Abstract Interpretation of PEPA Models

Stephen Gilmore, Jane Hillston(✉), and Natalia Zoń

Laboratory for Foundations of Computer Science,
University of Edinburgh, Edinburgh, Scotland
jane.hillston@ed.ac.uk

Abstract. This paper relates the fluid-flow semantics of the stochastic process algebra PEPA (Performance Evaluation Process Algebra) to the static analysis technique of *abstract interpretation*. The explanation in the paper is illustrated through the example of a *distributed denial of service* (DDoS) attack which is being launched against a server. DDoS attacks are mounted by a large population of attackers, who are coordinating and working together in attacking a specific server. The scale of the attack is crucial to its success, but the resulting large number of states in the system makes it difficult to model and analyse using the conventional discrete-state interpretation of PEPA.

1 Introduction

Discrete-state modelling of computer systems is the bedrock of our attempts to gain intellectual control of informatic systems by building precise models of their behaviour and reasoning about these models. For some types of reasoning, such as identifying contention problems or finding deadlocks, it can be sufficient to work with a model with a small number of components, and show that the problem arises there. *Resource-constrained networks* [1] are an example of this, where several processes compete for resources from a limited pool, as are the so-called *feature interaction* problems which arise in telephony networks.

For other types of problems, such as distributed denial of service attacks (DDoS), it is *not* possible to scale down the analysis: the problem only arises when large-scale systems are involved. The case in point here is whether a service endpoint providing a service can continue to maintain the robustness of that service if larger and larger numbers of attackers threaten to overwhelm the server. This sentiment has been expressed very well in [2] where the authors write: "We believe that with quantitative information on the robustness, it will be possible to better determine whether or not the software continues to deal appropriately with risks and threats as their application environment changes." To analyse these problems in large-scale systems we need to use scalable modelling methods.

Concurrency in informatic systems effectively thwarts our attempts to reason about large-scale systems using a concrete interpretation of our models because the asynchronous interleaving of concurrent processes gives rise to state spaces which are too large to be represented. With the scalable, virtualised services

C.W. Probst et al. (Eds.): Nielsons' Festschrift, LNCS 9560, pp. 140–158, 2016.
DOI: 10.1007/978-3-319-27810-0_7

which are in use today components are replicated to provide resilience and application scalability to serve growing numbers of clients using the service. In order to reason about systems with replicated components it is essential to move away from discrete-state models and use representations which provide efficient representations of populations of components [3].

The PEPA stochastic process algebra [4] supports such reasoning about large-scale systems by providing a formal language which allows system behaviour to be captured as a discrete-state model, and verification methods which scale to allow the analysis of models which are composed of replicated components. This is achieved through an abstract interpretation of PEPA models via a representation in terms of ordinary differential equations over a continuous-state space. This is an over-approximation of the true discrete state-space, and this relaxation of the strict small-step interleaving semantics of PEPA allows efficient analysis of models which could not be contemplated by other means.

Structure of this paper: Our goal here is to describe a formal dynamic analysis approach which is based on a continuous-space abstraction of discrete-space systems. We illustrate the use of this analysis method in practice by developing a PEPA model of a server system which is trying to withstand a distributed denial of service attack. We use our continuous approximation of the model to investigate effective ways to defend against such attacks. The rest of the paper is organised as follows. In Sect. 2 we relate the subject matter of the present paper to other research and give pointers to important work in this area. In Sect. 3 we present the relevant technical background for this paper, with an introduction to the PEPA stochastic process algebra and its concrete small-step operational semantics. Section 4 explains the abstract interpretation of PEPA models with reference to the scalable differential semantics of PEPA. Section 5 presents the case study of the paper, involving a PEPA model of a DDoS attack. Finally, Sect. 6 concludes the paper.

2 Related Work

Static analysis of PEPA models has first been presented in 2007, in [5], where the author developed an approach based on data flow analysis. A transfer function is defined and then the classical worklist algorithm is used to construct a finite automaton capturing all possible interactions between components, on which deadlock detection can be based. Here we take an alternative approach based on abstract interpretation rather than data flow analysis [6]. The interpretation of PEPA models as a set of ordinary differential equations was initiated in [7]. Similar work has been done for the Kappa modelling language [8].

Three recent papers are very directly relevant to our presentation here because they also use process algebra and quantitative methods to model denial-of-service attacks. They are [9–11], as discussed next. Each builds on the Quality Calculus [12], a process algebra in the family of CCS and the π-calculus, intended to study the behaviour of software components in distributed systems when the communication has vulnerabilities; similar to the type of system considered at the end

of this paper. In particular, one motivation for the Quality Calculus is seeking to ensure that messages are correctly received, and denial of service attacks are a major challenge in this context. In [9], the authors introduce the Applied Quality Calculus which extends the Quality Calculus. The Applied Quality Calculus is used in modelling secure systems which must operate in the context of low computational power devices which communicate by broadcast communication in a challenging computational context where communication failures cannot be ignored. The Applied Quality Calculus has an executable semantics which is implemented in the Maude term-rewriting system, resulting in a simulation engine which can be used both directly for prototyping and for solving bounded reachability problems. Here, the notion of denial of service centres on the distinction between data and *optional data* introduced by the Quality Calculus. A rewrite rule-based mechanism is used to implement both an input selection mechanism and cryptographic reasoning in addition to designing *quality guards* which are more expressive than traditional predicates which are used in propositions.

In [10], the authors extend the Hybrid CSP language of He Jifeng with the notion of *binders* from the Quality Calculus. The modelling domain of interest here is *hybrid systems* which bring together discrete-state computational controllers and continuous-time physical systems into a dynamic assembly. A small-step transition semantics is presented for the language, in a timed, discrete-event context. The purpose of the modelling is to show that error configurations are not reachable and that the (continuous) velocities of the moving objects in the hybrid system cannot be degraded out of the safe range of operation. In [11], the same authors place greater emphasis on the safety aspects of the problem.

In [13], Zeng *et al.* extend the Quality Calculus with quantitative information capturing explicit timing and probability of actions. Unlike PEPA, where the probability distributions governing the delays associated with actions are restricted to follow an exponential distribution, the Stochastic Quality Calculus supports generally distributed delays but with the associated cost that the semantics gives rise to a Generalised Semi-Markov Decision Process in general, and a Generalised Semi-Markov Process when non-determinism can be eliminated. Both these mathematical structures are very difficult to analyse.

PEPA has previously been used to analyse security in [14], but in that paper the focus was on securing systems against timing attacks, where an attacker gains information about the activity on secure channels through eavesdropping on the timing characteristics of message exchanges. In the model developed in this paper we are specifically considering the case of a denial of service attack, where the strategy of the attacker is much more brute force, relying on scalability vulnerabilities of the service under attack.

3 Background

3.1 PEPA

PEPA is a compact formal modelling language which provides the appropriate abstract language constructs to represent many dynamic systems. It has

stochastically-timed activities which can be used to encode activities which take time to complete, such as data processing and a probabilistic choice operator to express the likelihood of different outcomes, for example in the presence of communication failures. Different patterns of behaviour are encoded in recursive process definitions. Features such as these are found in many modelling formalisms [15] but a distinctive strength of the PEPA language is that populations of components, encoded as arrays of process instances, are both convenient to express in the language and efficiently supported by the dynamic analysis which reveals the collective behaviour which emerges from the interactions of the populations of components. The PEPA language has found application in many modelling problems such as scalable and quantitative analysis of web-services [16–18], software performance engineering with UML-based models [19,20], secure key distribution [21], internet worms [22], and peer-to-peer systems [23].

As a process calculus, PEPA has CSP-style multiway communication, and actions in PEPA have durations. A PEPA model consists of a collection of *components* (also known as *processes*) which undertake actions [4]. A component may perform an action autonomously, *independent actions*, or in synchronisation with other components, *shared actions*. PEPA models are generated by the following two-level grammar:

$$S ::= (\alpha, r).S \quad | \quad S + S \quad | \quad A_S, A_S \stackrel{def}{=} S$$
$$C ::= S \quad | \quad C \underset{L}{\bowtie} C \quad | \quad C/L \quad | \quad A_C, A_C \stackrel{def}{=} C$$

The first production defines *sequential components*, i.e., processes which only exhibit sequential or branching behaviour (by means of prefix, ".", or choice, "+", respectively). The second production defines *model components*, in which the interactions between the sequential components are expressed through the cooperation ("$\underset{L}{\bowtie}$") and hiding ("/") operators. Within a cooperation, the set L specifies which action types must be shared; components can proceed independently and concurrently on other action types. A *system equation* specifies all the components within a system and how they must interact.

Typically, each sequential component corresponds to a component of the system and the performance of the system is constrained by the interactions between components as imposed by the cooperations. For example for a client-server system, some number of clients may compete for access to a limited number of servers. This may be written as the system equation

$$System \stackrel{def}{=} Client[N_c] \underset{\{request\}}{\bowtie} Server[N_s]$$

where $Client[N_c]$ is shorthand for $Client \underset{\emptyset}{\bowtie} \cdots \underset{\emptyset}{\bowtie} Client$ for a population of N_c clients, and similarly for $Server[N_s]$.

3.2 Concrete Semantics

Stochastic process algebras, such as PEPA, are typically given a semantics in terms of a labelled transition system, derived from small-step operational semantics. In other words, a set of semantic rules, shown in Fig. 1, detail the possible

evolutions of a term in the language based on the syntactical construction of the term. The transitions which are derived are labelled by the activities and thus contain information about the dynamic behaviour in terms of the expected rate of the transition in addition to the type of activity performed. This inclusion of information about the rates within the labelled transition system means that a multi-transition system must be used in order to correctly reflect the dynamics of the system, i.e., if there are multiple instances of the same transition the resulting action will occur at a faster rate than if there is only a single instance, because each instance contributes to the apparent rate of the action.

The rules in Fig. 1 correspond to the operators of the language introduced in the previous section. Most of the rules are straightforward, and presented here without comment. Rule C_2 is the fundamental inference for the characterisation of the dynamic behaviour of a shared action. It implements the semantics of *bounded capacity*: informally, the overall rate of execution of a shared activity is the minimum between the rates of the synchronising components. The rule relies

Prefix

$$\mathrm{S}_0 : \frac{}{(\alpha, r).E \xrightarrow{(\alpha,r)} E}$$

Choice

$$\mathrm{S}_1 : \frac{E \xrightarrow{(\alpha,r)} E'}{E + F \xrightarrow{(\alpha,r)} E' + F} \qquad \mathrm{S}_2 : \frac{F \xrightarrow{(\alpha,r)} F'}{E + F \xrightarrow{(\alpha,r)} E + F'}$$

Cooperation

$$\mathrm{C}_0 : \frac{E \xrightarrow{(\alpha,r)} E'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E' \underset{L}{\bowtie} F}, \alpha \notin L \qquad \mathrm{C}_1 : \frac{F \xrightarrow{(\alpha,r)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,r)} E \underset{L}{\bowtie} F'}, \alpha \notin L$$

$$\mathrm{C}_2 : \frac{E \xrightarrow{(\alpha,r_1)} E' \qquad F \xrightarrow{(\alpha,r_2)} F'}{E \underset{L}{\bowtie} F \xrightarrow{(\alpha,R)} E' \underset{L}{\bowtie} F'}, \alpha \in L$$

$$R = \frac{r_1}{r_\alpha(E)} \frac{r_2}{r_\alpha(F)} \min\left(r_\alpha(E), r_\alpha(F)\right)$$

Hiding

$$\mathrm{H}_0 : \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\alpha,r)} E'/L}, \alpha \notin L \qquad \mathrm{H}_1 : \frac{E \xrightarrow{(\alpha,r)} E'}{E/L \xrightarrow{(\tau,r)} E'/L}, \alpha \in L$$

Constant

$$\mathrm{A}_0 : \frac{E \xrightarrow{(\alpha,r)} E'}{A \xrightarrow{(\alpha,r)} E'}, A \stackrel{def}{=} E$$

Fig. 1. Markovian semantics of PEPA.

on the notion of *apparent rate* to compute the total capacity of a cooperating component, according to the following definition.

The *apparent rate* of action α in process E, denoted by $r_\alpha(E)$, indicates the overall rate at which α can be performed by E. It is recursively defined as follows:

$$r_\alpha((\beta, r).E) = \begin{cases} r \text{ if } & \beta = \alpha \\ 0 \text{ if } & \beta \neq \alpha \end{cases}$$

$$r_\alpha(E + F) = r_\alpha(E) + r_\alpha(F)$$

$$r_\alpha\left(E \underset{L}{\bowtie} F\right) = \begin{cases} \min(r_\alpha(E), r_\alpha(F)) & \text{if } \alpha \in L \\ r_\alpha(E) + r_\alpha(F) & \text{if } \alpha \notin L \end{cases}$$

$$r_\alpha(E/L) = \begin{cases} r_\alpha(E) \text{ if } & \alpha \notin L \\ 0 \quad\quad\;\; \text{if } & \alpha \in L \end{cases}$$

According to this definition, for the array of sequential components $Client[N_C]$, where $Client \stackrel{def}{=} (comm, r_d).Client'$, the apparent rate of $comm$ is

$$r_{comm}(Client[N_C]) = N_C\, r_{comm}(Client) = N_C \times r_d. \tag{1}$$

Similarly, for $Server \stackrel{def}{=} (comm, r_u).Server'$,

$$r_{comm}(Server[N_S]) = N_S\, r_{comm}(Server) = N_S \times r_u. \tag{2}$$

Once the labelled transition system, or *derivation graph*, corresponding to a PEPA model has been constructed then it can be interpreted as the state transition diagram of a continuous time Markov chain (CTMC). In this CTMC each state corresponds to a distinct syntactic form of the PEPA expression, as the model evolves according to the semantics. The CTMC is stored as an infinitesimal generator matrix, a matrix which captures the rates of transitions between states. From this the probability distribution over the states of the model at any given time, or at steady state, can be readily derived using standard linear algebra algorithms.

4 Abstract Interpretation of PEPA Models

4.1 Overview

As we saw in the previous section the concrete semantics of a PEPA model gives rise to a mathematical object, a continuous time Markov chain (CTMC). For model analysis the CTMC is encoded as an *infinitesimal generator matrix*, $\mathbf{Q}$. If the model has N distinct states in the derivation graph, $\mathbf{Q}$ will be an $N \times N$ matrix, with each entry $q(i, j)$ storing the rate of the exponential distribution governing transitions from state s_i to state s_j. The behaviour of the CTMC is captured in terms of its probability distribution, typically denoted $\pi(t)$, the N-dimensional vector in which the i-th entry denotes the probability to be in

state i at time t. For steady state properties of a system we are interested in $\pi(\infty)$, usually denoted simply as π, which can be found by linear algebra as the solution to the equations

$$\pi \mathbf{Q} = \mathbf{0} \qquad \sum_i \pi_i = 1$$

where the equation on the left represents the *global balance equations*, ensuring that at equilibrium the probability distribution is stable, and the equation on the right represents the normalisation condition, capturing that π is indeed a probability distribution.

Considering the operation of the modelled system, each possible behaviour can be regarded as a possible trajectory through the state space. In contrast to solving the CTMC, as described above, to find the probability distribution over states, simulating the CTMC generates a single trajectory. Thus the CTMC itself encodes all possible trajectories, and the state probability distribution gives the relative likelihood of each one. This is a complete encapsulation of all possible behaviours of the model of the system, and thus the concrete semantics can be regarded as being exhaustive with respect to the possible executions. However, this exhaustive view relies on being able to construct and manipulate the whole CTMC, i.e., its infinitesimal generator matrix, something that is not possible when the size of N grows too large (say $> 10^8$ states). The alternative, based on simulation amounts to *sampling* trajectories/behaviours. This is computationally costly especially as many repeated samples are needed in order to derive statistically sound results.

A continuous *space* alternative has been proposed which makes an abstract interpretation of the CTMC underlying a PEPA model [24]. Instead of a probability distribution over a set of possible trajectories, the model gives rise to a single system of ordinary differential equations (ODEs). These ODEs can be regarded as representing the expectation over the probability distribution over possible trajectories, or as the single trajectory that captures the average behaviour of the system. The seminal theorem by Kurtz [25], establishes that if we consider a sequence of CTMCs, with increasing populations of components interacting in the same proportions, then as the population increases, behaviour of the CTMCs converges to this single ODE trajectory. This corresponds to a functional form of the law of large numbers. Instead of a sequence of random variables, i.e., the sample mean, converging to a deterministic value, i.e., the true mean, here we have a sequence of CTMCs or trajectories for increasing population size which converge to a deterministic trajectory, given by the ODEs. Thus the ODE semantics provide an abstract semantics for PEPA, which can be regarded as an over-approximation since it summarises all possible behaviours obtained via the concrete semantics. Details of the abstract semantics are given in the following subsection.

4.2 Scalable Differential Semantics

In [26] the authors explain how the ODEs of the abstract semantics can be inferred statically from the PEPA model. This is done via a more abstract representation of the underlying CTMC in terms of generating functions, which are then approximated from discrete functions, to functions over continuous variables, the ODEs. This approach allows the consistency between the concrete semantics, now encoded in generating functions, and the abstract semantics, to be readily proved. Moreover it is shown that the ODEs generated are indeed those consistent with Kurtz's theorem, implying correctness of the abstract semantics as explained above. Furthermore in a subsequent paper it is shown how the abstract semantics may be used to derive performance measures from PEPA models, in addition to the evolution of population counts, giving the expectations of measures over trajectories rather than simply the expectation of the trajectories themselves [27].

Construction of the abstraction semantics proceeds in three steps:

1. **Context reduction:** identifying the component *types* in operation in the model and the local state space of each type, resulting in a reduced state representation based on a counting abstraction;
2. **Identify the jump multiset:** characterising the effect of each type of action in terms of a symbolic update on the reduced state representation;
3. **Define generating functions:** expressing the rate of transitions in the state space as a function of the state.

Context reduction: The aim of context reduction is to statically reduce the state representation of the PEPA model to its most compact form. Previous work [28], Gilmore *et al.*, showed how a static analysis could find lumpable partitions in the underlying state space of a PEPA model and find a reduced state space based on a canonical form of the PEPA model expressions. In [26] the syntax of the PEPA expression is discarded in the state representation and a numerical vector is used to capture the state of the model. The objective of the context reduction is to identify the entries which are needed for the representation. Roughly speaking, two components are considered to have the same *type* if they have the same derivation graph *and* they are subject to the same cooperation sets in the expression of the model. For each component type there is one entry in the numerical vector for each of the states in its derivation graph. For the definition of the semantics the entries of the numerical vector are represented symbolically $\xi = (\xi_1, \ldots, \xi_n)$. Note that this is a list of the *local* states of each of the component types considered in isolation, and consequently much smaller than a list of all the possible *global* states that could be encountered through their interleaving. This is an important distinction. A system of ODEs, the Chapman-Kolmogorov equations, describing the evolution of the system based on the concrete interpretation of the CTMC can also be constructed. In these equations the variables are the probability mass associated with each *global* state, i.e., there is one equation corresponding to every global state of the system. This system equation becomes unmanageable for models of any reasonable size.

The *reduced context* of a PEPA component P, denoted by $red(P)$, is recursively defined as follows:

$$
\begin{aligned}
red\left((\alpha, r).P\right) &= (\alpha, r).P \\
red\left(P + Q\right) &= P + Q \\
red(A \stackrel{def}{=} P) &= red(P) \\
red(P \underset{L}{\bowtie} P') &= \begin{cases} red(P), & \text{if } L = \emptyset \wedge P = P' \\ & \wedge P, P' \text{ are sequential components} \\ red(P) \underset{L}{\bowtie} red(P'), & \text{otherwise} \end{cases} \\
red(P/L) &= red(P)/L
\end{aligned}
$$

The jump multiset: Once we have the symbolic representation of a prototypical state of the system we can consider the transitions induced on this state representation by the actions in the PEPA model. For each action type, from the model specification, we can identify the impact that completing the action will have on the counts of component types involved in the action: if a component enables an action, then completing the action will decrease the corresponding count by 1; conversely if a component is a one-step derivative of the action, then its corresponding count will be increased by one. Thus for each action type in the model we can build an update vector which will make the appropriate change to the symbolic state vector whenever the action is completed. For example if the update vector or *jump* associated with an action α is $\mathbf{u}_\alpha$, and a state ξ completes an action α, then the resulting state will be $\xi + \mathbf{u}_\alpha$.

Generating functions: Finally, in order to capture the dynamics of the process we need to know the rate at which actions will be completed. In general this will depend on the state of the system since, as remarked earlier, when an action is enabled multiple times its apparent rate increases. However, the (symbolic) state representation gives us the information needed to deduce the multiplicities of actions enabled in any state and consequently the rates of actions can be expressed symbolically too. For example, if (α, r) is an individual action of the component P_j whose count is captured by the variable ξ_j, then the rate of α in an arbitrary state will be $\xi_j \times r$. The generating function would then be expressed as $f_\alpha(\xi, \mathbf{u}_\alpha) = \xi_j \times r$.

These functions are parametrised by action types to keep track of the additional information about which action type is associated with a transition. Let $\mathbf{u}_\alpha \in \mathbb{Z}^d$ be the *transition jump*. The generating functions are denoted by $f_\alpha(\xi, \mathbf{u}_\alpha) : \mathbb{R}^d \to \mathbb{R}$ and give the transition rate for a jump $\mathbf{u}_\alpha$ and an activity of type $\alpha \in \mathcal{A}$. Thus, the entry in the generator matrix corresponding to the transition from ξ to $\xi + \mathbf{u}$, denoted by $q_{\xi,\xi+\mathbf{u}}$, can be written as

$$q_{\xi,\xi+\mathbf{u}} = \sum_{\alpha \in \mathcal{A}} f_\alpha(\xi, \mathbf{u}_\alpha).$$

The summation across $\mathcal{A}$ captures the fact that distinct action types may contribute to a transition to the same target state, e.g., $(\alpha, r).P + (\beta, s).P$.

These transitions are kept distinct in the labelled transition system of PEPA, because it records the action type as well as the transition rate, but they collapse onto the same entry in the underlying generator matrix. We use the notation

$$f(\xi, \mathbf{u}) \equiv \sum_{\alpha \in \mathcal{A}} f_\alpha(\xi, \mathbf{u}_\alpha)$$

to indicate the overall contribution to the transition. The extraction of the generating functions from the PEPA model usually presents very little computational challenge because the environment collected via the inference rules in our operational semantics abstracts away from the (potentially very large) actual population levels of the system under study.

As stated above, when the generating functions are instantiated with a state representation based on integer counts and updates of the component types, they may be used to derive the state space of a CTMC corresponding to a PEPA model instantiated with that many copies of each component type. This is a template for all possible trajectories over the reduced state space. But when the functions are treated as continuous functions (we replace ξ by a vector of real values $\mathbf{x}$, and allow them to evolve continuously) they give rise to a vector field which defines the evolution of the expectation over the trajectories. Thus from $f(\xi, \mathbf{u})$ it is possible to construct a vector field $F : \mathbb{R}^d \to \mathbb{R}^d$ defined as

$$F(x) = \sum_{\mathbf{u} \in \mathbb{Z}^d} \mathbf{u} f(x, \mathbf{u}) \tag{3}$$

and an associated ODE

$$\frac{dx(t)}{dt} = F(x(t)). \tag{4}$$

5 Modelling a Distributed Denial of Service Attack

In this section, we turn to the case study of the paper: a *Distributed Denial of Service* (DDoS) attack. Such attacks on server-based systems are particularly challenging because they are fundamentally different in nature from attacks which exploit logical weaknesses in the design of communication protocols. Protocol breaking-attacks are fundamentally *qualitative* in nature. In contrast, DDoS attacks are fundamentally *quantitative* in nature. A Dolev-Yao attack requires ingenuity and cunning; a DDoS attack simply requires a large enough pool of attackers to take down the server by brute force.

We present the model in three instances, each of which builds on the model which came before. In every model, components are replicated to form populations of components. There are multiple clients, multiple servers, multiple attackers, and so forth. Populations are numerous. There are hundreds of instances of each component, not just five or ten. The three versions of the model are explained below.

- In the first version of our DDoS model we present only the *servers* and the *clients*. There are no attackers in the first model: the purpose of this first model is only to show idealised *optimal* service, and thereby to serve as a basis against which to compare *sub-optimal* service. We are mostly interested in how connections are made and held, so we focus on the Servers on the server side, and the process of connecting and disconnecting.
- In the second version of our model we add the *attackers*, showing how they impede the use of the server, making it much more difficult for genuine clients to get any service at all. Creating this form of unproductive interference is the essence of a distributed denial of service attack.
- In the third version of our model we introduce new components: the *defenders* try to impede the attackers, by monitoring each socket. As we will see, the defenders are not able to restore the optimal level of service which was enjoyed in the absence of the attackers but they lessen the effectiveness of the DDoS attack by impeding the attackers. This is done by introducing a *delay* which monitors the connection to the server and ejects a connection if it appears to be taking too long. Unfortunately this means that genuine clients may also be ejected sometimes but as we will see, although the attackers continue to frequently connect to the server, the number of clients trying unsuccessfully to connect is significantly reduced.

We encoded our model in PEPA and analysed it with the PEPA Eclipse Plug-in [29], a modelling tool developed in the European project SENSORIA (Software Engineering for Service-Oriented Overlay Computers) and subsequently used in teaching and research internationally. It incorporates a custom editor for PEPA models, model visualisation and static analysis tools, a model debugger, Markov chain analysis tools, stochastic simulation and discrete analysis tools, a model compiler which delivers a continuous representation of the system, efficient ODE-based solvers, and plotting functions for analysis results.

5.1 Model Parameters, for All Models

PEPA models have both continuous variables and discrete variables. The continuous variables are *rate parameters* which are used to model the exponentially-distributed *rate* at which actions (more properly, *activities*) occur. The discrete variables are *population counts* which count the number of copies of each PEPA component in the model. Multiplicities are important in all models, but they are important in DDoS models in particular. A DDoS attack which has been launched by 10 hostile bots somewhere on the Internet is not anywhere near as troubling as a DDoS attack which is being launched by 10,000,000 hostile bots.

5.2 First Model: Server and Clients only

The basic model is formed as the cooperation of a population of servers with a population of clients on some work to be done. There is a protocol of interaction observed by both servers and clients, as shown in Figs. 3 and 4.

Model number	Rate variable	Rate value	Variable description	Used by PEPA component(s)
1,2,3	r_z	0.02	the client's *think* rate	*Client*
1,2,3	r_c	1.0	the *connect* rate	*Client*, *Server*
1,2,3	r_h	10	*handshake* rate (if innocent)	*Client*, *Server*
1,2,3	r_a	0.001	*handshake* rate (if attackers)	*Client*, *Attacker*
1,2,3	r_s	0.1	the server's *serve* rate	*Server*
1,2,3	r_d	5	the *disconnect* rate	*Client*, *Server*
1,2,3	r_t	0.01	a general *timeout* rate	*Client*, *Server*
3 only	r_d	0.5	*delay* rate	*Defender*
3 only	r_y	10	*eject* rate when under attack	*Defender*

Fig. 2. Parameters of the model

$$
\begin{aligned}
Server_{free} &\stackrel{def}{=} (connect, r_c).Server_{claimed} \\
Server_{claimed} &\stackrel{def}{=} (handshake, r_h).Server_{ready} \\
Server_{ready} &\stackrel{def}{=} (serve, r_s).Server_{idle} + (timeout, r_t).Server_{free} \\
Server_{idle} &\stackrel{def}{=} (disconnect, r_d).Server_{free}
\end{aligned}
$$

Fig. 3. The *Server* component of the PEPA model

Figure 3 is the *Server* process, which, if viewed from an automata-theoretic perspective, would accept all and only the sentences of the formal language $(connect, handshake, (serve, disconnect) \mid timeout)^*$ and that of course means that the *Server* component can give rise to only two possible *traces*, which are

– (*connect*; *handshake*; *serve*; *disconnect*); or
– (*connect*; *handshake*; *timeout*).

Figure 4 shows the *Client* process, which, if viewed from an automata-theoretic perspective, would accept all and only the sentences of the formal language $(think, connect, handshake, (serve, disconnect) \mid timeout)^*$ and that of course means that the *Server* component can give rise to only two possible *traces*, which are

– (*think*; *connect*; *handshake*; *serve*; *disconnect*); or
– (*think*; *connect*; *handshake*; *timeout*).

In a normal interaction there is a two-stage connection, with the client first connecting to the server, and the server then confirming the connection with a handshake, before providing the required service. At the end of service the client disconnects, freeing the server. Between service interactions the client operates independently, indicated by the *think* action, appearing idle from the server's

$$
\begin{aligned}
Client_{idle} &\stackrel{def}{=} (think, r_z).Client_{enter} \\
Client_{enter} &\stackrel{def}{=} (connect, r_c).Client_{connected} \\
Client_{connected} &\stackrel{def}{=} (handshake, r_h).Client_{waiting} \\
Client_{waiting} &\stackrel{def}{=} (disconnect, r_d).Client_{idle} \\
&\quad + \ (timeout, r_t).Client_{enter}
\end{aligned}
$$

Fig. 4. The *Client* component of the PEPA model

perspective. To guard against dropped connections and other communication difficulties there is also the possibility for the server to timeout a connection which seems inactive. The components are combined as:

$$
\begin{aligned}
System_0 \stackrel{def}{=} & \ Server_{free}[200] \underset{\mathcal{L}}{\bowtie} Client_{idle}[1000] \\
& \text{where } \mathcal{L} = \{ connect, handshake, disconnect, timeout \}.
\end{aligned}
$$

The use of array notation syntax in PEPA (say, for example, $P[2]$) indicates a PEPA component *array*, which is syntactic sugar for $P \parallel P$, which is itself syntactic sugar for $P \underset{L}{\bowtie} P$ when L, the *cooperation set*, is $\emptyset$ (meaning the empty set, as usual).

The behaviour of $System_0$, with the rate parameters given in Fig. 2, is shown in Fig. 5.

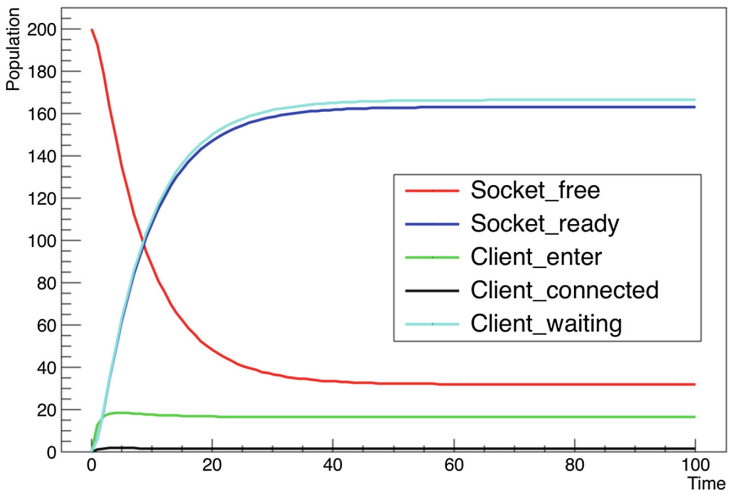

Fig. 5. The evolution of $System_0$ with the rate parameters given in Fig. 2

$$
\begin{aligned}
Attacker_{idle} &\stackrel{def}{=} (connect, r_c).Attacker_{connected} \\
Attacker_{connected} &\stackrel{def}{=} (handshake, r_a).Attacker_{hold} \\
Attacker_{hold} &\stackrel{def}{=} (timeout, r_t).Attacker_{idle} + (disconnect, r_d).Attacker_{idle}
\end{aligned}
$$

Fig. 6. The *Attacker* component of the PEPA model

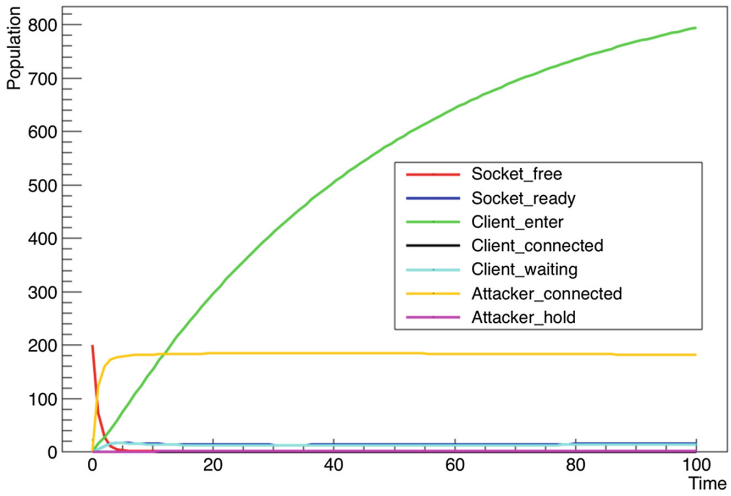

Fig. 7. The evolution of $System_1$ incorporating 250 attackers with the rate parameters given in Fig. 2

5.3 Second Model: Adding the Attackers

The objective of the attackers is to occupy the server for as long as possible so that it is unable to undertake any genuine service interactions. We might say that the attackers are 'tricking' the server. In any case, the semantics are very clear: the attacker initiates the protocol for requesting server-side computation masquerading as a genuine client. Only the initial part of the protocol is executed by the attacker: specifically, the sequential composition of actions which is (*connect*; *handshake*). The attacker has no intention of executing the second part of the protocol which performs the server-side computation cleanly (*serve*; *disconnect*). Moreover, in contrast to the *brisk* handshake of the genuine client, the attacker uses a *slow* handshake at a slower rate, r_a. Note that, *any* additional delay is of interest to the attacker because it impedes the progress of the genuine clients, and that is the attacker's priority. The behaviour of the attacker is shown in Fig. 6. The revised system becomes:

$$
\begin{aligned}
Server_{free} &\stackrel{def}{=} (connect, r_c).Server_{claimed} \\
Server_{claimed} &\stackrel{def}{=} (handshake, r_h).Server_{ready} \\
Server_{ready} &\stackrel{def}{=} (serve, r_s).Server_{idle} \\
&\quad + (timeout, r_t).Server_{free} \\
&\quad + (eject, r_e).Server_{free} \\
Server_{idle} &\stackrel{def}{=} (disconnect, r_d).Server_{free} \\
&\quad + (eject, r_e).Server_{free}
\end{aligned}
$$

$$
\begin{aligned}
Defender &\stackrel{def}{=} (connect, r_c).Defender_1 \\
Defender_1 &\stackrel{def}{=} (delay, r_y).Defender_2 \\
&\quad + (disconnect, r_d).Defender \\
&\quad + (timeout, r_t).Defender \\
Defender_2 &\stackrel{def}{=} (eject, r_e).Defender
\end{aligned}
$$

$$
\begin{aligned}
Client_{idle} &\stackrel{def}{=} (think, r_z).Client_{enter} \\
Client_{enter} &\stackrel{def}{=} (connect, r_c).Client_{connected} \\
Client_{connected} &\stackrel{def}{=} (handshake, r_h).Client_{waiting} \\
Client_{waiting} &\stackrel{def}{=} (disconnect, r_d).Client_{idle} \\
&\quad + (timeout, r_t).Client_{enter} \\
&\quad + (eject, r_e).Client_{idle}
\end{aligned}
$$

$$
\begin{aligned}
Attacker_{idle} &\stackrel{def}{=} (connect, r_c).Attacker_{connected} \\
Attacker_{connected} &\stackrel{def}{=} (handshake, r_a).Attacker_{hold} \\
Attacker_{hold} &\stackrel{def}{=} (timeout, r_t).Attacker_{idle} \\
&\quad + (disconnect, r_d).Attacker_{idle} \\
&\quad + (eject, r_e).Attacker_{idle}
\end{aligned}
$$

$$
\begin{aligned}
System_2 \stackrel{def}{=}\ & (Server_{free}[200] \underset{\mathcal{L}_1}{\bowtie} Defender[200]) \\
& \underset{\mathcal{L}_2}{\bowtie} (Client_{idle}[1000] \parallel Attacker_{idle}) \\
& \text{where } \mathcal{L}_1 = \{\, connect, disconnect, timeout, eject \,\} \\
& \text{and } \mathcal{L}_2 = \{\, connect, handshake, disconnect, timeout, eject \,\}.
\end{aligned}
$$

Fig. 8. Modified model containing the DDoS defence

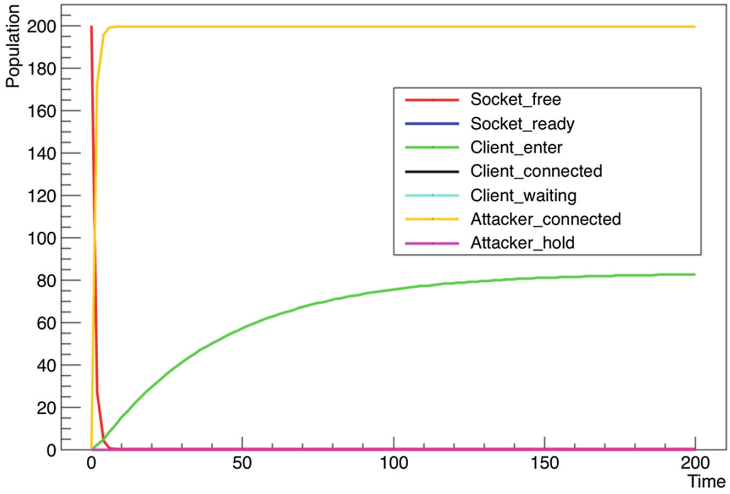

Fig. 9. The evolution of $System_2$ incorporating defence mechanisms with the rate parameters given in Fig. 2

$$System_1 \stackrel{def}{=} Server_{free}[200] \underset{\mathcal{L}}{\bowtie} Client_{idle}[1000]$$
$$\text{where } \mathcal{L} = \{\, connect, handshake, disconnect, timeout \,\}.$$

The behaviour of $System_1$, with the attackers incorporated and with the rate parameters given in Fig. 2, is shown in Fig. 7. In contrast to the behaviour seen in Fig. 5, we can see that the system does not reach a steady state behaviour, as a growing number of clients are in the state waiting for connection to the server. It can be seen that very quickly after the start of the attack, most of the sockets are held by an attacker, leaving almost no capacity for the genuine clients.

5.4 Third Model: Adding the Defenders

In order to defend against the DDoS attack we introduce a *Defender* which monitors each socket. When a connection appears to be consuming too much time it ejects the connection, allowing a fresh competition for connection between attackers and clients. The new model is shown in Fig. 8. Note that the *delay* in the *Defender* is raced against the on-going socket connection and aborts the interaction if the *delay* completes before either a *disconnect* or a *timeout*.

The results of analysis of the model are shown in Fig. 9 with all parameters values as shown in Fig. 2, i.e., the characteristics of the servers, users and attackers are unchanged except for the addition of the rapid *eject* action. We can see that the attackers are still successful in gaining access to the server, but there is much more turnover meaning that clients are also able to gain connections. This is evident because the number of clients in the $Client_{enter}$ state, waiting to form a connection, is significantly reduced compared to the growing value in Fig. 7.

6 Discussion and Conclusions

The phenomena studied and models presented here are intimately related to the scale of the system. It simply would not be possible to study the model with a numerical analysis of the CTMC derived from the discrete-state concrete semantics of the PEPA models because the state spaces generated by these models are prohibitively large. In contrast, stochastic simulation would be possible but much more computationally expensive. For example, running a simulation of $System_2$ (1000 replications) takes several minutes whereas the ODE-based analysis completes in a fraction of a second. This speed of solution makes the ODE-approach very suitable for exploring parameter space, for example to find the best value for the rate of the *delay* action in the *Defender*.

The abstract interpretation of PEPA models also makes it feasible to address the problem of model synthesis. In [30] the authors present an extension to the PEPA Eclipse Plug-in tool, which allows the user to specify a performance requirement for a model, currently expressed in terms of response time or throughput. From this specification the tool automatically searches parameter space to find the "smallest" model which is able to satisfy the performance requirement. Here "smallest" is essentially taken to mean the smallest number of components, but a user-defined cost function allows the modeller to weight different types of components differently (see [30] for details). The particle swarm optimisation (PSO) meta-heuristic [31] is used to efficiently explore parameter space. Nevertheless this approach would not be tractable if based on the discrete-state space representation of the PEPA models except for very small models.

Acknowledgements. This work is supported by the EU project QUANTICOL, 600708. The authors thank Mirco Tribastone for useful discussions.

References

1. Yüksel, E., Nielson, H.R., Nielson, F.: Key update assistant for resource-constrained networks. In: 2012 IEEE Symposium on Computers and Communications, ISCC 2012, Cappadocia, 1–4 July 2012, pp. 75–81. IEEE (2012)
2. Nielson, F., Nielson, H.R., Zeng, K.: Stochastic model checking of the stochastic quality calculus. In: De Nicola, R., Hennicker, R. (eds.) Wirsing Festschrift. LNCS, vol. 8950, pp. 522–537. Springer, Heidelberg (2015)
3. Hillston, J.: The benefits of sometimes not being discrete. In: Baldan, P., Gorla, D. (eds.) CONCUR 2014. LNCS, vol. 8704, pp. 7–22. Springer, Heidelberg (2014)
4. Hillston, J.: A Compositional Approach to Performance Modelling. Cambridge University Press, New York (1996)
5. Yang, F.: Static Analysis of Stochastic Process Algebras. MSc dissertation (2007)
6. Cousot, P.: Abstract interpretation based formal methods and future challenges. In: Wilhelm, R. (ed.) Informatics: 10 Years Back, 10 Years Ahead. LNCS, vol. 2000, p. 138. Springer, Heidelberg (2001)
7. Hillston, J.: Fluid flow approximation of PEPA models. In: Proceedings of the Second International Conference on the Quantitative Evaluation of Systems, pp. 33–43. IEEE Computer Society Press, Torino, September 2005

8. Danos, V., Feret, J., Fontana, W., Krivine, J.: Abstract interpretation of cellular signalling networks. In: Logozzo, F., Peled, D.A., Zuck, L.D. (eds.) VMCAI 2008. LNCS, vol. 4905, pp. 83–97. Springer, Heidelberg (2008)
9. Vigo, R., Nielson, F., Nielson, H.R.: Broadcast, denial-of-service, and secure communication. In: Johnsen, E.B., Petre, L. (eds.) IFM 2013. LNCS, vol. 7940, pp. 412–427. Springer, Heidelberg (2013)
10. Wang, S., Nielson, F., Nielson, H.R.: A framework for hybrid systems with denial-of-service security attack. CoRR, abs/1403.6367 (2014)
11. Wang, S., Nielson, F., Nielson, H.R.: Denial-of-service security attack in the continuous-time world. In: Ábrahám, E., Palamidessi, C. (eds.) FORTE 2014. LNCS, vol. 8461, pp. 149–165. Springer, Heidelberg (2014)
12. Nielson, H.R., Nielson, F., Vigo, R.: A calculus for quality. In: Păsăreanu, C.S., Salaün, G. (eds.) FACS 2012. LNCS, vol. 7684, pp. 188–204. Springer, Heidelberg (2013)
13. Zeng, K., Nielson, F., Nielson, H.R.: The stochastic quality calculus. In: Kühn, E., Pugliese, R. (eds.) COORDINATION 2014. LNCS, vol. 8459, pp. 179–193. Springer, Heidelberg (2014)
14. Buchholtz, M., Gilmore, S., Hillston, J., Nielson, F.: Securing statically-verified communications protocols against timing attacks. In: Bradley, J., Knottenbelt, W. (eds.) Proceedings of the First International Workshop on Practical Applications of Stochastic Modelling, pp. 61–79. England, London (2004)
15. Clark, A., Gilmore, S., Hillston, J., Tribastone, M.: Stochastic process algebras. In: Bernardo, M., Hillston, J. (eds.) SFM 2007. LNCS, vol. 4486, pp. 132–179. Springer, Heidelberg (2007)
16. Gilmore, S., Tribastone, M.: Evaluating the scalability of a Web service-based distributed e-learning and course management system. In: Bravetti, M., Núñez, M., Zavattaro, G. (eds.) WS-FM 2006. LNCS, vol. 4184, pp. 214–226. Springer, Heidelberg (2006)
17. Bravetti, M., Gilmore, S., Guidi, C., Tribastone, M.: Replicating Web services for scalability. In: Barthe, G., Fournet, C. (eds.) TGC 2007. LNCS, vol. 4912, pp. 204–221. Springer, Heidelberg (2008)
18. Cappello, I., Clark, A., Gilmore, S., Latella, D., Loreti, M., Quaglia, P., Schivo, S.: Quantitative analysis of services. In: Wirsing, M., Hölzl, M. (eds.) SENSORIA. LNCS, vol. 6582, pp. 522–540. Springer, Heidelberg (2011)
19. Tribastone, M., Gilmore, S.: Automatic extraction of PEPA performance models from UML activity diagrams annotated with the MARTE profile. In: Proceedings of the 7th International Workshop on Software and Performance (WOSP2008), pp. 67–78. ACM Press, Princeton (2008)
20. Tribastone, M., Gilmore, S.: Automatic translation of UML sequence diagrams into PEPA models. In: 5th International Conference on the Quantitative Evaluation of Systems (QEST 2008), pp. 205–214. IEEE Computer Society Press, St Malo (2008)
21. Zhao, Y., Thomas, N.: Approximate solution of a PEPA model of a key distribution centre. In: Kounev, S., Gorton, I., Sachs, K. (eds.) SIPEW 2008. LNCS, vol. 5119, pp. 44–57. Springer, Heidelberg (2008)
22. Bradley, J.T., Gilmore, S., Hillston, J.: Analysing distributed Internet worm attacks using continuous state-space approximation of process algebra models. J. Comput. Syst. Sci. **74**(6), 1013–1032 (2008)
23. Duguid, A.: Coping with the parallelism of BitTorrent: conversion of PEPA to ODEs in dealing with state space explosion. In: Asarin, E., Bouyer, P. (eds.) FORMATS 2006. LNCS, vol. 4202, pp. 156–170. Springer, Heidelberg (2006)

24. Hillston, J.: Fluid flow approximation of PEPA models. In: Second International Conference on the Quantitative Evaluaiton of Systems (QEST 2005), 19–22 September 2005, pp. 33–43. IEEE Computer Society, Torino (2005)
25. Kurtz, T.G.: Solutions of ordinary differential equations as limits of pure jump markov processes. J. Appl. Probab. **7**, 49–58 (1970)
26. Tribastone, M., Gilmore, S., Hillston, J.: Scalable differential analysis of process algebra models. IEEE Trans. Softw. Eng. **38**(1), 205–219 (2012)
27. Tribastone, M., Ding, J., Gilmore, S., Hillston, J.: Fluid rewards for a stochastic process algebra. IEEE Trans. Softw. Eng. **38**(4), 861–874 (2012)
28. Gilmore, S., Hillston, J., Ribaudo, M.: An efficient algorithm for aggregating PEPA models. IEEE Trans. Softw. Eng. **27**(5), 449–464 (2001)
29. Tribastone, M., Duguid, A., Gilmore, S.: The PEPA eclipse plug-in. Perform. Eval. Rev. **36**(4), 28–33 (2009)
30. Williams, C.D., Hillston, J.: Automated capacity planning for PEPA models. In: Horváth, A., Wolter, K. (eds.) EPEW 2014. LNCS, vol. 8721, pp. 209–223. Springer, Heidelberg (2014)
31. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization; an overview. Swarm Intell. **1**(1), 33–57 (2007)