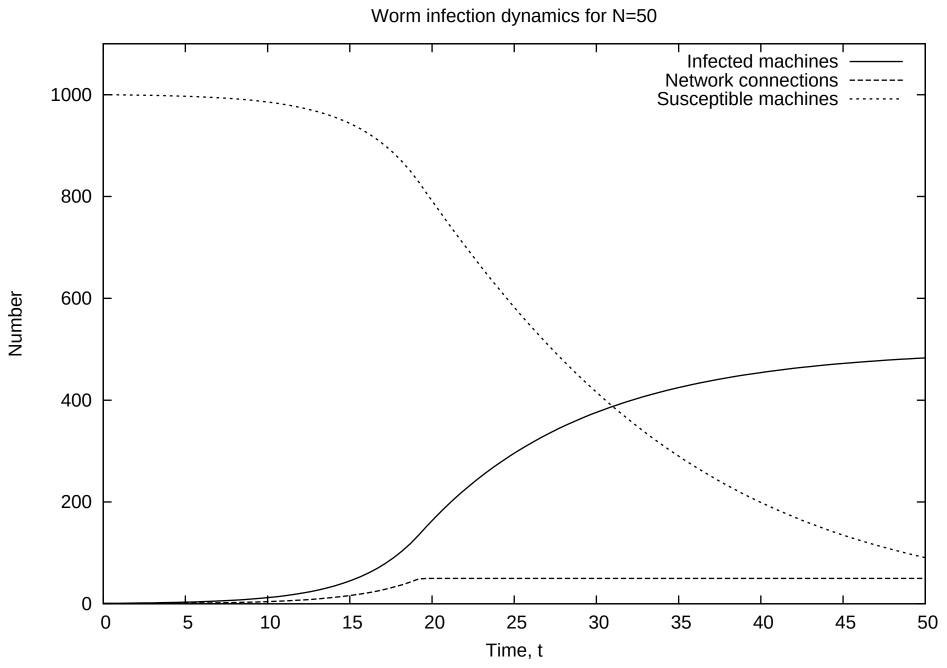

Figure 6: Plot of infection dynamics for unsecured SIR model with network capacity of $N = 50$ channels

worm. In some worms it is known that there is a bimodal behaviour to the worm, either a worm can infect another computer or it can start an attack on a victim computer. The attack need not itself exploit any particular security flaw, but can be something as simple as requesting a specific web page, or issuing a *ping* request. The combination of perhaps millions of machines making such requests quickly overwhelms the target computer, which either crashes under the huge load, or becomes unusably slow.

The PEPA model is extended with a new component to represent the target computer, V, and the behaviour of the network is enhanced to capture

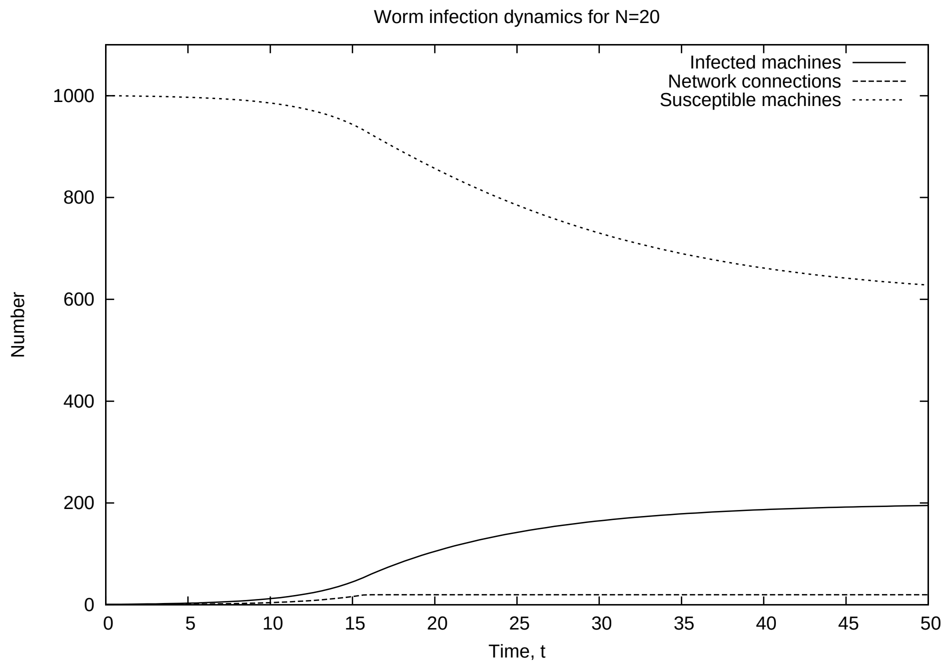

Figure 7: Plot of infection dynamics for unsecured SIR model with network capacity of $N = 20$ channels

the transmission of attacks in addition to infection messages.

$$\begin{aligned}
S &= (\mathit{infectS}, \top).I \\
I &= (\mathit{infectI}, \beta).I + (\mathit{infectS}, \top).I + (\mathit{patch}, \gamma).R \\
&\quad +(\mathit{attack\_mode}, \chi).A \\
A &= (\mathit{attackA}, \lambda).A + (\mathit{patch}, \gamma).R \\
R &= \mathit{stop} \\
\mathit{Net} &= (\mathit{infectI}, \top).\mathit{Net}' + (\mathit{attackA}, \top).\mathit{Net}'' \\
\mathit{Net}' &= (\mathit{infectS}, \beta).\mathit{Net} + (\mathit{fail}, \delta).\mathit{Net} \\
\mathit{Net}'' &= (\mathit{attackV}, \rho).\mathit{Net} + (\mathit{fail}, \delta).\mathit{Net} \\
V &= (\mathit{attackV}, \top).V' \\
V' &= (\mathit{release}, \sigma).V \\
\mathit{Sys} &= (S[1000] \mid\mid I \mid\mid V) \underset{L}{\bowtie} \mathit{Net}[N]
\end{aligned}$$

where $L = \{\mathit{infectI}, \mathit{infectS}, \mathit{attackA}, \mathit{attackV}\}$.

The time-based system equation for this model is:

$$\begin{aligned}
\mathit{System}(t) &= (\mathit{Population}(t) \mid\mid \mathit{Target}(t)) \underset{L}{\bowtie} \mathit{Network}(t) \\
\mathit{Population}(t) &= (N(S,t), N(I,t), N(A,t), N(R,t))_{\emptyset} \\
\mathit{Target}(t) &= (N(V,t), N(V',t))_{\emptyset} \\
\mathit{Network}(t) &= (N(\mathit{Net},t), N(\mathit{Net}',t))_{\emptyset}
\end{aligned}$$

We use the following mapping for Eq. (2):

$$\begin{array}{lll}
C_{11} \to S & C_{21} \to \mathit{Net} & C_{31} \to V \\
C_{12} \to I & C_{22} \to \mathit{Net}' & C_{32} \to V' \\
C_{13} \to A & C_{23} \to \mathit{Net}'' & \\
C_{14} \to R & &
\end{array}$$

The ODE system for the SIR-attack model is:

$$\begin{aligned}
\frac{\mathrm{d}v_{11}(t)}{\mathrm{d}t} &= -\beta I_{11}(t)v_{22}(t) \\
\frac{\mathrm{d}v_{12}(t)}{\mathrm{d}t} &= -\gamma v_{12}(t) - \chi v_{12}(t) + \beta I_{11}(t)v_{22}(t) \\
\frac{\mathrm{d}v_{13}(t)}{\mathrm{d}t} &= -\gamma v_{13}(t) + \chi v_{12}(t) \\
\frac{\mathrm{d}v_{14}(t)}{\mathrm{d}t} &= \gamma(v_{12}(t) + v_{13}(t)) \\
\frac{\mathrm{d}v_{21}(t)}{\mathrm{d}t} &= -\beta I_{21}(t)v_{12}(t) - \lambda v_{13}(t)I_{21}(t) \\
&\quad + \delta(v_{22}(t) + v_{23}(t)) + \rho I_{31}(t)v_{23}(t) \\
&\quad + \beta I_{11}(t)v_{22}(t) + \beta I_{12}(t)v_{22}(t) \\
\frac{\mathrm{d}v_{22}(t)}{\mathrm{d}t} &= -\beta I_{11}(t)v_{22}(t) - \beta I_{12}(t)v_{22}(t) - \delta v_{22}(t) \\
&\quad + \beta I_{21}(t)v_{12}(t) \\
\frac{\mathrm{d}v_{23}(t)}{\mathrm{d}t} &= -\rho I_{31}(t)v_{23}(t) - \delta v_{23}(t) + \lambda I_{21}(t)v_{13}(t) \\
\frac{\mathrm{d}v_{31}(t)}{\mathrm{d}t} &= -\rho I_{31}(t)v_{23}(t) + \sigma v_{32}(t) \\
\frac{\mathrm{d}v_{32}(t)}{\mathrm{d}t} &= -\sigma v_{32}(t) + \rho I_{31}(t)v_{23}(t)
\end{aligned}$$

where as before $v_{ij}(t) = N(C_{ij}, t)$.

Fig. 8, Fig. 9 and Fig. 10 show how an individual machine might fare under a concerted distributed denial-of-service attack from a population of worm infected computers. When the number of network connections of the victim machine is limited to 150 the machine is rapidly overwhelmed and remains so. In contrast, when 500 network connections are available the victim machine retains some capacity to handle connections throughout the attack. At the intermediate level of 250 connections the machine is briefly overwhelmed but recovers moderately quickly. Note that the characteristics of the original attack remains the same in each of the scenarios.

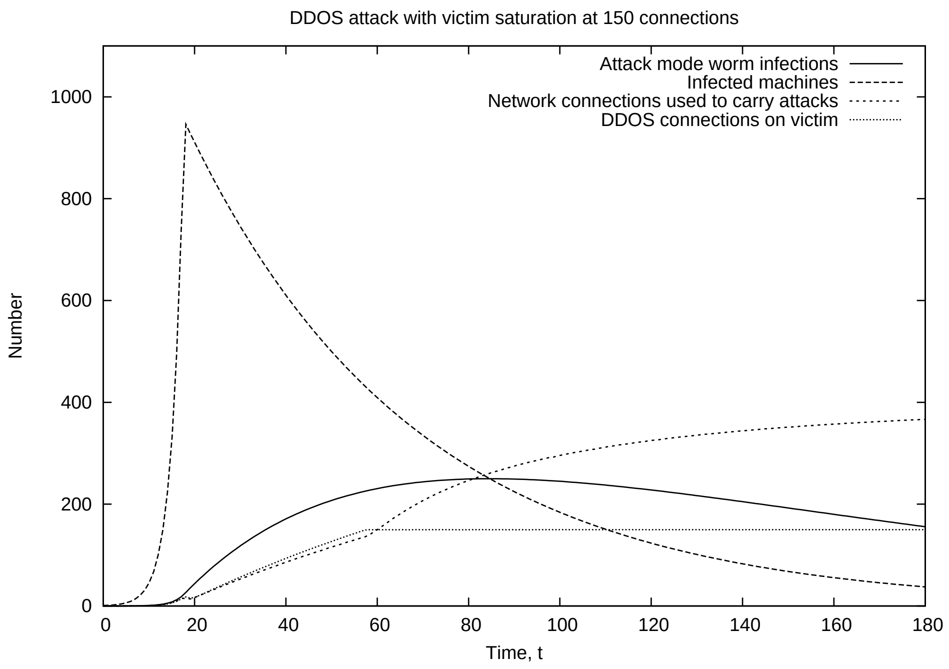

Figure 8: Characteristic of a DDOS attack that overwhelms a victim machine

6 Conclusion

Internet worms are a virtual pestilence. The distributed scale of their effects defeats attempts to model their behaviour in very close detail, and thus impedes the analysis which has the potential to bring understanding of their function and distribution. Large-scale modelling can be effective here, because it abstracts away from modelling of individual behaviour and considers population-based representations. In the present paper we have used a high-level modelling language (the PEPA process algebra) to generate frequency-based population dynamics models of Internet worm spread.

The continuous-space modelling methods which are used in population-based studies are often unfamiliar to those who model the discrete-state systems used in computer networks. By initiating our modelling from a

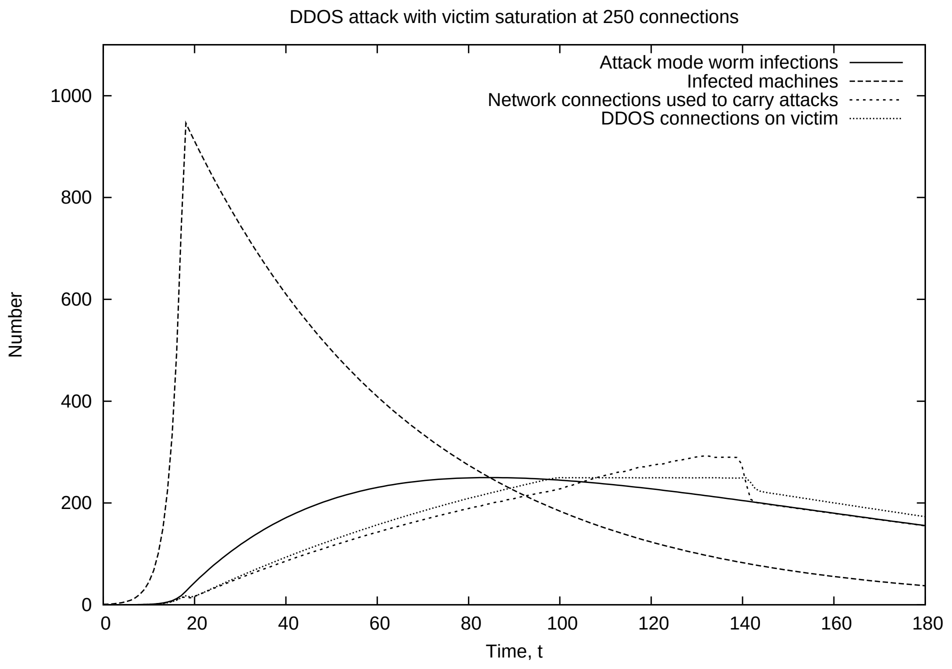

Figure 9: Characteristic of a DDOS attack that briefly incapacitates a victim machine before it recovers

high-level process-based representation rather than from differential equations directly, we hope that we have brought powerful and efficient analysis techniques closer to the practitioners who need to use them.

We have focused our attention on this aspect of the modelling process because numerical integration of systems of differential equation is a fully automatic process which runs without user intervention. Thus, if the process algebra model describes the behaviour of the major system components well then the modeller is well-placed to interpret the results of the time series analysis which results from solving the initial value problem for the ODEs. In practice, we have found this to readily admit an intuitive interpretation.

The scale of problems which can be modelled in this way vastly exceeds those which are founded on explicit state representations. The latter are prone to the well-known problems of state space explosion and cannot be made to scale up to systems of the complexity of the models of internet worms which are presented here. We believe the modelling methods exemplified in the present paper to be generally useful for analysing the behaviour of populations of interacting processes with complex dynamics.

Our plans for future work include the extension of our mapping from

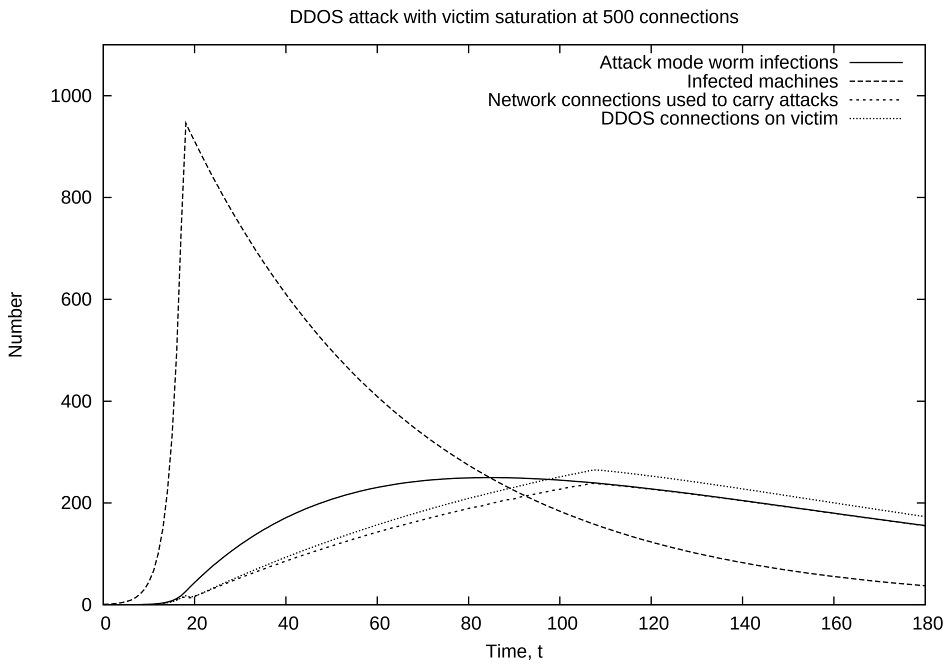

Figure 10: Characteristic of a DDOS attack that does not saturate the victim machine's capacity to handle connections

the PEPA language onto differential equation models. We seek to expand our coverage of the language to include other forms of cooperation across populations.

Acknowledgements

Jeremy Bradley is supported in part by EPSRC under grant PerformDB, EP/054087/01 and GRAIL, EP/505933/01. Jane Hillston's Advanced Research Fellowship is EPSRC award number EP/c543696/01. Stephen Gilmore and Jane Hillston are funded by the ENHANCE grant (EPSRC project GR/S21717/01), and FET-IST GC2 project SENSORIA.

References

[1] L. Bortolussi. *Constraint-based approaches to stochastic dynamics of biological systems.* PhD thesis, University of Udine, 2007.

[2] H. Bowman, J. W. Bryans, and J. Derrick. Analysis of a multimedia stream using stochastic process algebras. *The Computer Journal*, 44(4):230–245, 2001.

[3] 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 G. Kotsis, editor, *MASCOTS'03, Proceedings of the 11th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems*, pages 344–351, University of Central Florida, October 2003. IEEE Computer Society Press.

[4] J. T. Bradley and J. Hillston. Using ODEs and continuous state-space approximation to analyse massively parallel process algebra models. *Theoretical Computer Science*, 2007. (submitted).

[5] M. Calder, S. Gilmore, and J. Hillston. Automatically deriving ODEs from process algebra models of signalling pathways. In G. Plotkin, editor, *Proceedings of Computational Methods in Systems Biology (CMSB 2005)*, pages 204–215, Edinburgh, Scotland, Apr. 2005.

[6] L. Cardelli. From Processes to ODEs by Chemistry. `http://lucacardelli.name/` 22/04/07, 2006.

[7] J. Dormand and P. J. Prince. A family of embedded Runge-Kutta formulae. *Journal of Computational and Applied Mathematics*, 6:19–26, 1980.

[8] J. Forneau, L. Kloul, and F. Valois. Performance modelling of hierarchical cellular networks using PEPA. *Performance Evaluation*, 50(2–3):83–99, Nov. 2002.

[9] S. Gilmore, J. Hillston, and M. Ribaudo. An efficient algorithm for aggregating PEPA models. *IEEE Transactions on Software Engineering*, 27(5):449–464, May 2001.

[10] J. Hillston. *A Compositional Approach to Performance Modelling*, volume 12 of *Distinguished Dissertations in Computer Science.* Cambridge University Press, 1996. ISBN 0 521 57189 8.

[11] J. Hillston. Fluid flow approximation of PEPA models. In *QEST'05, Proceedings of the 2nd International Conference on Quantitative Evaluation of Systems*, pages 33–42, Torino, September 2005. IEEE Computer Society Press.

[12] D. R. W. Holton. A PEPA specification of an industrial production cell. In S. Gilmore and J. Hillston, editors, *Process Algebra and Performance*

Modelling Workshop, volume 38(7) of *Special Issue: The Computer Journal*, pages 542–551. CEPIS, Edinburgh, June 1995.

[13] M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proc. of 12th Int. Conf. on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation*, number 2324 in LNCS, pages 200–204, London, UK, Apr. 2002. Springer-Verlag.

[14] M. Liljenstam, D. M. Nicol, V. H. Berk, and R. S. Gray. Simulating realistic network worm traffic for worm warning system design and testing. In *WORMS'03*, Washington, October 2003. ACM.

[15] M. Liljenstam, Y. Yuan, B. J. Premore, and D. Nicol. A mixed abstraction level simulation model of large-scale internet worm infestations. In *MASCOTS'02, Proceedings of the 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems.* IEEE Computer Society Press, October 2002.

[16] D. Nicol, M. Liljenstam, and J. Liu. Multiscale modeling and simulation of worm effects on the internet routing infrastructure. In *TOOLS'03, Proceedings of Computer Performance Evaluation: Modelling Techniques and Tools*, Lecture Notes in Computer Science, University of Illinois at Urbana-Champaign, September 2003. Springer-Verlag.

[17] R. Norman and C. Shankland. Developing the use of process algebra in the derivation and analysis of mathematical models of infectious disease. In *EUROCAST 2003, 9th International Workshop on Computer Aided Systems Theory*, volume 2809 of *LNCS*, pages 404–414. Springer, 2003.

[18] D. Sumpter. *From Bee to Society: An Agent-based Investigation of Honey Bee Colonies.* PhD thesis, UMIST, 2000.

[19] N. Thomas, J. T. Bradley, and W. J. Knottenbelt. Stochastic analysis of scheduling strategies in a GRID-based resource model. *IEE Software Engineering*, 151(5):232–239, September 2004.

[20] C. Tofts. Describing social insect behaviour using process algebra. *Trans. of the Society of Computer Simulation*, pages 227–283, 1992.

[21] TrendMicro. The Sasser Event: History and Implications. Technical report, `www.trendmicro.com`, June 2004.

A The PEPA Process Algebra

PEPA as a performance modelling formalism has been used to study a wide variety of systems: multimedia applications [2], mobile phone usage [8], GRID scheduling [19], production cell efficiency [12] and web-server clusters [3] amongst others. The definitive reference for the language is [10].

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 is usually interpreted as a random delay which samples from an exponential random variable with parameter 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:

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

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 [10]. The structured operational semantics are shown in Fig. 11. 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 model is to give a component a designated first action using the prefix combinator, denoted by a full stop, which was introduced above. As explained, $(\alpha, r).P$ 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.

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

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

Figure 11: PEPA Structured Operational Semantics

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 $X \overset{def}{=} E$. The name X is in scope in the expression on the right hand side meaning that, for example, $X \overset{def}{=} (\alpha, r).X$ 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 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. 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.* Once enabled, the rate of a shared activity has to be altered to reflect the slower component in a cooperation.

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.

Within the cooperation framework, PEPA respects the definition of *bounded capacity*: that is, a component cannot be made to perform an activity faster by cooperation, so the rate of a shared activity is the minimum of the apparent rates of the activity in the cooperating components.

The total capacity of a component P to carry out activities of type α is termed the *apparent rate* of α in P, denoted $r_\alpha(P)$. It is used heavily when calculating the pairwise cooperation rate: when cooperating with another component, the bounded capacity principle ensures that the overall rate of cooperation does not exceed either of the constituent apparent rates.

To summarise the original ruleset from [10], the apparent rate function can be defined as:

$$r_\alpha(P) = \sum_{P \xrightarrow{(\alpha,\lambda_i)}} \lambda_i$$

where $\lambda_i \in \mathbb{R}^+ \cup \{n\top \mid n \in \mathbb{Q}, n > 0\}$, $n\top$ is shorthand for $n \times \top$ and $\top$ represents the passive action rate that inherits the rate of the coaction from the cooperating component. $\top$ requires the following arithmetic rules:

$$\begin{aligned}
m\top < n\top &\quad : \quad \text{for } m < n \text{ and } m, n \in \mathbb{Q} \\
r < n\top &\quad : \quad \text{for all } r \in \mathbb{R}, n \in \mathbb{Q} \\
m\top + n\top = (m+n)\top &\quad : \quad m, n \in \mathbb{Q} \\
\frac{m\top}{n\top} = \frac{m}{n} &\quad : \quad m, n \in \mathbb{Q}
\end{aligned}$$

Note that $(r + n\top)$ is undefined for all $r \in \mathbb{R}$ in PEPA therefore disallowing components which enable both active and passive actions in the same action type at the same time, e.g. $(a, \lambda).P + (a, \top).P'$.