Formal Modeling and Analysis of DoS Using Probabilistic Rewrite Theories*

Gul Agha, Michael Greenwald, Carl A. Gunter, Sanjeev Khanna
Jose Meseguer, Koushik Sen, and Prasannaa Thati†

May 15, 2005

Abstract

Existing models for analyzing the integrity and confidentiality of protocols need to be extended to enable the analysis of availability. Prior work on such extensions shows promising applications to the development of new DoS countermeasures. Ideally it should be possible to apply these countermeasures systematically in a way that preserves desirable properties already established. This paper investigates a step toward achieving this ideal by describing a way to expand term rewriting theories to include probabilitic aspects that can show the effectiveness of DoS countermeasures. In particular, we consider the shared channel model, in which adversaries and valid participants share communication bandwidth according to a probabilistic interleaving model, and a countermeasure known as selective verification applied to the handshake steps of the TCP reliable transport protocol. These concepts are formulated in a probabilistic extension of the Maude term rewriting sytem and automated techniques are used to demonstrate the effectiveness of the countermeasures.

1 Introduction

There are well-understood models on which to base the analysis of integrity and confidentiality. The most common approaches are algebraic techniques [2] based on idealized cryptographic primitives and complexity-theoretic techniques [1] based on assumptions about complexity. There has also been progress on unified perspectives that enable using the simpler algebraic techniques to prove properties like those ensured by the more complete cryptographic techniques. However, neither of these approaches or their unifications is designed to approach the problem of availability threats in the protocols they analyze. To fix on an example, suppose a protocol begins by a sender sending a short message to a receiver where the receiver's first step is to verify a public key signature on the message. A protocol like this is generally considered to be problematic because an adversarial sender can send many packets with bad signatures at little cost to himself while the receiver will need to work hard to (fail to) verify these signatures. Algebraic and complexity-theoretical analysis techniques ensure only that the recipient will not be fooled by the bad packets and will not leak information as a result of receiving them. They do not show that the receiver will be available to a valid sender in the presence of one or more invalid attackers.

In [3] we began an effort to explore a formal model for the analysis of DoS based on a simple probabilistic model called the "shared channel" model. This study shows that the shared channel model could be used to prove properties of DoS countermeasures for authenticated broadcast that could be verified in experiments. We have subsequently conducted a number of experiments to explore the application of such countermeasures to other

*Appearing in IEEE Foundations of Computer Security (FCS '05), Chicago IL, June 2005.

†Addresses of the authors: K. Sen, G. Agha, C. A. Gunter, J. Meseguer, University of Illinois at Urbana-Champaign; Michael Greenwald, Lucent Bell Labs; Sanjeev Khanna, University of Pennsylvania; Prasannaa Thati, Carnegie-Mellon University. This work was supported by ONR grant N00014-02-1-0715.

classes of protocols. The aim of this paper is to explore the prospects for using the shared channel model as a foundation for extending term rewriting models of network protocols to cover DoS aspects of the protocols and their modification with counter-measures. Our particular study is to investigate the use of a probabilistic extension of the Maude rewrite system called *PMaude* and its application to understanding the effectiveness of a DoS countermeasure known as "selective sequential verification". This technique was explored for authenticated broadcast in [3] but in the current paper we consider its application to handshake steps of the TCP reliable transport protocol.

At a high level, our ultimate aim in this work is to demonstrate techniques for showing how a network protocol can be systematically "hardened" against DoS using probabilistic techniques while preserving the underlying correctness properties the protocol was previously meant to satisfy. That is, given a protocol P and a set of properties T, we would like to expand T to a theory T^* that is able to express availability properties and show that a transformation P^* of P meets the constraints in T^* without the needing to re-prove the properties T that P satisfied in the restricted language. The shared channel model provides a mathematical concept for this extension.

In this paper we develop a key element of this program: a formal language in which to express the properties T^* and show that availability implications hold for P^*. We attempt to validate this effort by showing its effectiveness on a selective verification for TCP. In particular, we show how we can specify TCP/IP 3-way handshake protocol in PMaude algebraically. First, we take a previously specified formal non-deterministic model of the protocol. We then replace all non-determinism by probabilities. The resultant model with quantified non-determinism (or probabilities) is then analyzed for quantitative properties such as availability. The analysis is done by combining Monte-Carlo simulation of the model with statistical reasoning. In this way, we leverage the existing modelling and reasoning techniques to quantified reasoning without interfering with the underlying non-quantified properties of the model.

The rest of the paper is organized as follows. In Section 2, we give the preliminaries of DoS theory followed by its application to TCP/IP 3-way handshaking protocol in Section 3. Then we briefly describe PMaude and actor PMaude in Section 4. In Section 5, we describe and discuss the algebraic probabilistic specification of DoS hardened TCP/IP protocol in actor PMaude. We introduce a quantitative property query language in Section 6. We describe the results of our analysis of some desired properties written in the query language for the specification of TCP/IP protocol in Section 6.5 followed by conclusion.

2 DoS Theory

On the face of it, the established techniques for establishing confidentiality and integrity are inappropriate for analyzing DoS since they rely on very strong models of the adversary's control of the network. In particular they assume that the adversary is able to delete packets from the network at will. An adversary with this ability has an assured availability attack. Typical analysis techniques therefore adapt this assumption in one of two ways. A first form of availability analysis within these frameworks is to focus on the relationship between the sender and the attacker and ask whether the attacker/sender is being forced to expend at least as much effort as the valid receiver. In our example this is an extremely disproportionate level of effort since forming a bad signature is much easier than checking that it is bad. Thus the protocol is vulnerable to the imposition of a disproportionate effort by the receiver. This is a meaningful analysis, but it does not answer the question of whether a valid sender will experience the desired availability. A second form of availability analysis is to ask whether the receiver can handle a specified load. For instance, a stock PC can check about 8000 RSA signatures each second, and it can receive about 9000 packets (1500 bytes per packet) each second over a 100Mbps link. Thus a receiver is unable to check all of the signatures it receives over such a channel. A protocol of the kind we have envisioned is therefore deemed to be vulnerable to a *signature flood* attack based on cycle exhaustion. By contrast, a stock PC can check the hashes on 77,000 packets each second, so a receiver that authenticates with hashes can service all of its bandwidth

using a fraction of its capacity. This sort of analysis leads one to conclude that a protocol based on public key signatures is vulnerable to DoS while one based on hashes is not.

These techniques are sound but overly conservative, because they do not explicitly account for the significance of *valid* packets that reach the receiver. Newer techniques for analyzing DoS have emerged in the last year that provide a fresh perspective by accounting for this issue. In essence these new models are both more realistic for the Internet and suggest new ideas for countermeasures. We refer to one basic version of this new approach as the *shared channel model*. The shared channel model is a four-tuple consisting of the minimum bandwidth W_0 of the sender, the maximum bandwidth W_1 of the sender (where $W_0 \leq W_1$), the bandwidth α of the adversary, and the loss rate p of the sender where $0 \leq p < 1$. The ratio $R = \alpha/W_1$ is the *attack factor* of the model. When $R = 1$, this is a *proportionate* attack and, when $R > 1$, it is a *disproportionate* attack. As in the algebraic model, the adversary is assumed to be able to replay packets seen from valid parties and flood the target with anything he can form from these. But in the shared channel model he is not able to delete specific packets from the network. In effect, he is able to interleave packets among the valid ones at a specified maximum rate. This interleaving may contribute to the loss rate p of the sender, but the rate of loss is assumed to be bounded by p and randomly applied to the packets of the sender.

The key insight that underlies the techniques in this paper arises from recognizing the *asymmetry* the attacker aims to exploit; his willingness to spend his entire bandwidth on an operation that entails high cost for the receiver also offers opportunities to burden the attacker in disproportionate ways relative to the valid sender. This can be seen in a simple strategy we call *selective verification*. The idea is to cause the receiver to treat the signature packets she receives as arriving in an artificially lossy channel. The sender compensates by sending extra copies of his signature packets. If the recipient checks the signature packets she receives with a given probability, then the number of copies and the probability of verification can be varied to match the load that the recipient is able to check. For example, suppose a sender sends a 10Mbps stream to a receiver, but this is mixed with a 10Mbps stream of DoS packets devoted entirely to bad signatures. To relieve the recipient of the need to check all of these bad signatures, the receiver can check signatures with a probability of 25%, and, if the sender sends about 20 copies of each signature packet, the receiver will find a valid packet with a probability of more than 99% even if the network drops 40% of the sender's packets. This technique is inexpensive, scales to severe DoS attacks, and is adaptable to many different network characteristics.

3 SYN Floods as DoS for TCP/IP

TCP is an extremely common bi-directional stream protocol that uses acknowledgements and retransmissions for reliability, per-byte sequence numbers and windows for flow control, and three-way handshakes to establish and terminate connections. We assume that readers are broadly familiar with TCP and present an overly simplified description ignoring many details. TCP connections pass through three phases: connection initiation, data transfer, and connection termination. A sender initiates a connection by sending a packet with the SYN flag set and an initial sequence number. The receiver responds by acknowledging the SYN flag, and sending back a SYN of its own with its *own* sequence number. When the original sender acknowledges the receiver's SYN (the 3rd packet in a 3-way handshake), then the connection is ESTABLISHED.

Each established connection requires a TCB (Transmission Control Block) to be allocated at each end of the connection. The TCB occupies a few hundred bytes of connection identification, control information, and statistics, as well as a much larger allocation of packet buffers to receive data and hold other data awaiting transmission. In most operating system kernels both packet buffer space and the number of available TCBs are fixed at boot time and constitute a limited resource. This opens a significant vulnerability to adversaries who aim to overwhelm this limit by flooding a server with SYN packets; this is typically called a *SYN flood attack*. This threat is mitigated in many systems by storing connection information in a SYN cache (a lighter-weight data structure, recording only

identity information and sequence numbers for the connection) until the connection becomes ESTABLISHED, at which point the (more expensive) full TCB is allocated.

Normally, a legitimate connection occupies a slot in the SYN cache for only one round trip time (RTT). If no ACK for the SYN+ACK arrives, then the server eventually removes the entry from the SYN cache, but only after a timeout interval, t_A, which usually ranging from 30-120 seconds. TCP has other structures, such as packet reassembly buffers, that can become bottleneck resources, but those that are cleared after a timeout can be analyzed in a fashion similar to the SYN cache. If there is no timeout, then they can be analyzed like the TCB table.

SYN flooding attacks constitute an easy denial of service attack because SYN cache entries are relatively scarce, while the bandwidth needed to send a single SYN packet is relatively cheap. The attacker also gains leverage from the disparity between the one RTT slot occupancy (often on the order of a millisecond or less) for a legitimate client, compared with a fraudulent SYN packet that can hold a syn-cache slot for $t_A = 100$ seconds. In a SYN cache with $B = 10,000$ slots, and a 100 second hold time, only approximately 100 slots open up each second under a determined DoS attack. An energetic attacker can generate 300,000 SYN packets each second on a 100Mbps Fast Ethernet link, making it extremely unlikely that a legitimate client will successfully get any of the newly freed slots.

A SYN attack is simple to model; attackers merely send SYN packets, blindly. The attack can be characterized by the cumulative attackers' SYN arrival rate, which we will denote by r_A. To compute the effectiveness of the DoS attack, we must determine the probability of success of a client's attempt to connect, and from that compute the number of legitimate connections per second that the server can support under a given attack rate r_A.

As a baseline case, it is instructive to understand the simplest scenario, in which the server offers no defense. If the order in which incoming SYNs are processed at the server is adversarially chosen, then it is clear that $r_A \geq B/t_A$ suffices to completely take over the syn-cache, forcing the server to drop all valid requests. To see this, observe that every second B/t_A of the attacker's slots in the SYN cache expire, and B/t_A new ones arrive to take their places. Even in a more realistic model where the incoming SYN requests at any time instant are assumed to be ordered in accordance with a random permutation, it is easy to show that an attack rate of $O(B/t_A)$ suffices.

It is clear from this analysis (as well as from abundant empirical evidence) that even a moderate rate of DoS attack can totally disable a server. For a server with a SYN cache of size 10,000 and a timeout interval of 75 seconds a moderate attack rate of 200 to 300 SYNS per second is enough to almost completely overwhelm the server!

Selective verification can improve this performance significantly. Let B denote the number of slots in the SYN cache. Suppose we want to ensure that the attacker never blocks more than $f \times B$ table entries for some fraction $0 < f < 1$. We ask the server to process each incoming SYN with probability p where p satisfies $pt_A r_A \leq fB$, then we ensure that at least a $(1 - f)$-fraction of the SYN cache is available to legitimate users. We are effectively inflating the bandwidth cost of mounting an attack rate of r_A to be r_A/p. Considering once again an attacker on 100 Mbps channel ($300,000$ SYNs/sec), if we set $p = 10^{-3}/6$, we ensure that the attacker cannot occupy more than half the table at any point in time. The attacker can still deny service, but is now required to invest as much in bandwidth resources as the collective investment of the clients that it is attacking.

If we increase the cache size by a factor of 30, we can get an identical guarantee with $p = .005$. The overhead on a valid client to establish a connection then is only 200 SYN packets, roughly 8KB, for each request. These overheads are not insignificant but they allow us to provide unconditional guarantees on availability of resources for valid clients. If we downloaded the PS version of this paper (500KB), the blowup increases the transfer size by 2%. Moreover, these overheads should be contrasted with the naive alternative: the cache size would have to be increased to 6×10^7 to get the same guarantee.

4 Probabilistic Rewrite Theories

Rewriting logic is an expressive semantic framework to specify a wide range of concurrent systems [**?**]. In practice, however, some systems may be probabilistic in nature, either because of their environment, or by involving probabilistic algorithms by design, or both. This raises the question of whether such systems can also be formally specified by means of rewrite rules in some suitable probabilistic extension of rewriting logic. This would provide a general formal specification framework for probabilistic systems and would support different form of symbolic simulation and formal analysis. In particular, DoS-resistant communication protocols such as the DoS-hardened TCP/IP protocol discussed in Section 5 could be formally specified and analyzed this way.

The answer to whether such a semantic framework exists is affirmative, and is provided by the notion of a probabilistic rewrite theory. Usually, the rewrite rules specifying a non-probabilistic system are of the form

$$\texttt{crl}\ \ [L] : t \Rightarrow t' \quad \texttt{if}\ \ C$$

where the variables appearing in t' are typically a subset of those appearing in t, and where C is a condition. The intended meaning of such a rule is that if a fragment of the system's state is a substitution instance of the pattern t, say with substitution θ, and the condition $\theta(C)$ holds, then our system can perform a local transition in that state fragment changing it to a new local state $\theta(t')$. Instead, in the case of a probabilistic system, we will be using rewrite rules of the form,

$$\texttt{crl}\ \ [L] : t(\overrightarrow{x}) \Rightarrow t'(\overrightarrow{x}, \overrightarrow{y})\ \ \texttt{if}\ C(\overrightarrow{x})\ \ \texttt{with probability}\ \overrightarrow{y} := \pi_r(\overrightarrow{x})$$

where the first thing to observe is that the the term t' has new variables $\overrightarrow{y}$ disjoint from the variables $\overrightarrow{x}$ appearing in t. Therefore, such a rule is *non-deterministic*; that is, the fact that we have a matching substitution θ such that $\theta(C)$ holds does not uniquely determine the next state fragment: there can be many different choices for the next state, depending on how we instantiate the extra variables $\overrightarrow{y}$. In fact, we can denote the different such next states by expressions of the form $t'(\theta(\overrightarrow{x}), \rho(\overrightarrow{y}))$, where θ is fixed as the given matching substitution, but ρ ranges along all the possible substitutions for the new variables $\overrightarrow{y}$. The probabilistic nature of the rule is expressed by the notation `with probability` $\overrightarrow{y} := \pi_r(\overrightarrow{x})$, where $\pi_r(\overrightarrow{x})$ is a probability distribution *which depends on the matching substitution* θ, and we then choose the values for $\overrightarrow{y}$, that is the substitution ρ, probabilistically according to the distribution $\pi_r(\theta(\overrightarrow{x}))$.

We can illustrate these ideas with a very simple example, namely a digital battery-operated clock that measures time in seconds. The state of the clock is represented by a term `clock(t,c)`, where `t` is the current time in seconds, and `c` is a rational number indicating the amount of charge in the battery. The clock ticks according to the following probabilistic rewrite rule:

```
[tick] : clock(t,c) ⇒
                    if b then
                        clock(t + 1,c-c/1000)
                    else
                        broken(t,c - c/1000)
                    fi
         with probability B := BERNOULLI(c/1000) .
```

Note that rule's righthand side has a new boolean variable `B`. If all goes well (`B = true`), then the clock increments its time by one second and the charge is slightly decreased; but if `B = false`, then the clock will go into a broken state `broken(t,c - c/1000)`. Here the boolean variable `B` is distributed according to the Bernoulli distribution with mean $\frac{c}{1000}$. Thus, the value of `B` *probabilistically depends on the amount of charge* left in the

battery: the lesser the charge level, the greater the chance that the clock will break; that is, we have different probability distributions for different matching substitutions θ of the rule's variables (in particular, of the variable `c`).

Of course, in this example the variable `B` is a discrete binary variable; but we could easily modify this example to involve continuous variables. For example, we could have assumed that `t` was a real number, and we could have specified that the time is advanced to a new time `t + t'`, with `t'` a new real-valued variable chosen according to an exponential distribution. In general, the set of new variables $\overrightarrow{y}$ could contain both discrete and continuous variables, ranging over different data types. In particular, both discrete and continuous time Markov chains can easily be modeled, as well as a wide range of discrete or continuous probabilistic systems, which may also involve nondeterministic aspects [4]. Furthermore, the PMaude extension of the Maude rewriting logic language allows us to symbolically simulate probabilistic rewrite theories [5, **?**], and we can formally analyze their properties according to the methods described in Section 6.

We give below a precise mathematical definition of probabilistic rewrite theories. Note that an ordinary rewrite theory [6] is a triple $\mathcal{R} = (\Sigma, E, R)$, with (Σ, E) and equational theory and with R a collection of possibly conditional rewrite rules.

Definition 1 (Probabilistic rewrite theory) *A* probabilistic rewrite theory *is a 4-tuple $\mathcal{R} = (\Sigma, E \cup A, R, \pi)$, with (Σ, E, R) a rewrite theory where the equations E are confluent and terminating (perhaps modulo some structural axioms) and the rules R are coherent with respect to the equations E [**?**]. Furthermore, the rules $r \in R$ are of the form*

$$\texttt{crl}\ \ [L] : t(\overrightarrow{x}) \longrightarrow t'(\overrightarrow{x}, \overrightarrow{y})\ \textit{if}\ C(\overrightarrow{x})$$

where

- *$\overrightarrow{x}$ is the set of variables in t,*
- *$\overrightarrow{y}$ is the set of variables in t' that are not in t; thus, t' might have variables coming from the set $\overrightarrow{x} \cup \overrightarrow{y}$; however, it is not necessary that all variables in $\overrightarrow{x}$ occur in t',*
- *C is an equational condition, i.e., a conjunction of equations where all the variables involved are in $\overrightarrow{x}$,*

and π is a function assigning to each rewrite rule $r \in R$ a function

$$\pi_r : [\![C]\!] \rightarrow \textit{PFun}(\textit{CanGSubst}_E(\overrightarrow{y}), \mathcal{F}_r)$$

where:

- *$\textit{CanGSubst}_E(\overrightarrow{x})$ denotes the set of ground substitutions θ for the variables $\overrightarrow{x}$ which are in E-canonical form, i.e., cannot be further simplified by the equations E,*
- *$[\![C]\!]\{\mu \in \textit{CanGSubst}_E(\overrightarrow{x}) \mid E \vdash \mu(C)\}$ is the set of E-canonical ground substitutions for $\overrightarrow{x}$ satisfying the condition C,*
- *$\mathcal{F}_r$ is a σ-algebra structure on $\textit{CanGSubst}_E(\overrightarrow{y})$, and*
- *$\textit{PFun}(\textit{CanGSubst}_E(\overrightarrow{y}), \mathcal{F}_r)$ denotes the set of all probability measure functions on this σ-algebra.*

We denote a rule r together with its associated function π_r, by the notation

$$\texttt{crl [L]:}\, t(\overrightarrow{x}) \Rightarrow t'(\overrightarrow{x}, \overrightarrow{y})\ \ \texttt{if}\ C(\overrightarrow{x})\ \ \texttt{with probability}\ \overrightarrow{y} := \pi_r(\overrightarrow{x})$$

If the set $\textit{CanGSubst}_E(\overrightarrow{y})$ is empty because $\overrightarrow{y}$ is empty then $\pi_r(\overrightarrow{x})$ is said to define a trivial distribution*; this corresponds to the case of an ordinary rewrite rule with no probability. If $\overrightarrow{y}$ is nonempty but $\textit{CanGSubst}_E(\overrightarrow{y})$ is empty because there is no canonical substitution for some $y \in \overrightarrow{y}$ because the corresponding type is empty, then the rule is considered* erroneous *and will be disregarded in the semantics.*

A probabilistic rewrite theory has a natural operational semantics (see **??** for a more detailed exposition). Given a ground term u all the one-step rewrites with the rules in R are defined as expected:

- we need to find a subterm of u matched by one of the rules, say $r \in R$ with substitution θ and satisfying the rule's condition,
- we then need to choose a ground substitution ρ for the variables $\overrightarrow{y}$ in the given rule; of course, the choice of ρ should be made probabilistically, according to the probability distribution $\pi_r(\theta)$,
- in this way we obtain a *one-step probabilistic transition* $u \rightarrow v$, where we may assume that both u and v are in canonical form by the equations E, and where v is obtained from u by replacing the subterm which is the left-handside instance of the rule by the corresponding righthand side with $\overrightarrow{x}$ instantiated by θ and $\overrightarrow{y}$ instantiated by ρ, and then simplifying the resulting term with the equations E.

The *computations* of the system are then infinite paths of such one-step probabilistic transitions. The PMaude system can support symbolic simulation of such computations using the underlying Maude engine and a library of probability distributions, so that the substitutions ρ are obtained by sampling such distributions using a pseudo-random number generator.

Note however that, in general, a probabilistic rewrite theory $\mathcal{R}$ which we could execute this way *involves both probabilities and non-determinism*. The non-determinism is due to the fact that, in general, *different rules, possibly with different subterm positions and substitutions* could be applied to rewrite a given state u: the choice of what rule to apply, and where, and with which substitution is *non-deterministic*. It is only when such a choice has been made that probabilities come into the picture, namely for choosing the substitution ρ for the new variables $\overrightarrow{y}$. This gives our specifications a great flexibility to deal with different kinds of probabilistic-nondeterministic systems that have been considered in the literature, but this generality poses some limitations on the kinds of analysis that can be performed. In particular, for the kind of statistical model checking discussed in Section 6 that will be used to formally analyze our DoS-resistant TCP/IP protocol we need to assume that *all non-determinism has been eliminated* from our specification; that is, that at most one single rule, position, and substitution are possible to rewrite any given state.

What this amount to, in the specification of a concurrent system such as a network protocol is the *quantification of all non-determinism due to concurrency using probabilities*. This is natural for simulation purposes and can be accomplished by requiring the probabilistic rewrite theory to satisfy some simple requirements.

4.1 Sufficient condition for absence of un-quantified non-determinism in an object-oriented PMAUDE specification:

We will consider rewrite theories specifying concurrent actor-like objects and communicating by asynchronous message passing; this is particularly appropriate for communication protocols. In rewriting logic such systems (see [7] for a detailed exposition) have a distributed state that can be represented as a *multiset* of objects and messages, where we can assume that objects have a general record-like representation of the form: $\langle o : C \mid a_1 : v_1, \ldots a_1 : v_1 \rangle$, where o is the object's name, C its class, and the $a_i : v_i$ its corresponding attribute-value pairs in a given state. It is also easy to model in this way *real-time concurrent object systems*: one very simple way to model them is to include a global clock as a special object in the multiset of objects and messages. Rewrite rules in such a system will involve an object, a message, and the global time and will consume the message, change the object's state, and send messages to other objects. To deal with message delays and their probabilistic treatment we can represent messages as *scheduled objects* that are inactive until their associated delay has elapsed.

```
apmod SIMPLE-CLIENT-SERVER is
 protecting PMAUDE .
 including ACTORS .
 protecting NAT .

 vars t t1 t2 T : PosReal .
 vars C S : ActorName .
 vars N M : Nat .
 op counter:_ : Nat → Attribute .
 op server:_ : ActorName → Attribute .
 op total:_ : Nat → Attribute .
 op ctnt : Nat → Content .

 rl [send]: ⟨name: C | counter: N, server: S⟩ (C← empty) T ⇒
   ⟨name: C | counter: N+1, server: S⟩ [T+t1,(C← empty)] [T+t2,(S← ctnt(N))] T
        with probability t1:= EXPONENTIAL(2.0) and t2:= EXPONENTIAL(10.0) .

 rl[compute]: ⟨name: S | total: M⟩ (S← ctnt(N)) T ⇒ [T+t,⟨name: S | total: M+N⟩] T
        with probability t:= EXPONENTIAL(1.0) .

 rl[busy-drop]: [t,⟨name: S | total: M⟩] (S← ctnt(N)) ⇒ [t,⟨name: S | total: M⟩] .

 op init : → Config .
 op c : → ActorName .
 op s : → ActorName .
 eq init =  ⟨name: c | counter: 0, server: s⟩ ⟨name: s | total: 0⟩(c← empty) 0.0 .

endapm
```

Figure 1: A simple Client-Server model with exponential distribution on message sending delay and computation time by the server

Example 2 *We can illustrate systems of this kind by means of the client server example in Fig. 1. In the example, a client* `C` *continuously sends messages to a server* `S`*. The time interval between the messages is distributed exponentially with rate* 2.0*. The message sending of the client is triggered when it receives a self-sent message of the form* `(C← empty)`*. The delay associated with the message from the client to the server is distributed exponentially with rate* 10.0 *(see rule labelled* `send`*). The message contains a natural number which is incremented by 1 by the client, each time it sends a message. The server, when not busy, can receive a message and increment its attribute* `total` *by the number received in the message (see rule labelled* `compute`*). If the server is busy processing a message (computation time is exponentially distributed with rate* 1.0*), it drops any message it receives (see rule labelled* `busy-drop`*). Note that we can modify the rule* `busy-drop` *to allow the server actor to enqueue any message it receives when its is busy.*

The rule for sending a message by a client `C` *to a server* `S` *is labelled by* `send`*. The left hand side of the rule matches a fragment of the global state consisting of a client actor of the form* ⟨`name: C | counter: N, server: S`⟩*, a message of the form* `(C← empty)`*, and a global time of the form* `T`*. The rule states that the client* `C`*, on receiving an empty message, produces two messages: an empty message to itself and a message to a server, whose name is contained in its attribute* `server`*. Both the messages were produced as scheduled objects to represent that they are inactive till the delay time associated with the messages has elapsed. The delay times* t_1 *and* t_2 *are substituted probabilistically.*

Note that the model has no non-determinism. All non-determinism has been replaced by probabilistic choices. A model with no non-determinism is a key requirement for our statistical analysis technique briefly described in Section. 6. We next give a sufficient condition to ensure that a PMAUDE *specification has no non-determinism.*

1. The initial global state of the system or the initial configuration can have at most one non scheduled message.

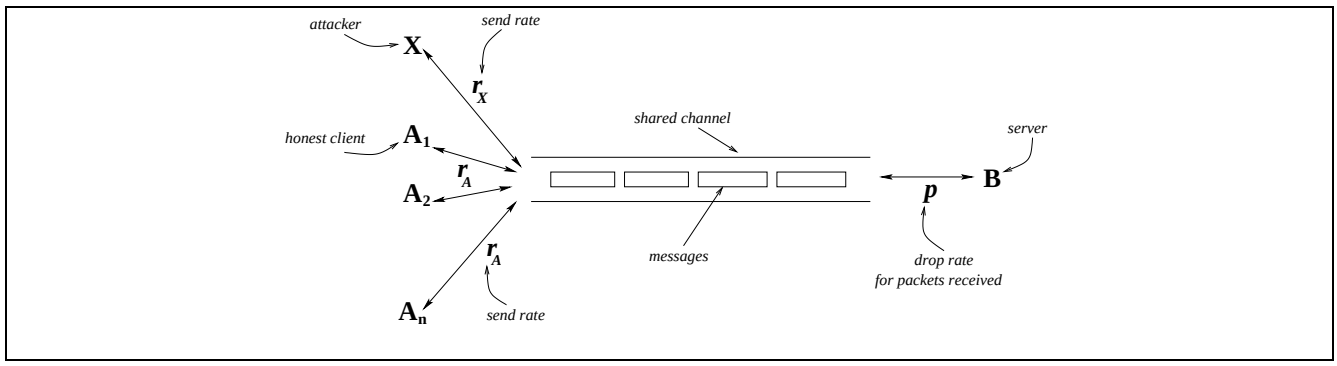

Figure 2: An instance of the TCP's 3-way handshake protocol.

2. The computation performed by any actor after receiving a message must have no un-quantified non-determinism; however, there may be probabilistic choices.

3. The messages produced by an actor in a particular computation (i.e. on receiving a message) can have at most one non scheduled message.

4. No two scheduled objects become active at the same global time. This is ensured by associating continuous probability distributions with message delays and computation time.

5 Probabilistic Rewrite Specification of DoS resistant 3-way handshaking in TCP

We now present an executable specification of TCP's 3-way handshake protocol in probabilistic rewriting logic. We consider a protocol instance composed of N honest clients $A_1, \ldots, A_N$ trying to establish a TCP connection with the server B, and a single attacker X that launches a SYN-flood attack on B (see Figure 2). The clients A_i transmit SYN requests to B at the rate r_A, while the attacker X floods spurious SYN requests at the rate r_X. These rates are assumed to be parameters of an exponential distribution from which the time for sending the next packet is sampled. The server B drops each packet it receives independently with probability p. Each message across the network is assumed to be subject to a transmission delay d, which we assume to be constant. Of course, these assumptions about the various distributions can be easily changed in the implementation that follows.

Each client A_i is modeled as an object with four attributes as follows.

```
<name: A(i) | isn:N, repcnt:s(CNT), sendto:BN, connected:false>
```

The attribute `isn` specifies the sequence number that is to be used for the TCP connection, `sendto` specifies the name of server B, `repcnt` specifies the number of times the SYN request is to be (re)transmitted in order to account for random dropping of packets at B, and `connected` specifies if the connection has been successfully established as yet. The attacker is modeled as an object with a single attribute as follows.

```
<name: XN | sendto: BN >
```

The server B is modeled as an object with two attributes.

```
<name: BN | isn: M , synlist: SC >
```

The attribute `isn` specifies the sequence number that B uses for the next connection request it receives, while `synlist` is the SYN cache that B maintains for the pending connection requests.

Following is the probabilistic rewrite rule that models the client A_i sending a SYN request.

```
rl <name:A(i) | isn:N, repcnt:s(CNT), sendto:BN, connected:false>
   (A(i)← poll) T
                ⇒
    <name: A(i) | isn:N, repcnt:CNT, sendto:BN, connected:false>
    [ d + T , (BN← SYN(A(i),N)) ]
    [ t + T , (A(i)← poll) ]   T
             with probability t := EXPONENTIAL(r_A)  .
```

We use special poll messages to control the rate at which A_i retransmits the SYN requests. Specifically, A_i repeatedly sends itself a poll message, and each time it receives a poll message it sends out a SYN request to B. The poll messages are subject to a random delay t that is sampled from the exponential distribution with parameter r_A. Specifically, the message is scheduled at time $t + T$, where T is the current global time. The net effect of this is that A_i sends SYN requests to B at rate r_A. Perhaps it is important to point out that the poll messages are not regular messages that are transmitted across the network; they have been introduced only for modeling purposes. Further, note that the approach of simply freezing A_i by scheduling it at time $T+t$ does not work since that would also prevent A_i from receiving any SYN+ACK messages that it may receive from B meanwhile. Finally, note that the replication count is decremented by one after the transmission of SYN message, and the message itself is scheduled with a delay d.

The rule for SYN flooding by the attacker is very similar, except that it uses randomly generated sequence numbers.

```
   rl <name: XN |  sendto: BN > (XN ←  poll) T
           ⇒  <name: XN | sendto: BN >  T
              [ d + T , (BN← SYN(XN,random(counter))) ]
              [ t + T , (XN← poll) ]
                    with probability t := EXPONENTIAL(r_X)  .
```

The following rule models the processing of SYN requests by the server B.

```
rl <name: BN | isn: M , synlist: SC > (BN← SYN(ANY,N)) T
    ⇒  if (drop? or size(SC) > SYN-CACHE-SIZE) then
          <name: BN | isn: M , synlist: SC > T
       else <name: BN | isn:s(M), synlist:add(SC,entry(ANY,M))>
            [ d + T , (ANY← SYN+ACK(BN,N,M)) ]
            [ TIMEOUT + T , (BN← tmout(entry(ANY,M))) ] T  fi
                  with probability drop? := BERNOULLI(p)  .
```

The random dropping of incoming messages is modeled by sampling from the Bernoulli distribution with the appropriate parameter p. Note that an incomming request can also be dropped if the SYN cache is full. If the cache is not full, for each request that is not dropped, the server B makes an entry for the request in the cache, and sends out a SYN+ACK message to the source of the request. A cache entry is of the form `entry(N,M)` where `N` is the name of the source which has requested a connection and `M` is the sequence number for the connection. Timing out of entries in the cache is modeled by locally sending a message to self that is scheduled after an interval of time equal to the timeout period. Here is the rule for removing timed out entries.

```
rl  <name: BN | isn: N, synlist: [s(SZ), (L1 entry(ANY,M) L2)]>
    (BN ←  tmout(entry(ANY,M)))
        ⇒   <name: BN | isn: N , synlist: [ SZ , (L1 L2) ] >  .
```

The first argument in the value of the `synlist` attribute above is the number of entries in the list, while the second argument is the actual list of entries. The rule for processing the SYN+ACK message at the clients is as follows.

```
rl <name: A(i) | isn:N, repcnt:CNT, sendto:BN, connected:false>
   (A(i)← SYN+ACK(BN,N,M))  T
        ⇒
   <name: A(i) | isn:N, repcnt:CNT, sendto:BN, connected:true>
           [ d + T , (BN← ACK(A(i),M)) ]  T  .
```

The rule is self-explanatory; the only significant point to be noted is that the attribute `connected` is set to true after processing the SYN+ACK message. Since the clients replicate their requests to account for random dropping of packets at the server, it is possible for them to receive a SYN+ACK message for a connection that has already been established. Such SYN+ACK messages are simply ignored as follows.

```
rl <name: A(i) | isn:N, repcnt:CNT, sendto:BN, connected:true>
   (A(i)← SYN+ACK(BN,N,M))
       ⇒
   <name: A(i) | isn:N, repcnt:CNT, sendto:BN, connected:true> .
```

In contrast to the honest clients, the attacker ignores all the SYN+ACK messages that it receives from the server B.

```
    rl <name: XN | sendto: BN > (XN ← SYN+ACK(BN,N,M))
          ⇒       <name: XN | sendto: BN >  .
```

Finally, the initial configuration of the system is

```
 < name: XN | ... >   [ t1 , < name: A(1) | ... > ]
      [ t2 , < name: A(2) | ... > ] ...
 [tn , < name: A(N) | ... > ] < name: BN | ... >
```

where $t_1, \ldots, t_n$ are all distinct and positive. Note that the since all the clients are scheduled at different times, it follows from our discussion in Section 4 that the system does not contain any un-quantified non-determinism, which is essential for statistical analysis to be possible.

6 Query Language for Analysis

To query various quantitative aspects of a probabilistic model, we introduce a query language called *Quantitative Temporal Expressions* (or QUATEX in short). The language is mainly motivated by probabilistic computation tree logic (PCTL) and EAGLE. In QUATEX, some example queries that can be encoded are as follows:

1. What is the expected number of clients that get connected to B out of 100 clients?
2. What is the probability that a client got connected with B within 10 seconds since it initiated the connection request?

We next introduce the notations that we will use to describe the syntax and the semantics of QUATEX followed by a few motivating examples. Then we describe the language formally, along with an example query that we have used to investigate if the DoS free 3-way TCP/IP handshaking protocol model meets our requirements. The results of our query on various parameters are given in Section. **??**.

We assume that an execution path is an infinite sequence

$$\pi = s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow \cdots$$

where s_0 is the unique initial state of the system, typically a term of sort `Config` representing the initial global state, s_i is the state of the system after the i^{th} computation step. If the k^{th} state of this sequence cannot be rewritten any further (i.e. is absorbing), then $s_i = s_k$ for all $i \geq k$.

We denote the i^{th} state in an execution path π by $\pi[i] = s_i$. The denote the suffix of a path π starting at the i^{th} state by $\pi^{(i)} = s_i \rightarrow s_{i+1} \rightarrow s_{i+2} \rightarrow \cdots$. We let $Path(s)$ be the set of execution paths starting at state s. Note that, because the samples are generated through discrete-events simulation of a PMAUDE model with no non-determinism, $Path(s)$ is a measurable set and has an associated probability measure. This is essential to compute the expected value of a path expression from a given state.

6.1 QUATEX through Examples

The language QUATEX, which is designed to query various quantitative aspects of a probabilistic model, allows us to write temporal query expressions like temporal formulas in a temporal logic. It supports a framework for parameterized recursive operator definitions using a few primitive non-temporal operators and a temporal operator. For example, suppose we want to make a query over an execution path that *"Whether the client $A(0)$ gets connected with B within 100 time units."* For this we write the following query in QUATEX

```
IfConnectedInTime()(t) =
   if t > time() then false
                 else if connected() then true
                                     else ◯ (IfConnectedInTime()(t)) fi fi;
IfConnectedInTime()(time() + 100)
```

The first four lines of the query define the operator `IfConnectedInTime()(t)`, which returns true, if along an execution path $A(0)$ gets connected to B in time t. The state function *time()* returns the global time associated with the state; the state function *connected*(t) returns true, if in the state, $A(0)$ gets connected with B and returns false otherwise. The fifth line of the query writes a path expression, which returns true, if by the global time *time*$() + 100$, $A(0)$ gets connected with B.

The above expression is a simple formula that can also be expressed in metric temporal logic. Let us complicate the example a bit by querying probabilities. Note that the above query is a query about a path. Now suppose that we want to know *"the probability that along a random path from a state, the client $A(0)$ gets connected with B within* 100 *time units."* This can be written as the following query

```
NumConnectedInTime()(t) =
   if t > time() then 0
                 else if connected() then 1
                                     else ◯ (NumConnectedInTime()(t)) fi fi;
E[NumConnectedInTime()(time() + 100)]
```

In this query, we define the operator `NumConnectedInTime()(t)`, which is similar to `IfConnectedInTime()(t)`, except that `NumConnectedInTime()(t)` returns 1 when `IfConnectedInTime()(t)` returns true and `NumConnectedInTime()(t)` returns 0 when `IfConnectedInTime()(t)` returns false. Then the state query at the fifth line returns the expected number of times $A(0)$ gets connected to B within 100 time units along a random path from a given state. This number lies in $[0, 1]$ since along a random path either $A(0)$ gets connected to B within 100 time units or $A(0)$ does not get connected to B within 100 time units. In fact, this expected value is equal to the probability that along a random path from a state, the client $A(0)$ gets connected with B within 100 time units.

A further rich query that is interesting to our probabilistic model is as follows

$$
\begin{array}{rcl}
Q & ::= & D\ \ \mathbb{SE} \\
D & ::= & \text{set of } Defn \\
Defn & ::= & N(x_1, \ldots, x_n)(y_1, \ldots, y_m) = \mathbb{PE}; \\
\mathbb{SE} & ::= & c \mid f \mid F(\mathbb{SE}_1, \ldots, \mathbb{SE}_k) \mid \mathbf{E[}\mathbb{PE}\mathbf{]} \\
\mathbb{PE} & ::= & \mathbb{SE} \mid F(\mathbb{PE}_1, \ldots, \mathbb{PE}_k) \mid N(\mathbb{PE}_1, \ldots, \mathbb{PE}_n)(\mathbb{SE}_1, \ldots, \mathbb{SE}_m) \\
 & & \mid \underline{\text{if}}\ \mathbb{SE}\ \underline{\text{then}}\ \mathbb{PE}_1\ \underline{\text{else}}\ \mathbb{PE}_2\ \underline{\text{fi}} \mid \bigcirc \mathbb{PE} \mid x_i
\end{array}
$$

Figure 3: Syntax of QUATEX

$$
\begin{array}{l}
\texttt{ConnectedInTime}()(t, count) = \\
\quad \underline{\text{if}}\ t > time()\ \underline{\text{then}}\ count \\
\qquad\qquad \underline{\text{else if}}\ anyConnected()\ \underline{\text{then}}\ \bigcirc(\texttt{ConnectedInTime}()(t, 1 + count)) \\
\qquad\qquad\qquad\qquad \underline{\text{else}}\ \bigcirc(\texttt{ConnectedInTime}()(t, count))\ \underline{\text{fi}}\ \underline{\text{fi}}; \\
\mathbf{E[}\texttt{ConnectedInTime}()(time() + 100, 0)\mathbf{]}
\end{array}
$$

In this query, the state function $anyConnected()$ returns true if any client $A(i)$ gets connected to B in the state. We assume that in a given execution path, at any state, at most one client gets connected to B, which is true with our probabilistic model. We will use a simpler variant of this query in our experiments.

6.2 Syntax of QUATEX

The syntax of QUATEX is given in Fig. 3. A query in QUATEX consists of a set of definitions D followed by a state query expression $\mathbb{SE}$. In QUATEX, we distinguish between two kinds of expressions, namely, *path expressions* (denoted by $\mathbb{PE}$) and *state expressions* (denoted by $\mathbb{SE}$); a path expression is interpreted over an execution path and a state expression is interpreted over a state. A definition $Defn \in D$ consists of a definition of a *path operator*. A path operator definition consists of a name N and two sets of formal parameters on the left-hand side, and a path expression on the right-hand side. The first set of formal parameters denote the *non-freeze formal parameters* and the second set of parameters denote the *freeze formal parameters*. When using an operator in a path expression, the non-freeze formal parameters are substituted by path expressions and the freeze formal parameters are replaced by state expressions. A state expression can be a constant c, a function f that maps a state to a concrete value, a k-ary function mapping k state expressions to a state expression, or the expectation $\mathbf{E[}\mathbb{PE}\mathbf{]}$ denoting the expected value of the path expression $\mathbb{PE}$. A path expression can be a state expression, an application of a path operator already defined in D, a conditional expression $\underline{\text{if}}\ \mathbb{SE}\ \underline{\text{then}}\ \mathbb{PE}_1\ \underline{\text{else}}\ \mathbb{PE}_2\ \underline{\text{fi}}$, or the unary temporal path operator $\bigcirc$ (standing for next). We assume that expressions are properly typed. Typically, these types would be `integer`, `real`, `boolean` etc. The condition $\mathbb{SE}$ in the expression $\underline{\text{if}}\ \mathbb{SE}\ \underline{\text{then}}\ \mathbb{PE}_1\ \underline{\text{else}}\ \mathbb{PE}_2\ \underline{\text{fi}}$ must have the type `boolean`. The path expression $\mathbb{PE}$ in the expression $\mathbf{E[}\mathbb{PE}\mathbf{]}$ must be of type `real`. We also assume that expressions of type `integer` can be coerced to the `real` type.

6.3 Semantics of QUATEX

Next, we give the semantics of a subset of query expressions that can be written in QUATEX. In this subclass, we put the restriction that the value of a path expression $\mathbb{PE}$ that appears in any expression $\mathbf{E[}\mathbb{PE}\mathbf{]}$ can be determined from a finite prefix of an execution path. We call such path expressions *bounded* path expressions. The semantics is given in Fig. 4. $(s)[\![\mathbb{SE}]\!]_D$ is the value of the state expression $\mathbb{SE}$ in the state s. Similarly, $(s)[\![\mathbb{PE}]\!]_D$ is the value of the path expression $\mathbb{PE}$ over the path π. Note that if the value of a bounded path expression can be computed from a finite prefix π_{fin} of an execution path π, then the evaluations of the path expression over all execution paths

$$
\begin{array}{rl}
(s)[\![c]\!]_D & = c \\
(s)[\![f]\!]_D & = f(s) \\
(s)[\![F(\mathbb{SE}_1, \ldots, \mathbb{SE}_k)]\!]_D & = F((s)[\![\mathbb{SE}_1]\!]_D, \ldots, (s)[\![\mathbb{SE}_k]\!]_D) \\
(s)[\![\mathbf{E[}\mathbb{PE}\mathbf{]}]\!]_D & = \mathbf{E[}(\pi)[\![\mathbb{PE}]\!]_D \mid \pi \in Paths(s)\mathbf{]} \\
(\pi)[\![F(\mathbb{PE}_1, \ldots, \mathbb{PE}_k)]\!]_D & = F((\pi)[\![\mathbb{PE}_1]\!]_D, \ldots, (\pi)[\![\mathbb{PE}_k]\!]_D) \\
(\pi)[\![\underline{\text{if}}\ \mathbb{SE}\ \underline{\text{then}}\ \mathbb{PE}_1\ \underline{\text{else}}\ \mathbb{PE}_2\ \underline{\text{fi}}]\!]_D & = \text{ if } (\pi[0])[\![\mathbb{SE}]\!]_D == \text{true} \\
 & \text{ then } (\pi)[\![\mathbb{PE}_1]\!]_D \\
 & \text{ else } (\pi)[\![\mathbb{PE}_2]\!]_D \\
(\pi)[\![\bigcirc \mathbb{PE}]\!]_D & = (\pi^{(1)})[\![\mathbb{PE}]\!]_D \\
(\pi)[\![N(\mathbb{PE}_1, \ldots, \mathbb{PE}_n)(\mathbb{SE}_1, \ldots, \mathbb{SE}_m)]\!]_D & \\
= B[x_1 \mapsto \mathbb{PE}_1, \ldots, x_n \mapsto \mathbb{PE}_n & , y_1 \mapsto (\pi[0])[\![\mathbb{SE}_1]\!]_D, \ldots, y_m \mapsto (\pi[0])[\![\mathbb{SE}_m]\!]_D] \\
\text{where} & N(x_1, \ldots, x_n)(y_1, \ldots, y_m) = B \in D
\end{array}
$$

Figure 4: Semantics of QUATEX

having the common prefix π_{fin} are the same. Since a finite prefix of a path defines a basic cylinder set (i.e. a set containing all paths having the common prefix) having an associated probability measure, we can compute the expected value of a bounded path expression over a random path from a given state. In our analysis tool, we estimate the expected value through simulation instead of calculating it exactly based on the underlying probability distributions of the model.

6.4 Statistical Evaluation of a QUATEX Expression

Given a probabilistic model and a QUATEX expression, we evaluate the expression at the initial state of the model. The evaluation of all path and state expressions, except the expectation expression, is straightforward and follows directly from the semantics. However, the evaluation of an expression of the form **E[**$\mathbb{PE}$**]** cannot be done directly for a complex probabilistic model. Rather, we use discrete-event simulation to estimate **E[**$\mathbb{PE}$**]** with certain given confidence and certain given confidence interval. Specifically, we assume that we are given three parameters with each expression **E[**$\mathbb{PE}$**]**, namely l, u, α, where l and u are the lower and upper bounds, respectively, of a α-confidence interval of the estimated value of **E[**$\mathbb{PE}$**]**. Let X be random variable giving the value of the expression $\mathbb{PE}$ along a random path π from a state s. Then $(s)[\![\mathbf{E[}\mathbb{PE}\mathbf{]}]\!]_D = \mathbf{E[}X\mathbf{]}$. Let $X_1, \ldots, X_n$ be n random variables having same distribution as X. We calculate n such that $\mathbf{Prob}[l \leq \sum_{i \in [1,n]} X_i/n \leq u] = \alpha$ holds. Once we know the number of samples n, we estimate **E[**X**]** by drawing n samples $x_1, \ldots, x_n$ of X. The estimated value of $(s)[\![\mathbf{E[}\mathbb{PE}\mathbf{]}]\!]_D$ is then given by $\sum_{i \in [1,n]} x_i/n$.

6.5 Experimental Evaluation

For our experiments, we evaluated the following expression with different values for r_X, the attacker rate.

```
CountConnected() = if completed() then count() else ◯ (CountConnected()) fi;
E[CountConnected()]
```

In this expression, $completed()$ is true in a state if all A_i's have either sent their all SYN packets or got connected with B. $count()$ in a state returns the number of A_i's that got connected to B.

7 Conclusions

Our study demonstrates that we are able to express and prove key properties, but performance limitations of the automated system in our current formulation require us to use parameters more limited than those that arise in practice. Addressing these efficiency limitations and showing the general invariance property described above remain future work objectives.

References

[1] Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. *Lecture Notes in Computer Science*, 1462, 1998.

[2] Danny Dolev and Andrew C. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.

[3] Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh Venkatesh. Dos protection for reliably authenticated broadcast. In Mike Reiter and Dan Boneh, editors, *Network and Distributed System Security (NDSS '04)*. Internet Society, February 2004.

[4] Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. Probabilistic rewrite theories: Unifying models, logics and tools. Technical Report UIUCDCS-R-2003-2347, University of Illinois at Urbana-Champaign, May 2003.

[5] Nirman Kumar, Koushik Sen, José Meseguer, and Gul Agha. A rewriting based model for probabilistic distributed object systems. In *Proceedings of 6th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'03)*, volume 2884 of *Lecture Notes in Computer Science*, pages 32–46. Springer, 2003.

[6] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theoretical Computer Science*, 96(1):73–155, 1992.

[7] José Meseguer. A logical theory of concurrent objects and its realization in the Maude language. In *Research Directions in Concurrent Object-Oriented Programming*, pages 314–390. MIT Press, 1993.