Probabilistic Modeling and Analysis of DoS Protection for the ASV Protocol

Musab AlTurki[1] José Meseguer[2] Carl A. Gunter

Department of Computer Science
University of Illinois at Urbana Champaign
Urbana, USA

Abstract

The Adaptive Selective Verification (ASV) protocol was recently proposed as an effective and efficient DoS countermeasure within the shared channel model, in which clients and attackers probabilistically share communication bandwidth with the server. ASV has been manually shown to satisfy some desirable availability and bandwidth consumption properties. Due to the probabilistic nature of the protocol and its underlying attacker model, it is intrinsically difficult to build a faithful model of the protocol with which one may automatically verify its properties. This paper fills the gap between manual analysis and simulation-based experimental analysis of ASV, through automated formal analysis. We describe a formal model of ASV using probabilistic rewrite theories, implemented in a probabilistic extension of Maude, and show how it can be used to formally verify various characteristics of ASV through automated statistical quantitative model checking analysis techniques. In particular, we formally verify ASV's connection confidence theorem and a slightly more general bandwidth consumption theorem of ASV. This is followed by a statistical comparison of ASV with non-adaptive selective verification protocols. We conclude with remarks on possible further development and future work.

Keywords: Adaptive Selective Verification, DoS, Probabilistic Rewrite Theories, Formal Statistical Analysis, Rewriting Logic, Maude

1 Introduction

Protocol security specification and verification involves meeting quite different types of requirements. The most thoroughly studied requirements are those of secrecy, authenticity and integrity, which are typically expressed as safety properties, and for which a rich body of work and a wide range of formal analysis tools are available. To verify these properties, a very powerful Dolev-Yao-like attacker, who has complete control of the netwok, is assumed, so that formal requirements of a secrecy, authenticity and integrity nature are verified for a given protocol in the presence of such an attacker.

For security requirements such as *availability*, considerably less work has been done (but see the papers [18,24,17] and our discussion of related work in Section 6).

[1] Email: alturki@cs.uiuc.edu
[2] Email: meseguer@cs.uiuc.edu

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

Consider, for example, a protocol availability requirement such as *being resistant to a Denial of Service (DoS) attack*. Such a requirement cannot be dealt with within the standard frameworks and tools for protocol specification and verification developed to verify secrecy, authenticity and integrity because of the following challenges:

(i) *New protocol models are needed.* For example, many DoS defense mechanisms are *probabilistic* in nature, whereas usual cryptographic protocol formal models used for secrecy, authenticity and integrity are not.

(ii) *New attacker models are needed.* A Dolev-Yao attacker is both *too powerful* to faithfully represent a DoS attack, since complete control of the network means by definition that it can *always* succeed in such an attack, and *too coarse*, since to faithfully model a DoS attack we need *quantitative information* on the estimated amount of resources available to the attacker, something not available at all in a Dolev-Yao-like attacker model.

(iii) *What counts as a DoS attack requires precise modeling.* That is, how to properly formalize availability requirements themselves is not an obvious matter. To begin with, unlike security properties such as secrecy, which do not allow of degrees (either the protocol ensures secrecy or it does not), it is perfectly reasonable to ask whether a given DoS protection mechanism is *more effective* than another one. The point is that we must now move from Boolean-valued (true or false) requirements such as secrecy to *quantitative* requirements such as effectiveness in protecting against DoS.

(iv) *Formal verification techniques now need to deal with real-time, probabilities, and quantitative properties.* This means that the standard methods used for verifying secrecy, authenticity and integrity properties, such as invariants and reachability analysis, standard temporal logic and model checking, or various specialized theorem proving schemes may not be directly usable in the realm of availability properties.

In this paper we further develop and apply to a novel DoS protection mechanism recently proposed in [14] a new rewriting-based approach to the formal specification and verification of security properties such as DoS-resistance initiated in [2] and further applied to other protocol properties beyond security in [13,15]. This rewriting-based approach directly addresses and answers the above-mentioned challenges (i)–(iv).

The way our approach addresses challenges (i)–(ii) is by modeling both the (typically probabilistic and real-time) protocol and the DoS attacker by means of *probabilistic rewrite rules* [3], that is, rewrite rules whose right-hand side result is not uniquely determined by the matching substitution for the rule's left-hand side, but depends instead on a *probability distribution*.

The way challenge (iii) is answered is by generalizing properties from Boolean-valued ones (such as standard temporal logic formulas expressing, say, a secrecy invariant) to *quantitative, real-valued* formulas, which can measure various quantities and performance values in our system. This is done by using the QuaTEx quantitative, probabilistic temporal logic defined in [3]. Finally, challenge (iv) is answered by formally verifying quantitative formal requirements expressed in QuaTEx about a protocol formally specified as a probabilisitc rewrite theory by *statistical*

model checking in the sense of [22], in which the model checker, once a QuaTEx formula and a desired degree of statistical confidence are specified, requests a sufficiently large number of Monte Carlo simulations of the rewrite theory allowing verification of the QuaTEx formula. In our experiments, the Monte Carlo simulations are performed using the Maude rewrite engine, and the statistical model checking of the QuaTEx properties is preformed by the VeStA tool [22,3], which has a Maude interface.

We view the form of formal specification and automated formal verification supported by the rewriting-based approach just described as providing a useful middle ground for the analysis of protocol availability properties such as DoS resistance between the other two methods currently available, namely: (i) by-hand mathematical analysis of the protocol to establish some of its properties analytically, typically under somewhat idealized assumptions; and (ii) analysis based on standard simulation techniques and tools.

As we show for the Adaptive Selective Verification (ASV) DoS-resistant protocol [14] that we specify and formally verify in this paper, our techniques complement and add significant analytic power to those in (i)–(ii). Specifically, we confirm by automatic statistical model checking techniques analytic results proved by hand in [14]. And we also confirm, wich the much stronger level of assurance provided by statitical model checking, various protocol properties suggested by the simulation analyses reported as well in [14]. In this way, a considerably higher level of assurance can be gained for both analytical properties proved by hand, and for properties suggested by simulation analyses. Furthermore, this assurance can be gained for scenarios and realistic deployment conditions too complex to be amenable to by-hand mathematical analysis.

2 Background

2.1 Probabilistic Rewrite Theories

Rewriting logic is an expressive formalism that unifies in a natural way different concurrency models [19]. The unit of specification in the logic is a rewrite theory, which gives a formal description of a concurrent system including its static state structure and dynamic behavior. A *rewrite theory* $\mathcal{R} = (\Sigma, E, R)$ consists of a signature Σ that declares the sorts and operators to be used in the system specification, a set E of equations and memberships on Σ describing algebraically the state structure of the system, and a set R of rewrite rules specifying the computational behavior of the system. A rewrite rule has the following form:

$$(\forall \overrightarrow{x}, \overrightarrow{y})\ r : t(\overrightarrow{x}) \longrightarrow t'(\overrightarrow{y}) \text{ if } C(\overrightarrow{x})$$

where the set of variables $\overrightarrow{y}$ are typically included in $\overrightarrow{x}$, r is a label, and C is a conjunction of equational or rewrite conditions. The operational meaning of such a rewrite rule is that if there exists a substitution θ such that $\theta(t)$ matches a subterm s in the system, and $\theta(C)$ is satisfied, then s may rewrite to $\theta(t')$. A rewrite rule, therefore, gives a general pattern for a possible change or transition in the state of a concurrent system (See [4] for a detailed account of generalized rewrite theories).

Many realistic systems are probabilistic in nature, and thus cannot be directly specified as rewrite theories of the form described above. For this purpose, *probabilistic rewrite theories*, which extend regular rewrite theories with probabilistic rules, have been developed [21]. Assuming $\overrightarrow{x}$ and $\overrightarrow{y}$ are disjoint, a probabilistic rewrite rule is of the following form:

$$(\forall \overrightarrow{x}, \overrightarrow{y})\ r : t(\overrightarrow{x}) \longrightarrow t'(\overrightarrow{x}, \overrightarrow{y}) \text{ if } C(\overrightarrow{x}) \text{ with probability } \overrightarrow{y} := \pi(\overrightarrow{x})$$

A probabilistic rule introduces on its right-hand side term new variables $\overrightarrow{y}$, the values of which depend on a probability distribution function π parametrized by $\theta(\overrightarrow{x})$, where θ is a matching substitution satisfying the condition C.

To illustrate the operational meaning of a probabilistic rewrite rule, consider the following rule, which is borrowed from the battery-operated clock example in [6]:

$$\begin{aligned} \mathit{clock}(t, c) \longrightarrow \text{if } B \text{ then } \mathit{clock}(t + 1, c - (c/1000.0)) \text{ else } \mathit{broken}(t, c) \\ \text{with probability } B := \mathit{Bernoulli}(c/1000.0) \end{aligned}$$

The rule specifies how a clock transitions to its next state, which is either a regular operational state $\mathit{clock}(t, c)$ or a broken state $\mathit{broken}(t, c)$, with t the current time and c the current battery charge. As the clock ticks, its battery charge decreases. Whether the clock transitions to an operational or a broken state depends on the outcome of a Bernoulli trial with success probability of $c/1000.0$ used by the new variable B. Since the probability of success in a Bernoulli trial is proportional to the current battery charge, the clock will have a higher chance of failing as time elapses and the battery charge decreases.

In general, probabilistic rewrite theories can model different probabilistic systems with discrete or continuous probability distribution functions. Furthermore, they can express models involving both probabilistic and non-deterministic features. The reader is referred to [3] for a rigorous definition of probabilistic rewrite theories.

An executable implementation of probabilistic rewrite theories is provided by Maude [6], an efficient rewriting logic engine. Probabilistic theories are supported in Maude by essentially sampling from probability distributions using a pseudo-random number generator function `random(s)`, with `s` a seed, and a counter function `counter` that rewrites to the next natural number with an internal strategy. Due to space limitations, we refer the reader to [3] for the details.

2.2 Statistical Analysis using VeStA

Through their implementation in Maude, probabilistic rewrite theories can be analyzed statistically using VeStA 2.0 [23]. In order to apply statistical model checking techniques to such theories, a rich, quantitative temporal logic for specifying real-valued quantities is needed. For this reason, we use *Quantitative Temporal Expressions* (QuaTEx) specified in [3]. In QuaTEx, real-valued state and path functions are used instead of boolean state and path predicates to quantitatively specify properties about probabilistic models. QuaTEx supports parameterized recursive function declarations, an if-then-else construct, and a *next* operator $\bigcirc$, which when combined, allow for an expressive language for real-valued temporal properties. Example QuaTEx expressions appear in Section 5 below. For a detailed account of QuaTEx's syntax and semantics, the reader is referred to [3].

QuaTEx is supported by the VeStA tool, which has an interface to Maude. Thus, given a QuaTEx path expression and a Maude module specifying a probabilistic rewrite theory, statistical quantitative analysis is performed by evaluating the expected value of the path expression against computation paths obtained by Monte Carlo simulations. VeStA's statistical model checking algorithm evaluates these quantities with respect to two confidence parameters: α and δ, which are user-provided. Specifically, when computing the expected value of a QuaTEx expression E, the output of the algorithm is a $(1-\alpha)100\%$ confidence interval with size bounded above by δ for the random variable associated to E. VeStA will perform as many Monte Carlo simulations as needed to arrive at a confidence interval with the given parameters

A requirement for VeStA's statistical model checking algorithm to be correctly applicable is that the probabilistic model has to avoid any unquantified non-determinism [3]. That is, a state in the model should only evolve into two or more possible different states if and only if each possibility has an associated probability of taking place. In probabilistic rewrite theories, this means that there should not exist a non-deterministic choice of rules to apply, or terms or subterms to match. This requirement can be easily satisfied, however, and [3] lists simple sufficient conditions for the satisfaction of this property in the context of actor-based rewrite specifications. We adopt a specification style originally developed by [2] that satisfies these conditions, and discuss it in Section 4.2 in some detail.

3 Tha Shared Channel Model and the ASV Protocol

As was discussed above, the Dolev-Yao model is inappropriate for the analysis of DoS properties. A more appropriate model for this purpose is the *shared channel model* [8]. In this model, attackers and legitimate clients probabilistically share a communication channel to the server. Unlike the Dolev-Yao model, an attacker does not have full control over the channel. However, an attacker may replay modified (or faked) versions of previously seen legitimate packets at some maximum rate, specified as a parameter in the model.

The Adaptive Selective Verification (ASV) protocol [14] is a cost-based, DoS-resistant protocol in which bandwidth is the currency. ASV assumes the shared channel model as its underlying attack model. The key idea of the protocol is for clients to spend more bandwidth to compete with attacker bandwidth usage, and for the server to selectively process incoming requests. A client attempts to adapt to the current level of attack by exponentially replicating its requests (up to a threshold) as the sensed severity of attack increases. The server implements a reservoir sampling algorithm to collect a random sample of the incoming requests and process them at its mean processing rate.

More precisely, we denote the server's mean processing rate by S, and the server and client timeout periods by T_s and T_c, respectively. The current client request rate is denoted by ρ, with the assumption that $\rho \in [\rho_{\min}, \rho_{\max}]$. Similarly, The current attack rate is denoted by $\alpha \in [\alpha_{\min}, \alpha_{\max}]$. The client replication threshold is specified by the protocol as 2^J, where $J = \lceil \log(\alpha_{\max}/\rho_{\min})/\log(2) \rceil$ (called the

retrial span). Under the ASV protocol, the server and clients behave as follows:

Client. When a client first arrives, it initializes its *retries count* $j \leftarrow 0$, and then sends a single copy of its request to the server. If the client receives and acknowledgement within T_c time units, the client is accepted and quits. Otherwise, it increments j ($j \leftarrow j + 1$) and then sends 2^j copies of its request to the server. This process is repeated until either an acknowledgement is received, or the retrial threshold is reached, i.e. $j > J$. In the latter case, the client fails and quits.

Server. The server first initializes its *window count* $k \leftarrow 1$, and its *request count* $j \leftarrow \lfloor ST_s \rfloor + 1$. During the kth window, the server attempts to collect the first $\lfloor ST_s \rfloor$ incoming requests. At this point, there are two cases:

(i) If it times out before the reservoir is filled, the server sends an acknowledgement for each request in the reservoir, empties its reservoir, increments its window count k, and repeats the process for the next window.

(ii) If the reservoir is filled before a timeout occurs, the server places the jth incoming request in the reservoir with probability $p \leftarrow \lfloor ST_s \rfloor / j$ and discards it with probability $1 - p$. If the request is to be placed in the reservoir, the server replaces a request in the reservoir selected uniformly at random with the accepted request. The server increments its request count and processes the next request in the same way. This step is repeated until the server times out (signaling the end of the current window). Once a timeout occurs, the server empties its reservoir after acknowledging its requests, increments k, resets j to $\lfloor ST_s \rfloor + 1$, and the whole process is repeated for the next window.

Despite the simplicity of the protocol, analyzing it manually turns out to be a fairly demanding task [14]. In the following, we describe a model of the protocol that enables automatic statistical verification of its properties and analyze the results.

4 Formal Modeling of ASV in Maude

Our model of the ASV protocol is based on a representation of actors in rewriting logic [3], which is built using the logic's Maude object-based programming framework [6]. Within this framework, the system state is represented by a *configuration*, which is a soup of objects and messages built using an associative and commutative juxtaposition operator with an identity element *none*. The exact forms of objects and messages comprising a configuration are application-specific. In our specification, an object is a term of the form $\langle name : Oid \mid A \rangle$, where Oid is a unique object identifier, and A is a set of attribute-value pairs of the form $attr_i : val_i$, representing the state of the object. A message is a term of the form $Oid \leftarrow C$, with Oid the target object id and C the contents of the message. Within a configuration, asynchronous message passing can be modeled by rules that rewrite a sub-configuration consisting of a message and its target object to another sub-configuration consisting of an updated version of the object and possibly one or more new messages:

$$\langle name : Oid \mid A \rangle (Oid \leftarrow C) \longrightarrow \langle name : Oid \mid A' \rangle (Oid_1 \leftarrow C_1) \ldots (Oid_n \leftarrow C_n)$$

We begin by describing the main entities of the model below.

4.1 Model Components

The ASV model specifies a simple topology in which multiple client objects and an attacker object share a communication channel with the server. The three main classes of objects, namely *server*, *client*, and *attacker* objects, are described below.

Server. The server object maintains three attributes: a buffer attribute *reqlist* that holds incoming requests between server timeouts, a REQ packet count attribute *reqcnt* that is used in determining the probability of accepting an incoming REQ when the buffer is full, and a counter for incoming legitimate client requests.

$$\langle name : SN \mid reqlist : L, reqcnt : R, cnt : I \rangle$$

A server object, with id SN, accepts two kinds of messages: (i) a connection request message of the form $SN \leftarrow REQ(Oid)$, with Oid the object id of the client or attacker object which initiated the request, and (ii) an internal timeout message $SN \leftarrow timeout$, which signals a new server time window. Self-addressed messages are commonly used in actor-based systems to schedule internal events [3].

Client. A client object maintains the current number of retries and the current replication count which are required to implement the adaptive protocol. It also contains four other attributes: a reference to the server with which the client communicates, the arrival and service times of the client, and a status variable indicating whether the client is waiting, has connected, or failed to establish a connection.

$$\langle name : CN | server : SN, retries : J, repcnt : K, atime : T, stime : T', status : Q \rangle$$

A Client, named CN, accepts two kinds of messages: (i) a connection acknowledgement message of the form $CN \leftarrow ACK(SN)$ from the server identified by SN, and (ii) an internal message of the form $CN \leftarrow poll$ that schedules client timeouts.

Attacker. The attacker object is a simple object that maintains only one attribute, namely, a reference to the server object on which the attack is to be carried out: $\langle name : X \mid server : SN \rangle$. An attacker object accepts the same kinds of messages a client object accepts although it exhibits a different behavior upon receiving them, as we shall see in Section 4.2.

In addition to the three classes above, a fourth, auxiliary class of objects is the *generator* class, of which a single object is used in the configuration to model new clients coming in at a rate ρS, with S the server's mean processing rate. The generator object maintains a counter for generating fresh client identifiers and the name of the server to which generated clients will attempt to connect: $\langle name : G \mid cnt : I, server : SN \rangle$. The generator object G uses a self-addressed message of the form $G \leftarrow spawn$ to schedule the creation of the next client object every $1/\rho S$ time units.

4.2 Model Dynamics

Beside the objects described above, a configuration contains a few other components that support its dynamic behavior. As was discussed in Section 2.2, in order for it to properly support statistical model checking, the specification must avoid any unquantified non-determinism. We adopt a specification style originally developed by [2] to support this requirement. In this style, the configuration uses a *scheduler* that stores a list of scheduled messages to be made active and ready for consumption

S	Mean server processing rate	J	Client retrial span
T_s	Server timeout (window size)	*initDelay*	Initial client generation delay
T_c	Client timeout	*delay*	Message transmission delay
$\rho_{\min}, \rho_{\max}$	Min and max client request rates	*drop*	Message drop probability (network loss rate)
$\alpha_{\min}, \alpha_{\max}$	Min and max attack rates	*Limit*	Duration of a sample run

Fig. 1. Model parameters

by the appropriate object in the configuration. The scheduler, which is a term of the form $\{t \mid L\}$, with t the current global clock of the configuration and L a list of scheduled messages, enforces the fact that only one message can be made active at any given instant of time. A *scheduled message* is a term of the form $[t, m, d]$, where t is the time at which the message is scheduled for delivery, m is the message itself, and d is a drop flag that is used when modeling lossy channels to indicate whether the message is to be dropped or kept.

The use of the scheduler object also provides a mechanism for managing the elapse of time and its effect on the configuration. This is achieved with the help of an operator *mytick*, which is used to extract the next message from the scheduler and update the current global time accordingly. The specification uses this operator repeatedly to advance time and process successive messages from the scheduler until the given time limit, specified by a parameter in the model, is reached. This is enforced by a flag element in the configuration of the form $flg(B, R)$, where B is a boolean indicating whether the specified limit is reached, and R is a number representing the current round of rewrites. This flexible mechanism enables us to specify the granularity of a round in terms of the amount of time we wish to run a Monte Carlo simulation.

Other components in the configuration include variables *crate* and *arate* maintaing, respectively, current client request and attacker request rates for the current window (for variable client request and/or attacker rates).

There are several parameters of interest that are supported by the model. These are listed in Figure 1.

Now that the building blocks of the model have been introduced, we describe next the behavior of the model. In order to save space and avoid presentation clutter, we describe the events associated with the rewrite rules specifying the model's dynamic behavior instead of including the rules verbatim from the specification.

Client sending a REQ [CSend]. When it is time for the client to send a new set of REQs to the server, which is signaled by the self-addressed *poll* message, and the client is in the *waiting* state, meaning that the client is yet to get connected and has not yet given up retrying, the [CSend] rule applies. In this rule, two cases are considered. If the client has not yet reached its retrial span limit, it sends K replicated REQ messages to the server, with K the value of its *repcnt* attribute, increments its retries counter *retries*, updates the replication count for the next attempt, and schedules another *poll* message after T_c time units. The scheduler inserts the new messages into its list for proper scheduling. Otherwise if the client has reached its limit, the client changes its status to *failed* and will no longer attempt to connect to the server.

Client receiving an ACK [CRec, CDscrd]. Once a client receives an acknowl-

edgement message from the server while in the *waiting* state, the client consumes the message and changes its status to *connected*. Accepted clients do not send new messages. If an acknowledgement is received while the client has already been accepted or failed, the acknowledgment message is just discarded.

Attacker sending a REQ [ASend]. Once it is time for the attacker to send a new (fake) REQ message to the server, which is signaled by the *poll* message, the attacker emits such a message and uses the current attack rate α to schedule the next REQ message after $1/(\alpha S)$ time units.

Attacker ignoring ACKs [ADscrd]. Obviously, an acknowledgment message from the server in response to an attacker REQ is just noise, and is discarded.

Server handling a REQ [ProcessREQ]. When a server receives a connection request message $REQ(Oid)$ from object Oid, it first checks whether its request buffer stored in the *reqlist* attribute is full or not. If the buffer is not yet full, the request is simply added to the list. Otherwise, if the buffer has already reached its maximum capacity, the server tosses a biased coin with success probability R, given by its *reqcnt* attribute, and uses the outcome of this experiment to decide on whether to replace an existing request selected uniformly at random with the incoming request or to drop the incoming request altogether. The server also increments its client request count *reqcnt* in preparation for the next incoming client request. This rewrite rule is probabilistic and involves sampling from both a Bernoulli distribution and a uniform distribution.

In either case, the server updates its counter *cnt* appropriately for the number of incoming legitimate client requests for analysis purposes.

Server timing out [Timeout]. Once a server times out, indicated by the self-addressed *timeout* message, the server resets its *reqcnt* counter, sends out an acknowledgment message for every connection request stored in its buffer, and then clears the buffer. The server also re-schedules an internal timeout message, and updates the client request and attacker rates (if variable client and attacker rates are used) in preparation for the new time window.

5 Statistical Model Checking Analysis

We have used the ASV model described above to perform statistical quantitative model checking analysis of various QuaTEx formulas using VeStA, which runs Monte Carlo simulations on the model through its interface with Maude to produce a point estimator of the quantity of interest for each of these formulas, given a desired confidence interval for the experiment and its maximum tolerable size. For this purpose, we specify the nature of the quantities to be statistically estimated. The quantities specified as QuaTEx formula declarations are listed below.

Connection ratio. This is the ratio of clients successfully connected to the total number of clients in a configuration:

$$\begin{aligned} connRatio(t) = {} & \textbf{if } time() > t \textbf{ then } countConnected()/countClients() \\ & \textbf{else } \bigcirc\,(connRatio(t)) \end{aligned}$$

with $time()$ a state function that returns the global clock time in the current

configuration. The number of accepted clients *countConnected*() is computed by equationally counting the number of clients whose status field is *connected*, while the total number of clients *countClients*() can be easily extracted from the client counter attribute maintained in the client generator object.

Average TTS. This is the ratio of the total time-to-service added up over all accepted clients to the number of accepted clients:

$$avgTTS(t) = \textbf{if } time() > t \textbf{ then } sumTTS()/countConnected() \textbf{ else } \bigcirc(avgTTS(t))$$

The total TTS $sumTTS()$ is computed by adding up the time intervals given by the arrival time and service time attributes of every accepted client. The number of accepted clients is computed as described above.

Bandwidth usage. This is the amount of bandwidth used by legitimate clients attempting to establish a connection with the server. Bandwidth usage is measured in terms of the number of legitimate requests and is given by the legitimate client request counter attribute of the server object.

$$bw(t) = \textbf{if } time() > t \textbf{ then } bwUsage() \textbf{ else } \bigcirc(bw(t))$$

Connection Confidence. This is the probability that a given client will successfully establish a connection to the server:

$$connConfidence(i,t) = \textbf{if } time() > t \textbf{ then } hasConnected(i) \textbf{ else } \bigcirc(connConfidence(i,t))$$

This is computed using a simple function $hasConnected(i)$ that, given a client id, returns 1.0 if the client has actually connected to the server and 0.0 otherwise.

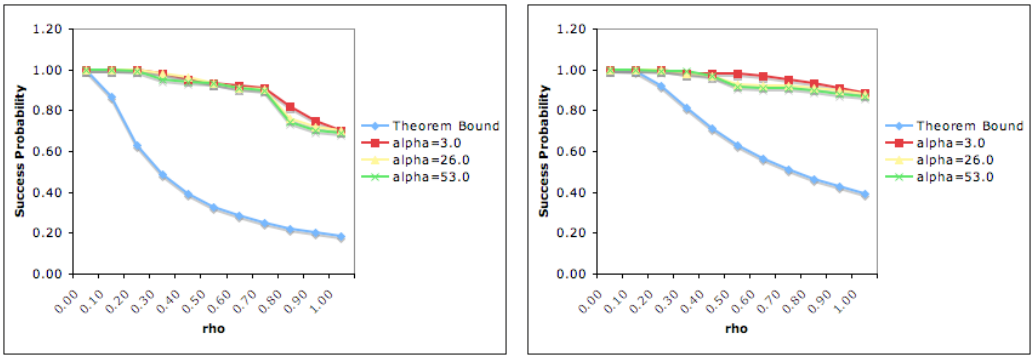
Throughout this section, we assume a 95% confidence interval with size at most 0.05. We also fix S, the mean server processing rate, to 600 packets per time units, the server and client timeouts, T_s and T_c, to 0.4 time units, and the initial generation delay, *initDelay*, to 0.2 time units, unless otherwise specified.

5.1 Verification of ASV Properties

In this section, we use the ASV model to formally verify two important properties of the protocol, which were given in [14]. The properties provide guarantees on the connection confidence and legitimate bandwidth consumption of ASV.

To provide a benchmark for the performance of ASV with respect to these properties, a simpler protocol, namely the omniscient protocol, in which the server and clients are always aware of the current ρ and α is also described in the cited paper. We modeled the omniscient protocol to compare the bounds given by these theorems and provide formal statistical evidence of their correctness. In the omniscient model, the server accepts client requests with probability that depends only on the (now known) ρ and α, which implies a simpler server object specification as it no longer needs to maintain a buffer *reqlist* nor a client request replication count *reqcnt*. Furthermore, a client uses the value $\lceil \alpha/\rho \rceil$ as its replication count, and fails if no ACK is received after T_c units of time.

Connection confidence (Theorems 1 and 2 of [14]). The property gives a lower bound on the probability with which a given client will be able to establish a



(a) The ASV protocol (b) The omniscient protocol

Fig. 2. Connection confidence: Theorem bound vs. estimated values

connection to the server, given a condition on the client request rate ρ. In particular, for the omniscient protocol, Theorem 1 states that if $\rho_{\max}$ is at most $1/(-2\log\delta)$, with δ a given confidence parameter, then any given client will be accepted with probability at least $1-\delta$. A similar lower bound is guaranteed for ASV under a slightly stronger condition on $\rho_{\max}$, which is that $\rho_{\max} \leq 1/(-5\log\delta)$. These properties are verified by fixing a client i (say the first client [3]) and then estimating the expectation of the $connConfidence(i,t)$ formula. Figure 2 plots the estimated probabilities of the first client getting connected versus the bound given by the theorem at different confidence parameter values, giving rise to different upper bounds on $\rho_{\max}$, and assuming three different levels of attack (low, medium, and high). For this analysis, we assume a worst-case analysis with $\rho = \rho_{\max}$. As both Figures 2(a) and 2(b) show, the estimated success probabilities under both protocols are always higher than the respective theorem bounds over the whole range of values of ρ, which confirms the statements of the theorems. We also note that, with respect to the connection confidence property, both protocols are able to maintain high success probabilities at higher attack levels compared to those at low attack levels.

Bandwidth usage (Theorem 4 of [14]). This property gives a bound on the bandwidth consumed by legitimate clients in ASV. In particular, Theorem 4 states that, under the assumption of bounded variability in ρ, the ratio of the legitimate bandwidth consumed in ASV to that in the omniscient protocol is bounded above by $\log(\alpha_{\max})/\log(\frac{1}{\rho_{\max}})$. As discussed in [14], the restriction on the variability of ρ is imposed only to simplify manual analysis and the statement of the theorem. Beside verifying the theorem, we confirm the conjecture that the upper bound holds even when the restrictions on ρ are lifted. This is achieved by estimating the expectation of the formula $bw(t)$, while fixing $\rho_{\min}$ to a very low value (close to 0.0) and allowing $\rho_{\max}$ to vary from very small values all the way up to almost 1.0. Figure 3 plots the estimated ratios at three different attack levels as well as the upper bound given by the theorem at the lowest level of attack ($\alpha = 3.0$). The bounds corresponding to medium and high attack rates are not shown as they are both mostly too high to appear within the figure's scale. The important observation here is that the ratios of the legitimate bandwidth consumed by ASV to that of the omniscient protocol

[3] For this analysis, the choice of the client is immaterial since all clients behave identically and are introduced to statistically similar attack conditions.

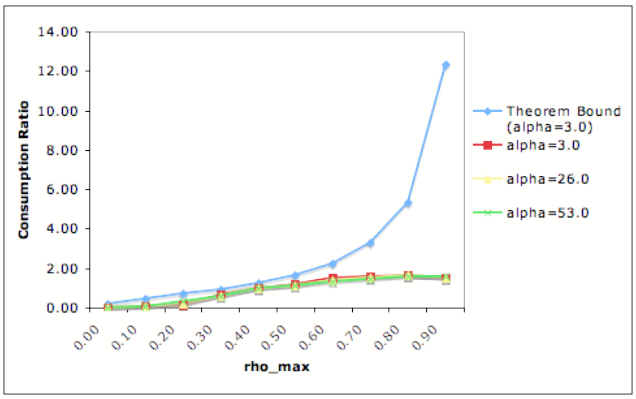


Fig. 3. Bandwidth Usage: Theorem bound vs. estimated values

are always below the theorem's bound. The figure also suggests that these ratios change only slightly across different attack conditions.

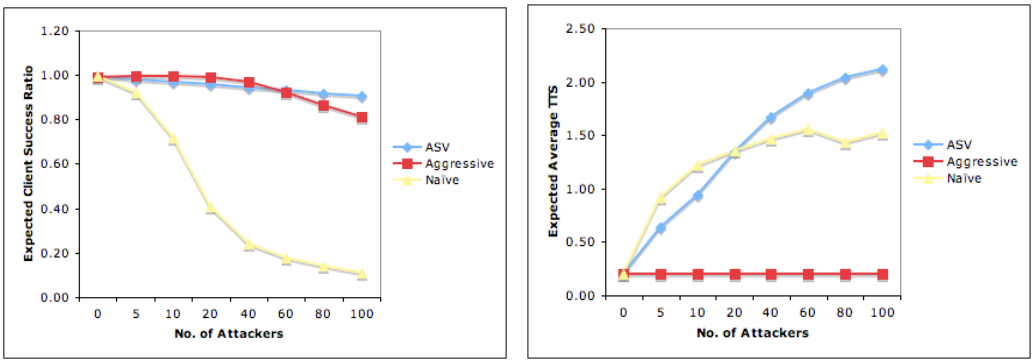
5.2 ASV versus Non-adaptive Selective Verification Schemes

In [14], the results of various NS-2 simulations comparing ASV to two non-adaptive selective verification variations under various attack conditions were reported. The non-adaptive schemes are: (i) the *Naive* protocol, in which a client does not increase its replication count with time, and (ii) the *Aggressive* protocol, in which a client sends the maximum number of requests (2^J) at once upon entering the configuration. The simulations reported validated the different trade-offs associated to the adaptive versus the non-adaptive schemes in terms of the ratio of successful connections, the average time-to-service, and the legitimate bandwidth used. We show here that using the ASV model in Maude along with two variants of it, corresponding to the non-adaptive protocols above, we obtain similar results through statistical quantitive analysis with VeStA, independently confirming the simulation analyses.

Since the non-adaptive protocols differ from the ASV protocol only in client request behavior, the Naive and Aggressive models are very similar to that of ASV. In fact, their models differ from the ASV model in essentially one rewrite rule, which is the one labeled [CSend]. In the Naive protocol model specification, the [CSend] rule maintains the initial replication count, which is equal to 1, causing the client to send exactly one REQ at every client timeout until connected or failed. On the other hand, the [CSend] rule of the Aggressive model distinguishes two cases. During the first attempt at making a request to the server (indicated by the *retires* attribute being 0), the rule replicates the request 2^J times (the replication count is simply ignored). Otherwise, if the client has already sent its initial set of requests ($0 < \mathit{retries} \leq J$), the client remains silent.

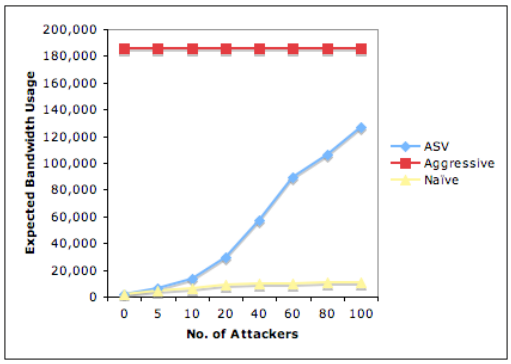
Since we intended to independently confirm the results of the NS-2 simulations of [14], we instantiate the model using essentially the same values for the parameters. That is, we set ρ to 0.08, J to 7, and *Limit* (the simulation duration) to 30.0 time units. The expectation of the connection ratios, average TTS values, and legitimate bandwidth usage are estimated at different attacker rates (given in terms of the number of attackers). The results are shown in Figure 5.2.

As the figure shows, the results obtained confirm the effectiveness and efficiency



(a) Connection success ratio

(b) Average TTS



(c) Bandwidth usage

Fig. 4. Performance of ASV compared to non-adaptive schemes

of ASV compared to non-adaptive selective verification schemes. In Figure 4(a), ASV is statistically shown to be effective even under high rates of attack, where it outperforms both non-adaptive protocols. ASV is able to achieve this high performance at the expense of some latency inherent in its adaptive behavior (See Figure 4(b)), which is higher than that of the Naive protocol during periods of medium to heavy attacks. For legitimate bandwidth consumption, Figure 4(c) shows that at low to medium attacks, ASV is able to maintain low bandwidth consumption levels that are comparable to that of the Naive protocol. Even at higher attacks, ASV manages to outperform the aggressive protocol by a respectable margin.

6 Related Work

The VeStA tool, along with Maude, has been used for statistical model checking in several projects, including analysis of TCP SYN floods-based DoS attacks within the shared channel model [2], analysis and redesign of a wireless sensor networks protocol [13], and a few case studies in object-based stochastic hybrid systems [20].

A similar rewriting-based approach using Maude was also described in [15], in which the statistical model checking algorithm improves on VeStA's by precomputing the required number of sample runs and using a normal distribution to approximate expected values. The approach was then applied to a resource optimization problem in embedded systems.

Other statistical model checking and quantitative analysis tools have been developed. They include PRISM [11], a BDD-based model checker with a simple state-language for system specification, and APMC [9], which is based on a a randomized algorithm for approximating probabilities in Discrete Markov Chains and supports client-server architectures for parallel computations. Other tools include Rapture [12] (which is also a BDD-based model checker) and Erlangen-Twente Markov Chain Checker $\mathsf{E} \vdash \mathsf{MC}^2$ [10]. Compared to the VeStA/Maude approach, most of these tools sacrifice expressiveness and generality for algorithmic decidability.

There have been several works in the literature to formally analyze DoS attacks. These include Meadow's formal framework for evaluating protocols against DoS attacks [18], an information flow-based framework using the Security Protocols Process Algebra (SPPA) for the detection of DoS vulnerabilities [16], and an observation-equivalence approach based on π-calculus for DoS detection [1]. Other formal approaches and extensions and applications of these approaches have also been developed [24,17,7].

7 Concluding Remarks and Future Work

We have described a rewriting-based approach to the formal specification and verification of protocol security properties related to availability such as resistance to DoS attacks. This approach is based on probabilistic rewrite theories, quantitative probabilistic temporal logic QuaTEx formulas, and statistical model checking, and is supported by tools such as Maude and VeStA. We have applied this approach to the formal specification and verification of the ASV DoS-resistant protocol, gaining a higher level of assurance about availability properties that had been previously studied for ASV either by hand-carried analytic proofs or by simulation.

This work should be further developed in several ways. On the one hand, following the ideas in [5], ASV should be formally specified as a generic *protocol wrapper* which provides DoS protection not for a single underlying protocol, but for a wide range of (originally unprotected) protocols. On the other, formal analysis results like those presented here should then be obtained for a variety of underlying protocols of interest. A more challenging but very interesting question is that of obtaining for a generic wrapper version of ASV *generic DoS protection guarantees* that would apply not just to a given underlying protocol, but to entire families of protocols.

Acknowledgements. This work was supported in part by NSF CNS05-5170 CNS05-09268 CNS05-24695, NSF CNS 07-16421, NSF CNS 07-16638, ONR N00014-04-1-0562 N00014-02-1-0715, DHS 2006-CS-001-000001 and a grant from the MacArthur Foundation. The views expressed are those of the authors only.

References

[1] Martín Abadi, Bruno Blanchet, and Cédric Fournet. Just fast keying in the pi calculus. In *Prog. Lang. and Systems, 13th European Symposium on Programming*, volume 2986 of *LNCS*. Springer, 2004.

[2] Gul Agha, Carl Gunter, Michael Greenwald, Sanjeev Khanna, José Meseguer, Koushik Sen, and Prasanna Thati. Formal modeling and analysis of DoS using probabilistic rewrite theories. In *International Workshop on Foundations of Computer Security (FCS'05)*, 2005.

[3] Gul Agha, José Meseguer, and Koushik Sen. PMaude: Rewrite-based specification language for probabilistic object systems. *Electronic Notes in Theoretical Computer Science*, 153(2):213–239, 2006.

[4] Roberto Bruni and José Meseguer. Semantic foundations for generalized rewrite theories. *Theor. Comput. Sci.*, 360(1-3):386–414, 2006.

[5] Rohit Chadha, Carl A. Gunter, Jose Meseguer, Ravinder Shankesi, and Mahesh Viswanathan. Modular preservation of safety properties by cookie-based DoS-protection wrappers. In *IFIP Formal Methods for Open Object-based Distributed Systems (FMOODS '08)*, Oslo, Norway, June 2008.

[6] Manuel Clavel, Francisco Durán, Steven Eker, Patrick Lincoln, Narciso Martí-Oliet, José Meseguer, and Carolyn Talcott. *All About Maude - A High-Performance Logical Framework: How to Specify, Program, and Verify Systems in Rewriting Logic (LNCS)*. Springer-Verlag, Secaucus, NJ, USA, 2007.

[7] Alwyn E. Goodloe. *A Foundation for Tunnel-Complex Protocols*. PhD thesis, University of Pennsylvania, 2008.

[8] Carl A. Gunter, Sanjeev Khanna, Kaijun Tan, and Santosh S. Venkatesh. DoS protection for reliably authenticated broadcast. In *Proceedings of the Network and Distributed System Security Symposium*. The Internet Society, 2004.

[9] Thomas Hérault, Richard Lassaigne, Frédéric Magniette, and Sylvain Peyronnet. Approximate probabilistic model checking. *Verification, Model Checking, and Abstract Interp.*, pages 307–329, 2004.

[10] Holger Hermanns, Joost-Pieter Katoen, Joachim Meyer-Kayser, and Markus Siegle. A markov chain model checker. In *TACAS'00: The 6th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, pages 347–362, London, UK, 2000. Springer-Verlag.

[11] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In H. Hermanns and J. Palsberg, editors, *Proc. 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.

[12] Bertrand Jeannet, Pedro R. D'Argenio, and Kim G. Larsen. RAPTURE: A tool for verifying markov decision processes. In *Tools Day, International Conference on Concurrency Theory, CONCUR'02*, Czech Republic, August 2002. Technical Report, Faculty of Informatics at Masaryk University Brno.

[13] Michael Katelman, Jose Meseguer, and Jennifer Hou. Redesign of the LMST wireless sensor protocol through formal modeling and statistical model checking. In *International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS)*, 2008. To appear.

[14] Sanjeev Khanna, Santosh S. Venkatesh, Omid Fatemieh, Fariba Khan, and Carl A. Gunter. Adaptive selective verification. In *IEEE Conference on Computer Communications (INFOCOM '08)*, Phoenix, AZ, April 2008. IEEE.

[15] Minyoung Kim, Mark-Oliver Stehr, Carolyn L. Talcott, Nikil D. Dutt, and Nalini Venkatasubramanian. A probabilistic formal analysis approach to cross layer optimization in distributed embedded systems. In *Formal Methods for Open Object-Based Distributed Systems (FMOODS '07)*, volume 4468 of *LNCS*. Springer, 2007.

[16] Stéphane Lafrance and John Mullins. An information flow method to detect denial of service vulnerabilities. *J. UCS*, 9(11):1350–, 2003.

[17] Ajay Mahimkar and Vitaly Shmatikov. Game-based analysis of denial-of-service prevention protocols. In *IEEE Computer Security Foundations Workshop, (CSFW-18 2005)*. IEEE Computer Society, 2005.

[18] Catherine Meadows. A formal framework and evaluation method for network denial of service. In *CSFW*, pages 4–13, 1999.

[19] José Meseguer. Conditional rewriting logic as a unified model of concurrency. *Theor. Comput. Sci.*, 96(1):73–155, 1992.

[20] José Meseguer and Raman Sharykin. Specification and analysis of distributed object-based stochastic hybrid systems. In *Hybrid Systems: Computation and Control (HSCC 2006)*, volume 3927 of *LNCS*. Springer, 2006.

[21] Koushik Sen, Nirman Kumar, Jose 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.

[22] Koushik Sen, Mahesh Viswanathan, and Gul Agha. On statistical model checking of stochastic systems. In *Computer Aided Verification (CAV 2005)*, volume 3576 of *LNCS*. Springer, 2005.

[23] Koushik Sen, Mahesh Viswanathan, and Gul A. Agha. VESTA: A statistical model-checker and analyzer for probabilistic systems. In *Second International Conference on the Quantitative Evaluaiton of Systems (QEST)*, pages 251–252, 2005.

[24] Che-Fn Yu and V. D. Gligor. A specification and verification method for preventing denial of service. *IEEE Trans. Softw. Eng.*, 16(6):581–592, 1990.