Performance Modeling of Moving Target Defenses with Reconfiguration Limits

Warren Connell, Daniel A. Menascé, *Fellow, IEEE,* and Massimiliano Albanese, *Member, IEEE*

Abstract—Moving Target Defense (MTD) has recently emerged as a game changer in the security landscape due to its proven potential to introduce asymmetric uncertainty that gives the defender a tactical advantage over the attacker. Many different MTD techniques have been developed, but, despite the huge progress made in this area, critical gaps still exist with respect to the problem of studying and quantifying the cost and benefits of deploying MTDs. In fact, all existing techniques address a very narrow set of attack vectors, and, due to the lack of shared metrics, it is difficult to quantify and compare multiple techniques. Building on our preliminary work in this field, we propose a quantitative analytic model for assessing the resource availability and performance of MTDs, and a method for maximizing a utility function that captures the tradeoffs between security and performance. The proposed model generalizes our previous model and can be applied to a wider range of MTDs and operational scenarios to improve availability and performance by imposing limits on the maximum number of resources that can be in the process of being reconfigured. The analytic results are validated by simulation and experimentation, confirming the accuracy of our model.

1 Introduction

In recent years, Moving Target Defense (MTD) has emerged as a game changer in cyber security because it has the potential of turning the typical asymmetry of the security landscape in favor of the defender [1], [2]. As we face more sophisticated and persistent attackers, Moving Target Defense becomes critical for enhancing the dependability of today's complex systems. Avižienis *et al.* [3] defined several dependability attributes. Security is sometimes treated as an attribute of dependability, but a common trend is that of referring to the composite concept of *dependability and security* [3], [4].

As we increasingly depend on IT systems, securing them and ensuring their dependability is of utmost importance. However, traditional approaches to cyber defense are governed by slow and deliberative processes. Adversaries can benefit from this situation and can systematically probe target networks with the confidence that they will change slowly, making it possible to eventually acquire sufficient knowledge to engineer reliable exploits against their targets. To address this problem, researchers and practitioners have developed a myriad of MTD techniques that aim at presenting adversaries with changing attack surfaces and system configurations, forcing them to continually re-assess and re-plan their cyber operations. This approach enables novel defenses that can adapt to evolving IT landscapes, sophisticated and persistent attackers, and changing attack vectors. One important limitation of current MTD techniques is that most of them are designed to protect systems against a very narrow set of attack vectors. For instance, MTD techniques have been developed to protect against DoS attacks [5], [6], data exfiltration [7], and SQL injection [8]. A direct consequence of the over-specialization of MTD techniques is the lack of shared metrics to assess their effectiveness. In fact, most of the proposed techniques tend to measure their effectiveness in different and often incompatible ways. Thus, despite the huge amount of work done in this area, gaps still exist with respect to the analysis and quantification of MTDs.

When deploying MTDs, as for any other security mechanism, there exists a trade-off between performance and security [9]. MTDs operate by periodically reconfiguring one or more system parameters. As the reconfiguration frequency increases, an MTD technique can provide better security, but inevitably increases the overhead on the system and reduces the availability of resources, thus affecting the overall performance. In particular, the problem of evaluating the impact of MTDs on the availability of resources and on system performance has not been formally studied. This is an important problem because, regardless of the MTD technique used, resources being reconfigured are temporarily unavailable to legitimate users.

In our previous work [10], we proposed a quantitative analytic model for assessing the resource availability and performance of MTDs, and a simple method for determining the highest possible reconfiguration rate, and thus the smallest probability of attacker's success, that meets performance and stability constraints. Although that preliminary framework represents a first important step towards a comprehensive solution to the MTD quantification problem, several limitations still exist. As shown in [10], the availability of computational resources may drop significantly if too many resources are being reconfigured at the same time, causing a backlog of service requests and response time peaks. As a simple method to prevent a similar scenario, we defined a stability metric and devised a method to determine the maximum reconfiguration rate that meets stability constraints. In this paper, in order to ensure a baseline for resource availability, we introduce limits on the number c^* of resources that can be reconfigured at the same time. To achieve this objective, we define two policies – namely the *drop* policy and the *wait* policy – to handle reconfiguration requests that are received when there are already c^* resources in the reconfiguration phase. We significantly revised our previous analytic model to capture the effects of

W. Connell, D.A. Menascé, and M. Albanese are with the Volgenau School of Engineering, George Mason University, Fairfax, VA, 22030. *The work of W. Connell and M. Albanese was partially supported by Army Research Office grants W911NF-13-1-0421 and W911NF-13-1-0317, and by the Office of Naval Research grant N00014-13-1-0703. The work of D.A. Menascé was partially supported by the AFOSR grant FA9550-16-1-0030.*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

these policies on the availability and performance of MTDs. In summary, this paper provides the following major contributions: (i) A quantitative analytic model for assessing the availability and performance of resources that are reconfigured by an MTD as well as an attacker's success probability, in the presence of limits on the number of resources that can be reconfigured at the same time; (ii) A method for determining the reconfiguration rate that maximizes a utility function that incorporates the tradeoffs between the attacker's success probability and response time; (iii) A validation of the analytic model through simulation and experimentation.

The rest of this paper is organized as follows. Section 2 provides background information on MTDs and discusses our attack model. Sections 3-7 present our quantitative analytic model, whereas Section 8 describes the experimental testbed used for evaluating the model and for analyzing some transient behaviors of MTDs. Section 9 presents numerical results obtained with the help of the analytic model and through experimentation. Section 10 discusses related work and Section 11 provides some concluding remarks and discusses ongoing and future research directions.

2 MTDs and Attack Model

Cyber attacks are typically preceded by a reconnaissance phase in which adversaries collect information about the target system, including network topology, service dependencies, and unpatched vulnerabilities. Because most system configurations are static – hosts, networks, software, and services do not reconfigure, adapt, or regenerate except in deterministic ways to support maintenance and uptime requirements – it is only a matter of time for attackers to acquire accurate knowledge about their targets and plan their attacks. In order to address this important problem, significant work has been done in the area of Adaptive Cyber Defense (ACD), which includes concepts such as Moving Target Defense (MTD), as well as artificial diversity and bio-inspired defenses. MTD techniques are mechanisms for continuously changing or shifting a system's attack surface, thus increasing complexity and cost for the attackers [1]. A system's attack surface has been defined as the *"subset of the system's resources (methods, channels, and data) that can be potentially used by an attacker to launch an attack"* [11]. Thus, the majority of MTD techniques operate by periodically reconfiguring one or more system parameters in order to disrupt the knowledge an attacker may have acquired about those parameters and, consequently, render the attack's preconditions impossible or unstable. Intuitively, dynamically reconfiguring a system is expected to introduce uncertainty for the attacker and increase the cost of the reconnaissance effort.

Different MTDs may be designed to address different stages of the Cyber Kill Chain, a framework developed by Lockheed Martin as part of the Intelligence Driven Defense model for identification and prevention of cyber intrusions activity. The model identifies what steps the adversaries must complete in order to achieve their objective: reconnaissance, weaponization, delivery, exploitation, installation, command & control, actions on objectives. The majority of the techniques currently available are designed to address the reconnaissance phase of the cyber kill chain, as they attempt to interfere with the attacker's effort to gather information about the target system.

A major drawback of many MTDs is that they force the defender to periodically reconfigure the system, which may introduce a costly overhead to legitimate users, as well as the potential for denial of service conditions. Additionally, most existing techniques are purely proactive in nature or do not adequately consider the attacker's behavior. To address this limitation, alternative approaches aim at inducing a "virtual" or "perceived" attack surface by deceiving the attacker into making incorrect inferences about the system's configuration [12], rather than actually reconfiguring the system. Honeypots have also been traditionally used to try to divert attackers away from critical resources [13], but they have proven to be less effective than MTDs because they provide a static solution: once a honeypot or honeynet has been discovered, the attacker will simply avoid it.

As indicated in [14], one of the key MTD problems is the *timing problem*, i.e., "when to adapt." The longer it takes for a system to adapt (or reconfigure), the more time is available for an attacker to gather information about the system and therefore the higher the probability that an attacker will succeed. The *attack model* considered here assumes that an attacker will eventually be able to penetrate a system given sufficient time to obtain the information necessary to perpetrate the attack. Additionally, we also consider that the probability that an attacker succeeds increases monotonically with time. The probability $P_s(t)$ that an attacker will succeed in t time units in attacking a resource is important in determining the required *reconfiguration rate*. Fig. 1 shows two examples of the $P_s(t)$ function: a linear and an exponential one. The linear function has the form $P_s(t) = t/T_s$ and indicates that the probability of success of the attack increases linearly with time and reaches 1 (i.e., success) at time T_s. The exponential function indicates a situation in which the attacker initially accumulates knowledge at a low rate, becomes exponentially more knowledgeable over time, and succeeds at time T_s. Both curves in Fig. 1 assume $T_s = 10$. The expression for the exponential version of $P_s(t)$ is

$$P_s(t) = 1 - \frac{1 - e^{(t-T_s)}}{1 - e^{-T_s}}. \quad (1)$$

As an example, consider an IP sweep combined with a port scan, where the attacker's goal is to discover the IP address of the machine running a specific service within the target network. The attacker sequentially scans all IP addresses in a given range. Assuming an IP space of n addresses and that t^* time units are required to scan a single IP, we obtain $T_s = n \cdot t^*$ and $P_s(t) = t/T_s = t/(n \cdot t^*)$.

As another example, consider the following DoS attack. The attacker initially compromises n hosts, which takes t^* time units. Then, each of the newly compromised hosts compromises additional n hosts, which takes additional t^* time units. At any given

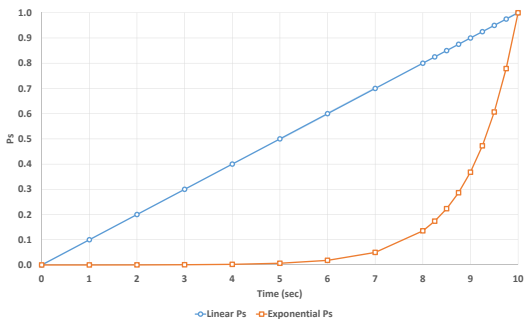


Fig. 1. Probability of success P_s vs. time for $T_s = 10$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

time t, the total number of compromised hosts, including the attacker's machine, is $N(t) = 1 + n + n^2 + \ldots + n^k = \frac{1-n^{k+1}}{1-n}$, where $k = \lfloor t/t^* \rfloor$. We can assume that the attacker's success probability is proportional to the aggregate amount of flood traffic that compromised hosts can send to the victim, compared to the victim's capacity to handle incoming traffic. Let V denote the volume of traffic the victim can handle per time unit and let v denote the amount of traffic each compromised node can send per time unit. Then,

$$P_s(t) = \min\left\{1, \frac{N(t)\cdot v}{V}\right\} = \min\left\{1, \frac{v}{V}\cdot\frac{1-n^{\lfloor t/t^* \rfloor+1}}{1-n}\right\}$$

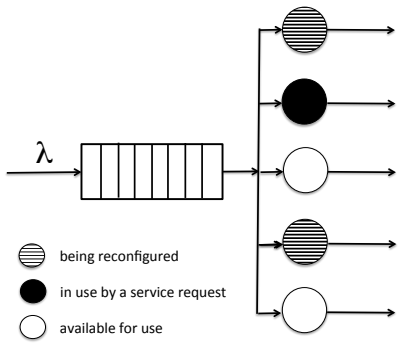


Fig. 2. Queuing representation of the reference scenario

3 Quantitative Analysis of MTDs

The computing environment we consider in this paper consists of c similar *resources* (e.g., VMs) available to serve incoming service requests that arrive with an average arrival rate λ, join a single queue and are served by any of the available resources, with an average service time T (i.e, an average service rate equal to $1/T$). All service requests have the same priority and are served in FCFS order. An MTD technique consists in each resource occasionally, at random intervals, reconfiguring itself independently of the other resources. Thus, each resource handles *service requests* as well as *reconfiguration requests*. While a resource is being reconfigured, it is not available to handle service requests (see Fig. 2). Without reconfigurations, the system behaves exactly like an M/M/c queue [15]. However, because resources become unavailable while they are being reconfigured, the number of available resources varies over time and the M/M/c results do not apply here.

Now, assume that each resource is reconfigured at an average rate of α. The quantitative models presented here do not depend on the specific nature of the *reconfiguration technique* applied to the system resources. Some examples of MTD-based reconfiguration techniques include: swapping out a VM with a clean instance [16] that has a new IP address, Address Space Layout Randomization (ASLR) [17], service diversification, IP address rotation, and TCP or SSL connection rotation. These reconfigurations make it more difficult for an attacker to learn about the resources, and disrupt attacker's persistence in the system. The attacker's success probability is a function of the average reconfiguration rate α, which also affects the average number of resources available to serve requests and the queueing time for service requests.

While these *qualitative* tradeoffs are intuitive and not surprising, there is a need for *quantitative* models that allow us to determine the impact of the reconfiguration rate on resource availability, response time of service requests, and attacker's success probability. We use Continuous Time Markov Chains (CTMC) (see e.g. [15]) to compute the probability distribution of the number of resources being reconfigured as a function of α and other parameters and then use that distribution to determine resource availability and response time, among other metrics. Markov chains have been used for many decades to study various aspects of computer and communication systems. The novelty in each case is on how the state of a CTMC should be defined to represent the system to be analyzed.

To ensure that there is always a minimum number of resources available to handle service requests, we consider policies that limit the maximum number c^* of resources being reconfigured. Note that if $c^* = c$ (i.e., the unlimited case), the results obtained here revert to our previous simpler model [10]. If c^* resources are being reconfigured, additional reconfiguration requests may either be dropped (*drop* policy) or queued (*wait* policy). We analyze this generic MTD in three steps: (i) analysis of the effect of the reconfiguration rate α on the probability distribution of available resources; (ii) analysis of the effect of that availability on response time; and (iii) calculation of the effective reconfiguration rate and determination of the attacker's probability of success.

We summarize our assumptions below .

- A1: All c resources have the same average processing rate $1/T$ and serve requests from a single queue in FCFS order.
- A2: The average processing rate $1/T$ of the resources does not depend on the number of service requests in the system. Note that this assumption, called *homogeneous service times*, is more general than the assumption that service times are exponentially distributed [18].
- A3: The average rate λ at which service requests arrive does not depend on the number of requests in the system. This assumption, called *homogeneous arrivals*, is more general than the Poisson arrivals assumption. [18].
- A4: All service requests have the same priority.
- A5: All resources reconfigure independently of each other and do so at the same average rate α. Resources are not reconfigured while processing a service request. We consider two policies to handle this situation: drop or queue the reconfiguration request.
- A6: The average rates λ, $1/T$, and α are assumed to be stationary (i.e., not changing over time), as they respectively depend on (1) the nature of the service request arrival process, (2) the capacity of resources and the characteristics of incoming service requests, and on (3) system owner's setting for the reconfiguration rate. While these average rates are assumed to be stationary, the respective processes are not deterministic. Clearly, changes in any of these average rates can be taken into account by recomputing the models described here.
- A7: The time S to reconfigure a resource is assumed to be exponentially distributed in our models, even though in our experiments this time was normally distributed and our models showed to be quite robust with respect to the reconfiguration time distribution (see Section 8).

4 Analytic Model Overview

Our analytic model is derived from the queuing representation in Fig. 2. It can be further divided, as shown in Fig. 3, into two parts: a *reconfiguration model* $\mathcal{R}$ for reconfiguration requests and

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

a *performance model* $\mathcal{S}$ for service requests. The reconfiguration model takes as inputs the target rate α at which resources are reconfigured, the average reconfiguration time S, the number of resources c, and the maximum number of resources c^* that can be reconfigured at the same time. This model produces as outputs the availability of resources, the average number of resources available, and the probability distribution $\{p_k^r\}$ of the number of resources being reconfigured. See Table 1 for the names and descriptions of all variables. The $\{p_k^r\}$ distribution, along with the number of resources, the maximum number of resources that can be reconfigured at the same time, the average arrival rate λ of service requests, and the average service time T of requests are inputs to the performance model, which produces the distribution $\{P_k\}$ of service requests in the system and the average response time of requests. The performance model only addresses server-side response time; networking issues and/or delays between clients and the servers depicted in Fig. 2 are outside the scope of this paper and have been addressed elsewhere. The reconfiguration and performance models are solved using CTMCs as explained in the next two sections and are combined iteratively as explained in Section 7.

TABLE 1
Summary of variable names and their description

Variable	Description
$P_s(t)$	Probability that an attacker will succeed in t time units
T_s	Time needed for an attacker to succeed. $P_s(T_s) = 1$
c	Number of resources
c^*	Maximum number of resources than can be in the process of being reconfigured
c_a	Minimum number of resources that are available for use. $c_a = c - c^*$
$\bar{c}$	Average number of resources not being reconfigured.
$\bar{n}_r$	Average number of resources being reconfigured
$\bar{n}_{qr}$	Average number of waiting reconfiguration requests. This number is zero for the dropped policy
α	Target reconfiguration rate (measured in rec/sec)
α'	Effective reconfiguration rate (measured in rec/sec)
S	Average time to reconfigure a resource
p_k	Probability that there are k reconfiguration requests in the system
p_k^r	Probability that k resources are being reconfigured
P_k	Probability that there are k service requests in the system (being served or waiting to be served)
λ	Average arrival rate of service requests
pd	Probability that a reconfiguration request is dropped
age_d	Average age of a resource under the drop policy
age_w	Average age of a resource under the wait policy
d	Average reconfiguration delay, i.e., average time between the start of a reconfiguration and its scheduled start
T	Average time a service request spends using a resource
R	Average server-side response time of service requests

5 Reconfiguration Models

This section presents analytic models for the *drop* and *wait* reconfiguration policies.

5.1 Core Results

This subsection describes core results that apply to both reconfiguration policies. In later subsections we provide specific results for each policy. Each resource cycles through periods in which it is available for use or is being reconfigured. We use $k, (k = 0, \ldots, c)$ to denote the number of resources being reconfigured. Several useful results can be obtained from the probabilities p_k^r that k resources are being reconfigured. These probabilities are a function of the reconfiguration rate α, the average time S to reconfigure a resource, the number of resources c, and the maximum number c^* of resources that can be reconfigured at the same time. Note that c^* is a parameter set by the system admins to control the tradeoff between performance and availability as we discuss later. Thus,

$$p_k^r = f(k, \alpha, S, c, c^*). \tag{2}$$

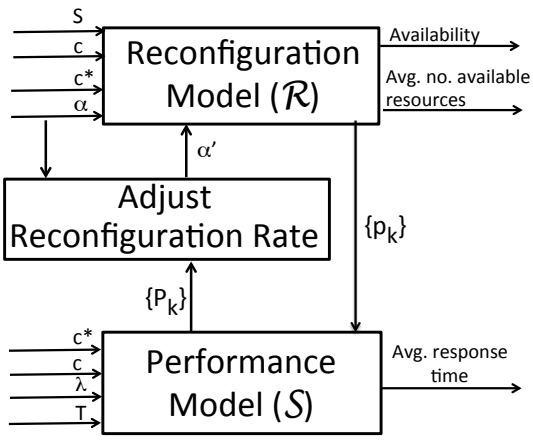


Fig. 3. Analytic model framework

To derive the core results, we assume we know the values of p_k^r, and then show in subsequent sections how these probabilities can be obtained for each reconfiguration policy. Let $\bar{c}$ be the average number of resources available for use (i.e., not being reconfigured) and $\bar{n}_r$ the average number of resources being reconfigured. Thus,

$$c = \bar{c} + \bar{n}_r. \tag{3}$$

But, $\bar{n}_r$ can be obtained from the probabilities p_k^r as

$$\bar{n}_r = \sum_{k=1}^{c^*} k \cdot p_k^r. \tag{4}$$

The *availability* A of the set of resources is given by the fraction of resources available for use, i.e.,

$$A = \frac{\bar{c}}{c} = 1 - \frac{\sum_{k=1}^{c^*} k \cdot p_k^r}{c}. \tag{5}$$

While α is the target reconfiguration rate, it cannot always be achieved because the start of a reconfiguration may be delayed for some time d due to a reconfiguration request being dropped or queued. This is illustrated in Fig. 4, which shows the effective time between reconfigurations. This time is also the average age of each of the resources and defines the effectiveness of the MTD.

$$1/\alpha' = 1/\alpha + d. \tag{6}$$

Therefore, the effective reconfiguration rate is $\alpha' = \alpha/(1+\alpha \cdot d)$. It turns out that the value of the delay d depends primarily on the results of the reconfiguration model $\mathcal{R}$, but is also influenced by whether the resource is idle or not when a reconfiguration is scheduled to start, which depends on the results of the performance model $\mathcal{S}$. The cyclic dependency between the two models is addressed in detail in Section 7. The next section derives the equations for the reconfiguration model assuming that the effective reconfiguration rate is equal to the target rate (i.e., $d = 0$). Then,

in Section 6, we derive the performance model for service requests as a function of the probability distribution of the number of resources being reconfigured.

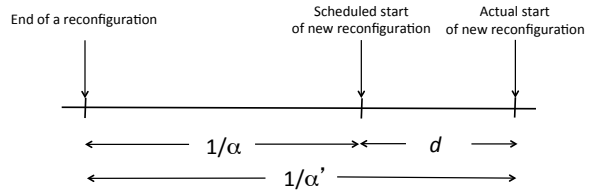


Fig. 4. Target and effective reconfiguration rate

5.2 Drop Reconfiguration Requests Policy

Figure 5 illustrates the flowchart of the reconfiguration process for the drop policy. If the number of resources k being reconfigured is equal to c^*, a request to reconfigure is dropped and a new reconfiguration request is generated after $1/\alpha$ time units on average. If the threshold c^* has not been reached and the resource to be reconfigured is idle (i.e., not handling a service request), k is incremented by 1, the resource is reconfigured, k is decremented by 1, and a new reconfiguration request is generated after $1/\alpha$ time units on average. If the resource to be reconfigured is not idle, the reconfiguration has to wait for the resource to become available.

Because our analysis throughout the paper uses Continuous Time Markov Chains (CTMC), we offer here a very brief definition of these stochastic processes. A CTMC is a stochastic process that has a discrete set of (finite or countable) states and a set of possible transitions between these states. These transitions can happen at any time (thus the continuous time) and are typically associated with events in the system being modeled (e.g., arrival of a service request, completion of a service request). Each transition between two states is associated with a transition rate that measures the rate at which the process moves from one state to another. These transition rates are measured in transitions/time unit (e.g., arrivals/sec). In particular, we will be using here a special case of CTMCs, called Birth-Death processes, in which the states are ordered and transitions are only allowed between neighboring states. The reader is referred to [15] for a comprehensive analysis of CTMCs.

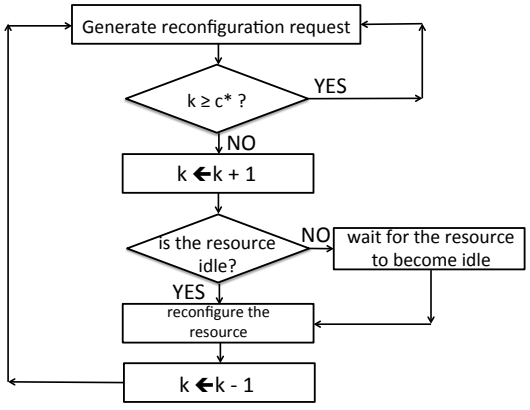


Fig. 5. Flowchart of the reconfiguration cycle under the drop policy

We now use the CTMC of Fig. 6 to compute $p_k^r, (k = 0, \ldots, c^*)$, the probability that k resources are being reconfigured. The state k in the CTMC of Fig. 6 represents the number of reconfiguration requests in the system, which in the case of the drop policy is also the number of resources being reconfigured, so $p_k = p_k^r,\ k = 0, \ldots, c^*$.

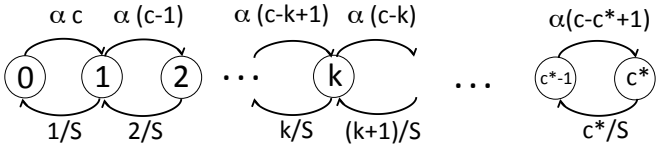


Fig. 6. State transition diagram of the Markov Chain for the reconfiguration model under the drop policy

An expression for $p_k\ (k = 0, \ldots, c^*)$ is obtained by using the general birth-death equation for CTMCs [15]:

$$p_k = p_0 \cdot \prod_{i=0}^{k-1} \frac{\gamma_i}{\mu_{i+1}} \qquad k = 1, \ldots, c^* \tag{7}$$

$$p_0 = \left[1 + \sum_{k=1}^{c^*} \Pi_{i=0}^{k-1} \frac{\gamma_i}{\mu_{i+1}}\right]^{-1} \tag{8}$$

where $\gamma_k = \alpha \cdot (c-k)$, for $k = 0, \ldots, c^* - 1$, is the aggregate rate at which resources are reconfigured when there are k resources being reconfigured and $\mu_k = k/S$, for $k = 1, \ldots, c^*$, is the aggregate rate at which resources complete reconfiguration when there are k resources being reconfigured. Using the expressions for γ_k and μ_k in Eqs. 7 and 8, we obtain

$$p_k = p_0 \cdot \prod_{i=0}^{k-1} \frac{\alpha(c-i)}{(i+1)/S} = p_0 (\alpha \cdot S)^k \binom{c}{k} \quad k = 1, \ldots, c^* \tag{9}$$

An expression for p_0 is obtained by noting that the sum of all probabilities is equal to 1. Thus,

$$p_0 = \left[1 + \sum_{k=1}^{c^*} (\alpha \cdot S)^k \binom{c}{k}\right]^{-1}. \tag{10}$$

The values of p_k can be easily computed because the summation needed to compute p_0 is finite.

In the drop policy, a reconfiguration request is dropped if it arrives when the number of resources being reconfigured is equal to c^*. Thus, the *drop probability*, pd, can be computed as the ratio of the rate of reconfiguration requests that arrive at state $k = c^*$ multiplied by the probability of being at that state, to the sum of the aggregate rates $\gamma_k = \alpha(c-k)$ of reconfiguration requests across all states $k = 0, \ldots, c^*$. Thus,

$$pd = \frac{p_{c^*}\ \alpha\ (c - c^*)}{\sum_{k=0}^{c^*} p_k\ \alpha\ (c-k)} = \frac{p_{c^*}\ (c - c^*)}{\sum_{k=0}^{c^*} p_k\ (c-k)}. \tag{11}$$

We can now compute the average *age* of a resource, i.e., the average time it takes for a resource to be reconfigured after its last reconfiguration. The probability that a reconfiguration request is dropped exactly j times is $pd^j \cdot (1 - pd)$. If a reconfiguration request is dropped exactly j times, the average age of the resource will be $(j+1) \cdot 1/\alpha$ because $1/\alpha$ is the average time between successive reconfiguration requests. Thus, the average age of a resource under the drop policy is

$$\text{age}_d = \sum_{j=0}^{\infty} \frac{j+1}{\alpha} \cdot pd^j \cdot (1 - pd) = \frac{1}{\alpha \cdot (1 - pd)}. \tag{12}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

5.3 Wait Reconfiguration Requests Policy

The flowchart of the reconfiguration cycle under the wait policy is depicted in Fig. 7. This flowchart is very similar to that of Fig. 5, with the difference that when the threshold c^* has been reached, the reconfiguration request is not dropped. Instead, it waits until the the number k of resources being reconfigured drops below c^*.

To analyze the wait policy, we consider the CTMC of Fig. 8 in which the state k $(k = 0, \ldots, c)$ represents the number of reconfiguration requests in the system, either being processed or waiting to be processed. As before, p_k is the probability that there are k reconfiguration requests in the system.

An expression for p_k $(k = 0, \ldots, c)$ is obtained by using the general birth-death equation for Markov Chains given by Eqs 7 and 8, where $\gamma_k = \alpha(c-k)$, for $k = 0, \ldots, c-1$, is the aggregate rate at which reconfiguration requests are generated when there are k reconfiguration requests in the system and the aggregate reconfiguration completion rate μ_k for $k = 1, \ldots, c$ is given by

$$\mu_k = \begin{cases} k/S & k = 1, \ldots, c^* \\ c^*/S & k = c^*+1, \ldots, c \end{cases} \tag{13}$$

Using the expressions for γ_k and μ_k in Eqs. 7 and 8 we obtain

$$p_k = p_0 \cdot \prod_{i=0}^{k-1} \frac{\alpha(c-i)}{(i+1)/S} = p_0(\alpha \cdot S)^k \binom{c}{k} \quad k = 1, \ldots, c^* \tag{14}$$

and

$$\begin{aligned} p_k &= p_0 \cdot \prod_{i=0}^{c^*-1} \frac{\alpha(c-i)}{(i+1)/S} \prod_{i=c^*}^{k-1} \frac{\alpha(c-i)}{c^*/S} \\ &= p_0(\alpha \cdot S)^k \frac{c!}{c^*! c^{*k-c^*}(c-k)!} \quad k = c^*+1, \ldots, c \end{aligned} \tag{15}$$

An expression for p_0 is obtained by noting that the sum of all probabilities is equal to 1. Thus,

$$p_0 = (1 + S_1 + S_2)^{-1} \tag{16}$$

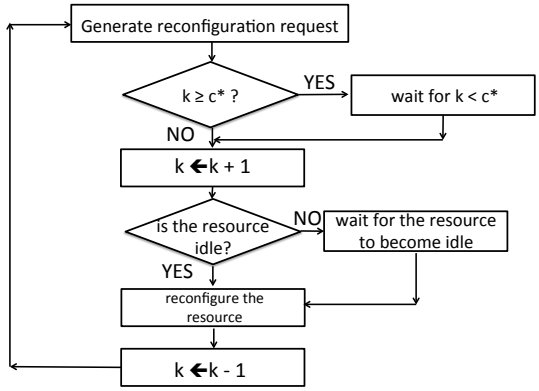


Fig. 7. Flowchart of the reconfiguration cycle under the wait policy

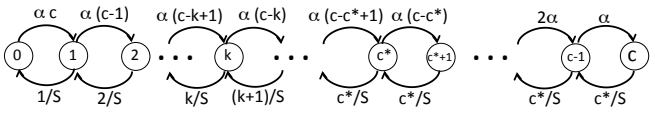


Fig. 8. State transition diagram of the Markov Chain for the reconfiguration model under the wait policy

where

$$S_1 = \sum_{k=1}^{c^*} (\alpha \cdot S)^k \binom{c}{k} \tag{17}$$

and

$$S_2 = \frac{c!}{c^*!} \sum_{k=c^*+1}^{c} (\alpha \cdot S)^k \frac{1}{c^{*k-c^*}(c-k)!}. \tag{18}$$

The values of p_k can be easily computed because the summations needed to compute p_0 are finite. The values of p_k^r, the probability that k resources are being reconfigured, can be computed as a function of p_k as $p_k^r = p_k$ for $k = 0, \ldots, c^*-1$ and $p_{c^*}^r = \sum_{k=c^*}^{c} p_k$. In fact, when the number of reconfiguration requests in the system is smaller than c^*, all reconfiguration requests cause a resource to be reconfigured. When a reconfiguration request finds the number of resources being reconfigured equal to the threshold, and this happens with probability $\sum_{k=c^*}^{c} p_k$, the request has to wait.

One can compute the throughput X_r of reconfiguration requests as a function of p_k^r as

$$X_r = \frac{1}{S} \sum_{k=1}^{c^*} k \cdot p_k^r \tag{19}$$

and the average number N_r of reconfiguration requests in the system as

$$N_r = \sum_{k=1}^{c} k \cdot p_k. \tag{20}$$

Using Little's law, we can the determine the average time in the system for reconfiguration requests as $R_r = N_r/X_r$. This corresponds to the sum of the average reconfiguration time S and the average reconfiguration delay d. Thus,

$$d = \frac{N_r}{X_r} - S. \tag{21}$$

We can now determine the average age of each resource under the wait policy as follows. After a reconfiguration request completes, it takes $1/\alpha$ time units on average for the next reconfiguration request to arrive. But, the next request may have to wait. The arrival of a reconfiguration request can occur anytime within the reconfiguration delay d. On average, that arrival will have to wait $d/2$ time units. Thus, the average age of a resource age_w is given by

$$\text{age}_w = 1/\alpha + d/2. \tag{22}$$

6 Response Time Model

For the performance model, we use the CTMC of Fig. 9 with an infinite number of states where a state $k = 0, 1, 2, \ldots$ represents the number of service requests in the system, either using one of the available resources or waiting for one. Service requests are assumed to come from a Poisson process at an average rate λ and complete at a rate $\mu\delta_k$, where $\mu = 1/T$ (the request completion rate at a resource) and δ_k is derived from the probability distribution obtained from the reconfiguration model. Note that the queue of Fig. 2 is similar to an M/M/c queuing system with an important difference. In an M/M/c model, the rate at which transactions complete is $k\mu$ for $k = 1, \ldots, c$ and $c\mu$ for $k > c$. In our case we need to adapt the transaction completion rate to take into account the resources that may be in the process of being reconfigured. Thus, we follow an approach similar to the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

derivation of the M/M/c queue results [15], with a modification in the average transaction completion rate.

Consider the following additional notation: (i) P_k, the probability that there are k requests in the system, either being processed or waiting for an available resource; (ii) $\mu = 1/T$, the average service rate of each resource; and (iii) $\rho = \lambda/(\mu\ \bar{c})$, the average utilization of the resources. We now provide and explain an expression for δ_k, the multiplier of the resource service rate μ in the CTMC of Fig. 9. Before providing a general expression, we discuss a numerical example. Let $c = 10$ and $c^* = 4$. Therefore, there are $c_a = c - c^* = 6$ resources always available for service requests because at most 4 resources can be reconfigured at the same time. Thus, when the number of service requests in the system is at most c_a, the average aggregate departure rate is equal to the number of requests multiplied by μ (i.e., $\delta_k = k$ for $k = 1, \ldots, 6$). Consider, for example, that there are 8 service requests in the system, thus $c_a < k < c$. If 0, 1, or 2 resources are being reconfigured, and this happens with probability $p_0^r + p_1^r + p_2^r$, there are enough resources for all service requests in the system, and the aggregate departure rate is 8μ. If three resources are being reconfigured, and this happens with probability p_3^r, there is only one resource, beyond the six, that can be used for service requests. So, the aggregate departure rate is $(6+1)\mu = 7\mu$. For the same reason, if four resources are being reconfigured, there are only 6 available resources and the aggregate departure rate is 6μ. Table 2 shows the departure rates for all states in the example considered here.

The expression for δ_k can be generalized as shown below. Note that $\delta_k = \delta_c$ for $k = c+1, \ldots$, and that δ_c is the average number of resources that are not being reconfigured (e.g., see the expression for state 10 in Table 2), and can be used to serve service requests. The ratio $(\rho \cdot \bar{c}/\delta_c) = \lambda/(\mu \cdot \delta_c)$ can be interpreted as the average utilization of the resources.

$$\delta_k = \begin{cases} k & k = 1, \ldots, c_a \\ c_a + \sum_{j=1}^{k-c_a-1} j \cdot p^r_{c^*-j} + & \\ \qquad (k - c_a) \sum_{j=0}^{c-k} p_j^r & k = c_a+1, \ldots, c \\ c_a + \sum_{j=1}^{c^*} j . p^r_{c^*-j} & k = c+1, \ldots \end{cases} \tag{23}$$

As Fig. 9 shows, the transition rate from state k to $k+1$ is λ, the average arrival rate of requests to the system, and the transition rate β_k from a state k to state $k-1$ is given by

$$\beta_k = \begin{cases} \mu\ \delta_k & k < c \\ \mu\ \delta_c & k \geq c \end{cases} \tag{24}$$

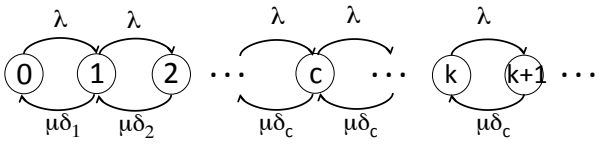


Fig. 9. State transition diagram for the response time model

TABLE 2
Example of the aggregate departure rate for $c = 10$ and $c^* = 4$

State	Departure rate
$k,\ k = 1, \ldots, 6$	$k\mu$
7	$6\mu + \mu(p_0^r + p_1^r + p_2^r + p_3^r)$
8	$6\mu + \mu p_3^r + 2\mu(p_0^r + p_1^r + p_2^r)$
9	$6 + \mu p_3^r + 2\mu p_2^r + 3\mu(p_0^r + p_1^r)$
10	$6\mu + \mu p_3^r + 2\mu p_2^r + 3\mu p_1^r + 4\mu p_0^r$
$k,\ k = 11, \ldots$	$6\mu + \mu p_3^r + 2\mu p_2^r + 3\mu p_1^r + 4\mu p_0^r$

We can now use the generalized birth-death equations (see Eqs. 7 and 8) to solve for P_k and P_0. We have to break down the expression for P_k into two parts (for $k = 1, \ldots, c$ and $k > c$) because β_k has two expressions. Hence, for $k = 1, \ldots, c$

$$P_k = P_0 \prod_{i=0}^{k-1} \frac{\lambda}{\mu\ \delta_{i+1}} = P_0\ \frac{(\lambda/\mu)^k}{\prod_{i=0}^{k-1} \delta_{i+1}} = P_0\ \frac{(\rho.\bar{c})^k}{\prod_{i=0}^{k-1} \delta_{i+1}} \tag{25}$$

and, for $k = c+1, \ldots$

$$\begin{aligned} P_k &= P_0 \prod_{i=0}^{c-1} \frac{\lambda}{\mu\ \delta_{i+1}} \prod_{i=c}^{k-1} \frac{\lambda}{\mu\ \delta_c} = P_0 \frac{(\rho \cdot \bar{c})^k}{\delta_c^{k-c} \prod_{i=0}^{c-1} \delta_{i+1}} \\ &= P_0 \frac{\rho^k \cdot \delta_c^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \end{aligned} \tag{26}$$

P_0 can now be computed as

$$P_0 = \left[1 + \sum_{k=1}^{c} \frac{(\rho \cdot \bar{c})^k}{\prod_{i=0}^{k-1} \delta_{i+1}} + \sum_{k=c+1}^{\infty} \frac{\delta_c^c \rho^k}{\prod_{i=0}^{c-1} \delta_{i+1}} \right]^{-1} \tag{27}$$

If we move $\delta_c^c / \prod_{i=0}^{c-1} \delta_{i+1}$ out of the infinite summation in the above expression, we are left with the following geometric series, which converges for $\rho < 1$:

$$\sum_{k=c+1}^{\infty} \rho^k = \frac{\rho^{c+1}}{1-\rho} \tag{28}$$

Hence, P_0 can be easily computed as follows:

$$P_0 = \left[1 + \sum_{k=1}^{c} \frac{(\rho \cdot \bar{c})^k}{\prod_{i=0}^{k-1} \delta_{i+1}} + \frac{\delta_c^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \frac{\rho^{c+1}}{1-\rho} \right]^{-1} \tag{29}$$

Note that Eqs. 25, 26, and 29 simplify to the well-known equations for the M/M/c queue [15] when $c^* = 0$. The average number N_s of requests in the system can be computed as

$$N_s = \sum_{k=1}^{\infty} k \cdot P_k = \sum_{k=1}^{c} k \cdot P_k + \sum_{k=c+1}^{\infty} k \cdot P_k. \tag{30}$$

The first summation in the expression above is an easy-to-compute finite summation:

$$P_0 \sum_{k=1}^{c} k \frac{(\rho \cdot \bar{c})^k}{\prod_{i=0}^{k-1} \delta_{i+1}}. \tag{31}$$

The infinite summation in Eq. 30 can be written as

$$P_0 \frac{\delta_c^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \sum_{k=c+1}^{\infty} k \cdot \rho^k \tag{32}$$

which can be computed as

$$P_0 \frac{\delta_c^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \left[\rho \frac{\partial}{\partial \rho} \sum_{k=c+1}^{\infty} \rho^k \right] \tag{33}$$

and is equal to

$$P_0 \frac{\delta_c^c}{\prod_{i=0}^{c-1} \delta_{i+1}} \cdot \frac{\rho^{c+1}}{1-\rho} \left[\frac{\rho}{1-\rho} + 1 + c \right]. \tag{34}$$

Thus, Eqs. 31 and 34 allow us to compute N_s. Finally, using Little's Law we can compute the average response time R as $R = N_s/\lambda$.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

7 Combined Model

We now consider the fact that when a reconfiguration request arrives to a resource, it may be busy serving a service request. In this case, the reconfiguration has to wait until the resource becomes idle. This affects the rate at which reconfigurations occur. Let α' be the effective reconfiguration rate, i.e., the rate at which reconfigurations occur. This effective reconfiguration rate should be used to compute the reconfiguration probabilities p_k^r. The reconfiguration rate is equal to the inverse of the average time between reconfigurations. Thus,

$$1/\,\alpha' = (1/\,\alpha) \cdot Pr[\text{idle}] + (1/\,\alpha + T_{\text{res}}) \cdot (1 - Pr[\text{idle}]) \quad (35)$$

where $Pr[\text{idle}]$ is the probability that a resource is idle when it is time to reconfigure and T_{res} is the average residual service time when the resource is busy. From renewal theory,

$$T_{\text{res}} = E[\tilde{T}^2]/2E[\tilde{T}] \quad (36)$$

where $\tilde{T}$ is the random variable that represents the service time [15]. If that variable is exponentially distributed, $T_{\text{res}} = E[\tilde{T}] = T$ due to the memoryless property of the exponential distribution. Let $p = 1 - Pr[\text{idle}]$. We can then compute p as a function of the probabilities P_k using the law of total probability as indicated by the equation below that shows values of p and their corresponding probabilities.

$$p = \begin{cases} 0 & P_0 \\ k/c & P_k,\ k = 1, \ldots, c-1 \\ 1 & 1 - \sum_{k=0}^{c-1} P_k. \end{cases} \quad (37)$$

The explanation behind Eq. 37 is the following. When there are no service requests in the system, and this happens with probability P_0, the probability that a reconfiguration request finds the resource busy is zero. When all resources are busy, and this happens with probability $1 - \sum_{k=0}^{c-1} P_k$, the probability that a reconfiguration request for a specific resource finds the resource busy is 1. When k $(k = 1, \ldots, c-1)$ resources are busy, the probability that a reconfiguration request finds a specific resource busy is equal to 1 minus the probability that it finds the resource idle. Thus, the probability that the resource is busy is equal to

$$1 - \binom{c-1}{k} / \binom{c}{k} = \frac{k}{c} \quad (38)$$

because the the probability that the resource is idle is equal to the number of ways one can choose k resources to be busy out of the remaining $c-1$ resources divided by the total number of ways one can select k resources to be busy out of c resources. Thus, using the Law of Total Probability we get,

$$\begin{aligned} p &= 0 \times P_0 + \left[\frac{1}{c}\sum_{k=1}^{c-1} k \cdot P_k\right] + 1 \times (1 - \sum_{k=0}^{c-1} P_k) \\ &= \frac{1}{c}\sum_{k=1}^{c-1} k \cdot P_k + (1 - \sum_{k=0}^{c-1} P_k) \end{aligned} \quad (39)$$

We can now rewrite Eq. 35 as

$$\alpha' = \frac{\alpha}{1 + \alpha p T} \quad (40)$$

Because p is a function of $\{P_k\}$ (see Eq. 39), and $\{P_k\}$ is a function of $\{p_k^r\}$ (see Section 6), and $\{p_k^r\}$ is a function of α' (see Sections 5.A-C) and α' is a function of p (see Eq 35), it follows that $p = f(p)$ for some function f. Thus, p is a fixed point of f. We can solve for p using the iterative algorithm described next. Let $\mathcal{S}(\{p_k^r\}, c, c^*, \lambda, T)$ be the service response time model (see Section 6) that computes the probabilities P_k and let $\mathcal{R}(c, c^*, \alpha, S)$ be the reconfiguration model (see Section 5) that computes the probabilities p_k^r. The following iterative algorithm can be used to solve this fixed point problem. The busy probability p is initially set to zero and it is recomputed at Step 5. The difference between the values of p in successive iterations is checked at Step 6 against a given tolerance ξ.

- **Step 1**. Initialize: $i \leftarrow 0$; $p^i \leftarrow 0$;
- **Step 2**. Compute α': $\alpha' \leftarrow \alpha/(1 + \alpha p^i T)$;
- **Step 3**. Compute the reconfiguration probabilities p_k^r: $\{p_k^r\} \leftarrow \mathcal{R}(c, c^*, \alpha', S)$;
- **Step 4**. Compute the service request probabilities P_k: $\{P_k\} \leftarrow \mathcal{S}(\{p_k^r\}, c, c^*, \lambda, T)$;
- **Step 5**. Increment iteration count and compute new value of the busy probability: $i \leftarrow i + 1$; $p^i \leftarrow \frac{1}{c}\sum_{k=1}^{c-1} k \cdot P_k + (1 - \sum_{k=0}^{c-1} P_k)$;
- **Step 6**. Check tolerance: if $|\ \frac{p^i - p^{i-1}}{p^i}\ | > \xi$ go to Step 2;
- **Step 7**. Compute the average response time R as a function of the probabilities P_k.

8 Simulation and Experimental Testbed

Our analytical results were validated by simulation and by experiments that implemented a shuffling MTD. We implemented the simulation using SimPy[1], a process-based discrete-event simulation framework based on standard Python. SimPy supports multiple processes that contend for access to a resource and automatically handles queuing of events if a resource is busy, making it ideal for our purposes. Additionally, we used SimPy as a real-time event generator to control VM reconfigurations and implement a fully operational MTD. For our VM environment, we used Citrix's open-source XenServer platform[2], which offers pooling of resources, the ability to quickly clone VMs for reconfiguration, and a command-line interface that is compatible with our simulation framework.

Our MTD controller runs on a separate server and starts an independent process for each VM – either a simulated VM or an actual VM in the XenServer pool – that generates a reconfiguration request. The simulator lets us choose how each of our random values are generated. For the analytical model and the simulations used to validate it, we used an exponentially distributed reconfiguration duration S with average value of 120 sec. In our experiments, it turns out that S can vary based on the values of c and α and is normally distributed, so we instead used a normal distribution for S with the same mean and standard deviation as was observed in the experiment for the corresponding value of α.

Reconfigurations can consist of a number of possible actions, including changing the IP address or software. In our experiments, we remove the instance of a VM from the virtual network and replace it with a fresh copy, similar to how SCIT operates [16]. The fresh copy also has a new IP address obtained from DHCP, enabling a basic IP-hopping scheme. Our reconfiguration process also collects statistics such as percentage of requests dropped and all possible delays.

The MTD controller also serves as a traffic generator that creates service requests to our VMs. Each service request is

1. Available at https://simpy.readthedocs.io/en/3.0/.
2. Available at https://xenserver.org/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

an independent process with exponentially distributed interarrival times with an average arrival rate equal to λ and average service time T in the simulations. For the experiments, an HTTP request is sent to an idle VM, which has a scripted delay on the HTTP response with average time T to simulate the time to process a generic service request. Each process records the time at which it was generated, began service, and completed according to the environment's internal clock. These records are used to compute queue time, service time, and response time and are maintained for each request.

We also collected statistics from a separate monitor process that operates at set intervals to collect information about the number of resources idle, in use, and being reconfigured, as well as the current queue length. An overview of the system and processes is shown in Fig. 10.

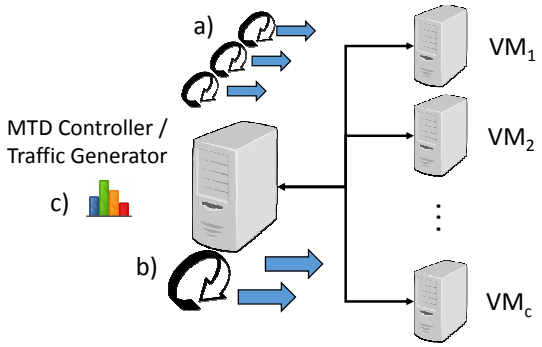


Fig. 10. Experimental setup: a) c independent processes to generate reconfiguration requests (arrival rate α), b) 1 process to generate independent service requests (arrival rate λ), c) Monitor process (every 0.01 sec)

The pool of VMs is tracked using 3 separate states for the VMs: *idle*, *in use* (i.e., serving a request), or *being reconfigured*. All requests for a VM must first acquire a shared resource that gives them access to the pool of idle VMs. A priority queue is used, giving priority to reconfiguration requests so that reconfiguration is not unnecessarily delayed; however, reconfiguration requests for a specific VM will not preempt a request currently being served. Instead, the reconfiguration request flags that VM for reconfiguration and then releases its lock on the idle pool before waiting for that resource to appear in the pool of VMs to reconfigure. When service requests receive access to the idle pool, they remove a random VM from the pool and place it in the pool of VMs in use. Once completing the request, if that VM is flagged for reconfiguration, it is placed in the reconfiguration pool where the reconfiguration request will pick it up for reconfiguration, otherwise it is placed back in the idle pool. In the event that a service request finds no VMs in the idle pool, it waits for one to appear. This additional wait is included in the overall queue time. The overall flow of control and VM state transitions is shown in Fig. 11.

Each iteration of the simulation lasted 6,000 seconds, with no statistics recorded in the first 1,000 seconds to allow the system to achieve steady-state. Thirty runs were performed for values of α from 0.001 to 0.050 to obtain the mean, standard deviation, and 95% confidence intervals for the mean for each statistic. For the experiments, each run is limited to 600 seconds with statistics recorded after the first 60 seconds for select values of α. The values of the other input parameters used in the simulations and experiments are given in Table 3.

TABLE 3
Values of variables used in the simulation and experiments

Variable	Description
c	20
c^*	14
α	from 0.001 to 0.050 req/sec
S	120 sec
λ	10 requests/sec
T	0.5 sec
T_s	300 sec

9 Numerical Results and Validation

This section presents several numerical results starting with results obtained with the analytic model. Then, simulation is used to validate the analytic results. In what follows, simulation and experimental results are compared. Finally, we show how the analytic model can be used to find an optimal value of the reconfiguration rate that considers tradeoffs between response time and security.

9.1 Analytic Model Results

There are some important tradeoffs illustrated by the equations derived in Sections 5 and 6. First, as the reconfiguration rate α increases, less time is given for an attacker to succeed, but the resource availability decreases and both the probability that a request for a resource has to queue and the response time increase. We illustrate these tradeoffs by using the equations above in a variety of numerical examples.

Figs. 12 and 13 show the response time and average age of the resources for both policies. For very low reconfiguration rates ($\alpha \to 0$) all c resources are available and the two policies exhibit a similar behavior. As α increases, the wait policy ensures that a reconfiguration request is honored as soon as $k < c^*$, whereas the drop policy drops requests generated when $k \geq c^*$. Therefore, the wait policy reconfigures more often than the drop policy and, as a consequence, its response time is higher (see Fig. 12) and the average age of resources is lower compared to the drop policy (see Fig. 13). For example, for $\alpha = 0.05$ rec/sec, the average response time of service requests for the wait policy is 17.5% higher and the average age of a resource for the drop policy is

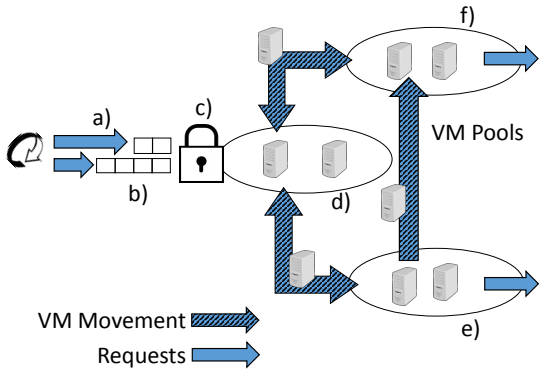


Fig. 11. Control Flow and VM Movement: a) incoming requests, b) priority queue, c) resource lock on idle pool, d) idle VM pool, e) VMs in use, f) reconfiguration VMs

57% higher than the corresponding values for the wait policy. This illustrates the tradeoff we discussed above in the sense that the wait policy always exhibits a worse response time than the drop policy but it exhibits a lower resource age, which reduces the probability of an attacker's success. Note that, with a policy that limits the maximum number of resources reconfiguring at a time, the response time and resource age are also limited as a function of the average reconfiguration rate α.

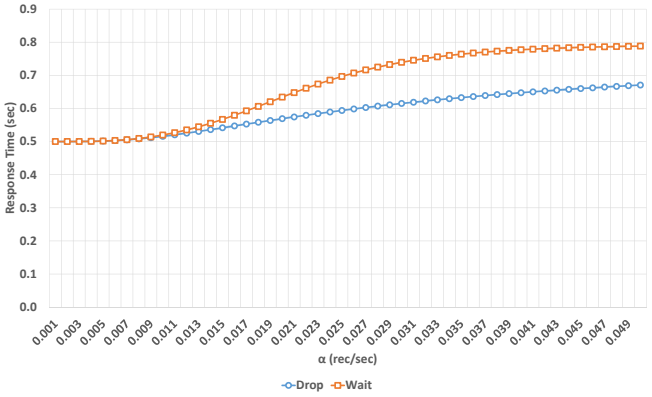


Fig. 12. Average response time for drop and wait policies

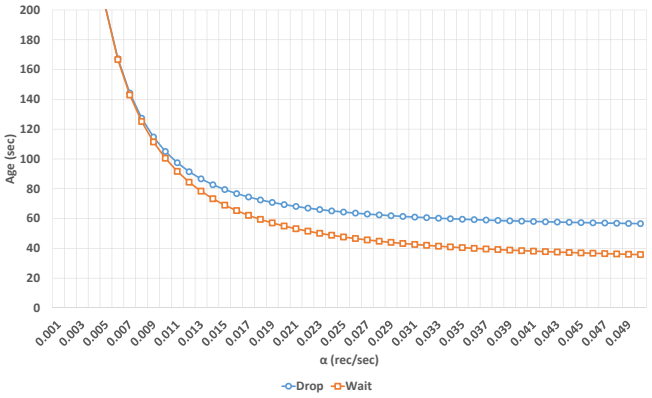


Fig. 13. Average resource age for drop and wait policies

Figs. 14 and 15 show average response times for other values of c^*. When $c^* > 14$, we observe that for values of α above a certain threshold, the system becomes unable to handle all incoming service requests because of the scarcity of resources, causing the queue to grow indefinitely. Therefore, we must choose a value of c^* such that $c - c^* > \lambda \cdot T$, leading to our choice of $c^* = 14$ for $\lambda = 10, T = 0.5$.

Fig. 16 shows the distribution of the number k of resources being reconfigured – out of a total of 20 resources – for several values of the reconfiguration rate α, using the drop policy and for $c^* = 14$. The graphs show that the distribution is bell-shaped for low values of α. But, as α increases from 0.02 up to 0.04 rec/sec, the probability distribution shifts to the right. The average number of resources being reconfigured is 7.49 for $\alpha = 0.005$ rec/sec and 13.47 for $\alpha = 0.04$ rec/sec. When $\alpha = 0.04$ rec/sec, the system spends 62.2% of the time with the maximum number of resources being reconfigured.

Similarly, Fig. 17 shows the same trend for the wait policy. Because, as explained above, with the wait policy, all reconfig-

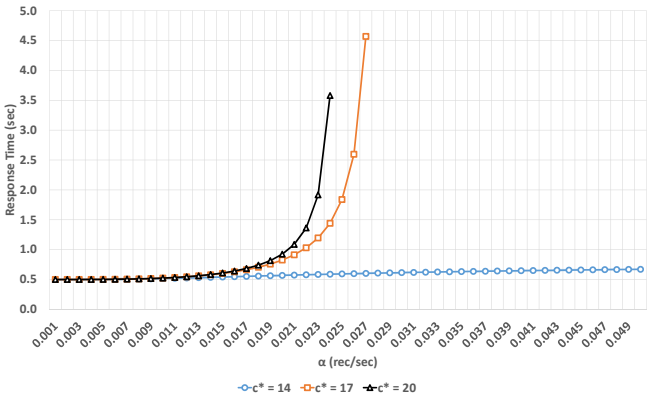


Fig. 14. Average response time for varying levels of c^* (Drop Policy)

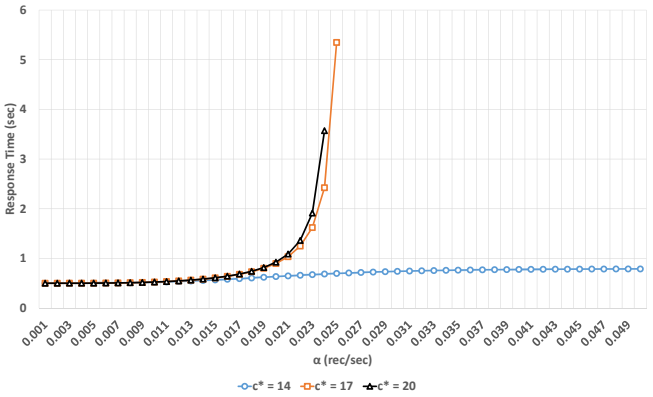


Fig. 15. Average response time for varying levels of c^* (Wait Policy)

uration requests will eventually be served, on average there are more active such requests in the system. The average number of resources being reconfigured is 7.50 for $\alpha = 0.005$ rec/sec but rises to 13.96 for $\alpha = 0.04$ rec/sec. When $\alpha = 0.04$ rec/sec, the system spends 97.2% of the time with the maximum number of resources being reconfigured.

9.2 Validation with Simulation Results

This section presents results validating the analytic model using the simulation described in Section 8. All simulation curves show

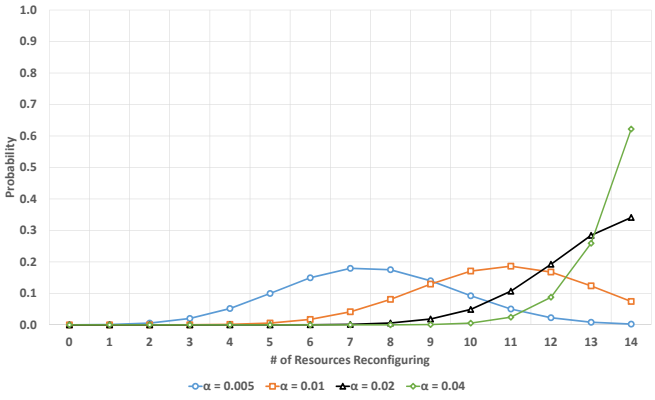


Fig. 16. Probability distribution of p_k for varying levels of α (drop policy)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

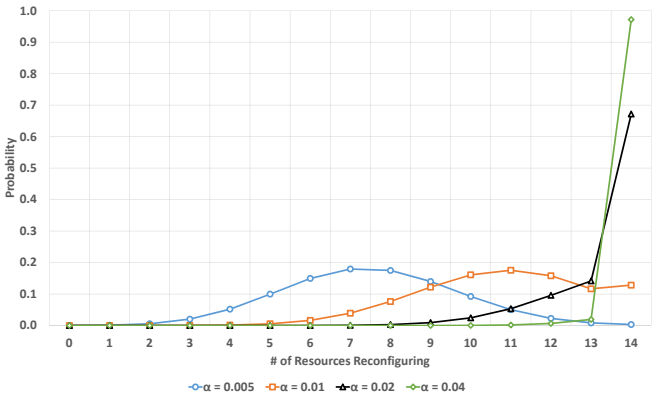


Fig. 17. Probability distribution of $\hat{p}_k$ for varying levels of α (wait policy)

95% confidence intervals, and show that the analytic model results match the simulations very closely. Fig. 18 shows simulation and analytic results for the average response time of requests under the drop policy for a range of values of α. The maximum absolute percent relative error is 8.65% and occurs for $\alpha = 0.21$ rec/sec.

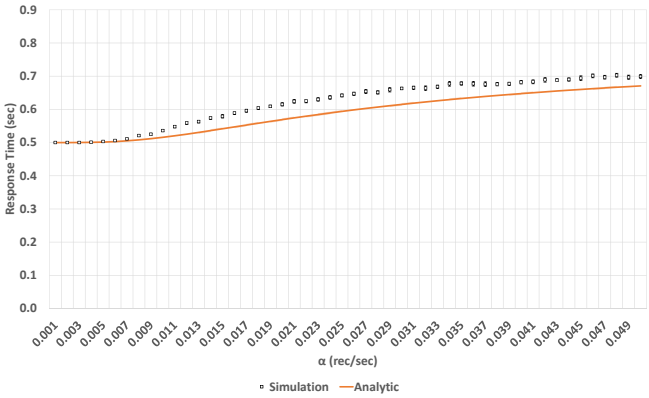


Fig. 18. Response time: simulation vs. analytical model (drop policy)

Similarly, Fig. 19 compares results for the average age of a resource. The maximum absolute percent relative error is below 10% for all values of α. Finally, Fig. 20 compares the percentage of dropped reconfiguration requests. For most values of α, the maximum absolute percent relative error is below 5%.

A similar analysis was conducted to validate the wait policy, and confirmed that simulation an analytical results closely match. A detailed discussion is omitted for reasons of space and we only show a comparison between simulation and analytical results for the average waiting time to start a reconfiguration. As shown in Fig. 21, the results match closely across the entire range of α values.

9.3 Validation of the Simulation with Experimental Results

We now describe the validation of the simulation model with experimental results obtained using the setup described in Section 8. Tables 4 and 5 compare simulation and experimental results for availability and response time for the drop and wait policies respectively, for values of α ranging from 0.005 to 0.050 rec/sec. The average values and corresponding 95% confidence

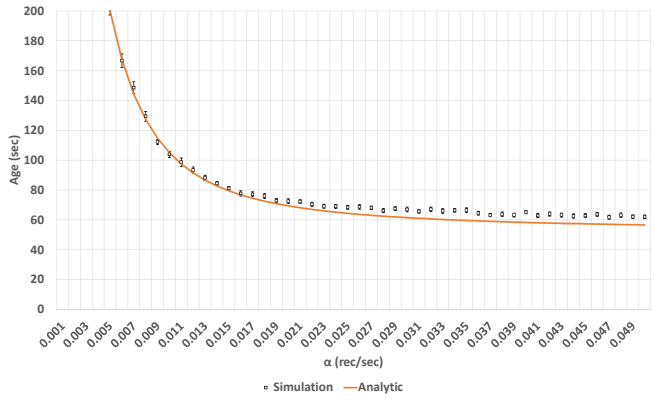


Fig. 19. Average resource age: simulation vs. analytical model (drop policy)

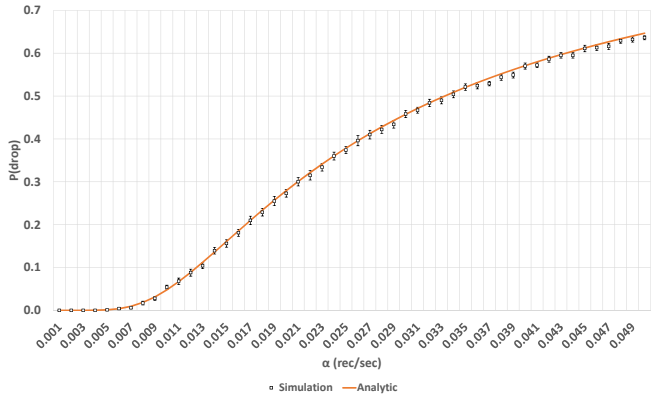


Fig. 20. Drop percentage: simulation vs. analytical model (drop policy)

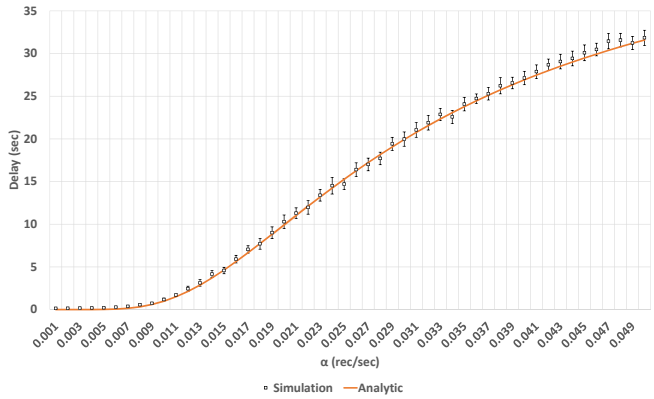


Fig. 21. Reconfiguration delay: simulation vs. analytical model (wait policy)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

intervals are shown in the tables. The third column of each set of results shows the absolute percent relative error computed as $100 \times (\text{simulation} - \text{experiment})/\text{simulation}$. As we can see from the tables, errors are low and are below 5% in all cases except for one case in which the error is 7.8%. These results, along with the findings in the previous subsection, validate the analytical results.

9.4 Determining the Optimal Reconfiguration Rate

The analytic model presented in Sections 3-7 allows one to predict the response time and the average age of each resource in the system. We can then use these results to answer questions such as "Given objective values for response time and level of protection, what is the reconfiguration rate that maximizes overall utility?"

We can solve this by estimating the attacker's likelihood of success using the method described in Section 7. For instance, we can use the linear method, with T_s = 300, and $P_s(t) = 1$ when $t \geq T_s$. Next, we can assign utility values to the response time and attacker's likelihood of success using the following sigmoid functions:

$$U_R(t_r) = \frac{e^{\sigma(-t_r+\beta_R)}}{1+e^{\sigma(-t_r+\beta_R)}} \tag{41}$$

$$U_S(ps) = \frac{e^{\sigma(-ps+\beta_S)}}{1+e^{\sigma(-ps+\beta_S)}} \tag{42}$$

where t_r is the response time, β_R is the response time objective, ps is the attacker's success probability computed as $P_s(\text{age})$ where age — the resource's average age — is either given by Eq. (12) or (22) for the drop or wait policies, β_S is the attacker's success probability objective, and σ is a steepness parameter for the sigmoid. Sigmoids are commonly used as utility functions in autonomic computing because they are smooth, differentiable at all points, and can be easily adjusted to react more or less aggressively to violations of service level agreements through the steepness parameter [19]. We can now compute a global utility function U_g as:

$$U_g = w_R \cdot U_R(t_r) + w_S \cdot U_S(ps) \tag{43}$$

where w_R and w_S are weight factors chosen such that $w_R + w_S = 1$. Different values of w_R and w_S influence the optimal reconfiguration rate. For example, Fig. 22 shows the overall utility values for the drop policy when T_S = 300 sec, β_R = 55 sec, β_S = 0.2, and σ = 10. When $w_R = w_S$, the optimal value is found at $\alpha = 0.018$ rec/sec. When $w_R = 0.75$, denoting an emphasis on response times at the cost of protection, the optimal value can be found at $\alpha = 0.018$ rec/sec, and when $w_S = 0.75$, denoting an emphasis on protection at the cost of response times, the optimal value is 0.041 rec/sec.

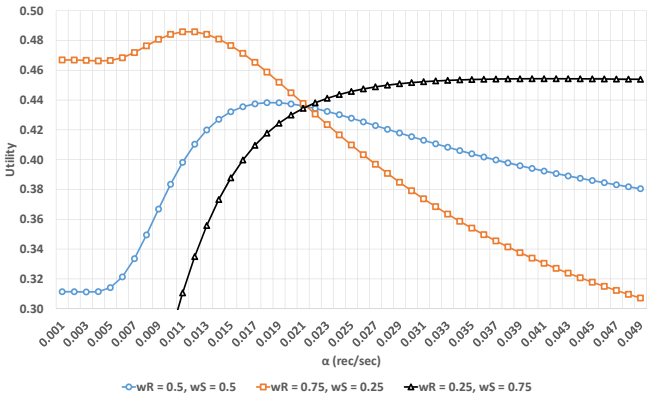


Fig. 22. Utility values for various weight combinations (drop policy)

10 Related Work

A vast array of different MTD techniques have been proposed in recent years, with the majority of such techniques designed to protect systems against a very narrow set of attack vectors such as SQL injection [8] and data exfiltration [7]. A rich line of research has focused on MTD techniques to mitigate distributed DoS attacks [5], [6] by deploying proxies between clients and servers, and periodically reconfiguring – either proactively or in response to detected threats – the associations between clients and proxies in order to disrupt knowledge accumulated by adversaries. In [14], the authors indicate that in order to support analysis, a theory of MTDs "should include an analytic model that can be used by designers to determine how different parameter settings will impact security." This is precisely what our paper achieves, i.e., it allows designers to select reconfiguration rates and reconfiguration techniques according to their reconfiguration time in order to balance security and performance tradeoffs. The authors of [17] presented a theory of cyber attacks, which they used to formalize several attack types including a Web-based planner with several backend databases and Address Space Layout Randomization. Xu *et al.* attempt to bridge the gap between low-level attack-based experiments focusing on specific programs and higher-level probabilistic models and simulations by proposing a method using three layers of state machines to capture interactions between programs [20].

Duan *et al.* [21] present a proactive Random Route Mutation technique to randomly change the route of network flows to defend against eavesdropping and DoS attacks. Jafarian *et al.* [22] use an IP virtualization scheme based on virtual DNS entries and Software Defined Networks. Their goal is to hide network assets from scanners. Using OpenFlow, each host is associated with a range of virtual IP addresses and mutates its IP address within its pool. In Chapter 8 of [2], an approach based on diverse virtual servers is presented. Each server is configured with a set of software stacks, and a rotational scheme is employed for substituting different software stacks for any given request.

Deception-based approaches aim at misleading reconnaissance tools, such as *nmap* or *Xprobe2*. Such tools can identify a service or an operating system by analyzing packets that can reveal implementation specific details about the host [23], [24]. Reconnaissance tools store known system's features and compare them against the scan responses in order to match a fingerprint. Watson *et al.* [24] adopted protocol scrubbers in order to avoid revealing implementation-specific information and restrict an attacker's ability to determine the operating system of a protected host.

With respect to quantification of MTD techniques, several different metrics have been proposed in the literature to measure their effectiveness. Some authors assess the performance of their techniques in terms of the attacker's success rate [25], while others introduce deception, deterrence, and detectability metrics [26]. Still others utilize a total of 8 metrics (Productivity, Success, Confidentiality, and Integrity) for the attacker and defender [27], leading to confusion over the multiple dimensions.

Markov models have been used for years in a wide variety

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TDSC.2018.2882825, IEEE Transactions on Dependable and Secure Computing

TABLE 4
Simulation vs. experimental results for availability

	Drop Policy			**Wait Policy**		
α	Simulation	Experimental	Error	Simulation	Experimental	Error
0.005	0.706 ± 0.013	0.701 ± 0.013	0.71%	0.703 ± 0.014	0.701 ± 0.015	0.28%
0.010	0.487 ± 0.012	0.495 ± 0.02	1.64%	0.470 ± 0.015	0.473 ± 0.018	0.64%
0.015	0.391 ± 0.006	0.381 ± 0.007	2.56%	0.357 ± 0.009	0.359 ± 0.011	0.56%
0.020	0.360 ± 0.003	0.350 ± 0.004	2.78%	0.319 ± 0.005	0.311 ± 0.003	2.51%
0.025	0.342 ± 0.003	0.335 ± 0.002	2.05%	0.308 ± 0.002	0.305 ± 0.002	0.97%
0.030	0.335 ± 0.003	0.331 ± 0.003	1.19%	0.303 ± 0.001	0.301 ± 0.000	0.66%
0.040	0.322 ± 0.001	0.321 ± 0.001	0.31%	0.301 ± 0.001	0.300 ± 0.000	0.33%
0.050	0.317 ± 0.001	0.316 ± 0.001	0.32%	0.301 ± 0.000	0.300 ± 0.000	0.33%

TABLE 5
Simulation vs. experimental results for response time

	Drop Policy			**Wait Policy**		
α	Simulation	Experimental	Error	Simulation	Experimental	Error
0.005	0.503 ± 0.002	0.506 ± 0.003	0.60%	0.507 ± 0.004	0.508 ± 0.002	0.20%
0.010	0.533 ± 0.008	0.532 ± 0.010	0.19%	0.544 ± 0.010	0.558 ± 0.017	2.57%
0.015	0.605 ± 0.014	0.594 ± 0.014	1.82%	0.664 ± 0.025	0.673 ± 0.029	1.36%
0.020	0.636 ± 0.013	0.651 ± 0.017	2.36%	0.731 ± 0.023	0.788 ± 0.030	7.80%
0.025	0.667 ± 0.016	0.683 ± 0.017	2.40%	0.801 ± 0.027	0.793 ± 0.024	1.00%
0.030	0.679 ± 0.012	0.687 ± 0.021	1.18%	0.791 ± 0.025	0.805 ± 0.020	1.77%
0.040	0.718 ± 0.019	0.713 ± 0.021	0.70%	0.806 ± 0.022	0.816 ± 0.029	1.24%
0.050	0.725 ± 0.019	0.758 ± 0.025	4.55%	0.798 ± 0.029	0.819 ± 0.031	2.63%

of applications; however their application in security domains has been limited. The model presented in [28] is the closest to our work and introduces a discrete-time Markov-model-based framework for MTD analysis. The framework allows modeling of a broad range of MTD strategies, and presents results on how the probability of an adversary defeating an MTD strategy is related to the time/cost spent by the adversary. They also show how their approach can be used to analyze a composition of MTDs. However, differently from our model, the approach proposed in [28] cannot predict the performance of deployed MTD techniques, which the authors list as future work.

While several metrics have been offered, they do not attempt to evaluate more than a few select MTDs chosen for a specific study. One expert survey provides a thorough assessment of the effectiveness and cost of many techniques across the spectrum of existing MTDs [29]. However, the survey is qualitative in nature and potentially subject to reviewer bias.

Our work is also inspired by research in the field of autonomous systems, particularly self-protecting systems [30][31]. These systems change their settings in response to their environment. This concept can be seen as a form of moving target defense. To change their settings effectively, self-protecting systems must be able to quantify both their effectiveness and their cost or overhead in order to provide an accurate measure of their utility.

11 Conclusions and Future Work

Moving Target Defense (MTD) has recently emerged as one of the potentially game-changing themes in cyber security. While the typical asymmetry of the security landscape tends to favor the attacker, MTD holds promise to change the game in favor of the defender. Thus, MTD has received significant attention in the last decade, prompting researchers and practitioners to develop a myriad of different MTD techniques. Unfortunately, most such techniques are designed to address a very narrow set of attack vectors. Additionally, despite the significant progress made in this area, the problem of studying and quantifying the cost and benefits associated with the deployment of MTD techniques has not received sufficient attention, and shared metrics to assess the performance of MTD techniques are still lacking.

Our preliminary work [10] provided a first important step toward addressing some of these limitations. This paper significantly extended that work by introducing analytic models to analyze MTDs that allow for limits to be placed on the number c^* of resources being reconfigured at the same time. We also presented and modeled two policies (drop and wait) that deal, in different ways, with reconfiguration requests that arrive when the limit on the number of resources being reconfigured is reached.

Our analytic models allow us to compute resource availability, response time, and attacker's success probability as well as trade-offs between these metrics as a function of a variety of parameters. Our models also allow us to determine the optimal reconfiguration rate that maximizes a utility function, which is a function of both response time and attacker's success probability. We are currently investigating the design of autonomic controllers that dynamically vary the threshold c^* to maximize this utility function. As part of work-in-progress, we are looking at how to relax some of our assumptions, for example that all resources are homogeneous (e.g., they have the same average reconfiguration rate and the same average processing time).

References

[1] Executive Office of the President, National Science and Technology Council, "Trustworthy cyberspace: Strategic plan for the federal cybersecurity research and development program," http://www.whitehouse.gov/, December 2011.

[2] S. Jajodia, A. K. Ghosh, V. Swarup, C. Wang, and X. S. Wang, Eds., *Moving Target Defense: Creating Asymmetric Uncertainty for Cyber Threats*, 1st ed., ser. Advances in Information Security. Springer, 2011, vol. 54.

[3] A. Avižienis, J.-C. Laprie, B. Randell, and C. Landwehr, "Basic concepts and taxonomy of dependable and secure computing," *IEEE Tr. Dependable and Secure Computing*, vol. 1, no. 1, pp. 11–33, January-March 2004.

[4] K. S. Trivedi, D. S. Kim, A. Roy, and D. Medhi, "Dependability and security models," in *Proc. 7th Intl. Workshop on Design of Reliable Communication Networks*, October 2009, pp. 11–20.

[5] Q. Jia, K. Sun, and A. Stavrou, "MOTAG: Moving target defense against internet denial of service attacks," in *Proc. 22nd Intl. Conf. Computer Communications and Networks (ICCCN 2013)*, August 2013.

[6] Q. Jia, H. Wang, D. Fleck, F. Li, A. Stavrou, and W. Powell, "Catch me if you can: a cloud-enabled DDoS defense," in *Proc. 44th Annual IEEE/IFIP Intl. Conf. Dependable Systems and Networks (DSN 2014)*, June 2014, pp. 264–275.

[7] S. Venkatesan, M. Albanese, G. Cybenko, and S. Jajodia, "A moving target defense approach to disrupting stealthy botnets," in *Proc. 3rd ACM Workshop on Moving Target Defense (MTD)*, Vienna, Austria, October 2016.

[8] S. W. Boyd and A. D. Keromytis, "SQLrand: Preventing SQL injection attacks," in *Proceedings of the 2nd Conference on Applied Cryptography and Network Security (ACNS)*, ser. Lecture Notes in Computer Science, vol. 3089. Yellow Mountain, China: Springer, June 2004, pp. 292–302.

[9] D. A. Menascé, "Security performance," *IEEE Internet Computing*, vol. 7, no. 3, pp. 84–87, May/June 2003.

[10] W. Connell, D. A. Menascé, and M. Albanese, "Performance modeling of moving target defenses," in *Proc. 4th ACM Workshop on Moving Target Defense (MTD)*, Dallas, TX, USA, October 2017, pp. 53–63.

[11] P. K. Manadhata and J. M. Wing, "An attack surface metric," *IEEE Tr. Software Engineering*, vol. 37, no. 3, pp. 371–386, May 2011.

[12] M. Albanese, E. Battista, S. Jajodia, and V. Casola, "Manipulating the attacker's view of a system's attack surface," in *IEEE Conf. Communications and Network Security (CNS 2014)*, San Francisco, CA, USA, October 2014, pp. 472–480.

[13] F. H. Abbasi, R. J. Harris, G. Moretti, A. Haider, and N. Anwar, "Classification of malicious network streams using honeynets," in *Proc. IEEE Conf. Global Communications (GLOBECOM 2012)*. Anaheim, CA, USA: IEEE, December 2012, pp. 891–897.

[14] R. Zhuang, S. DeLoach, and X. Ou, "Towards a theory of moving target defenses," in *Proc. First ACM Workshop on Moving Target Defense (MTD)*, Scottsdale, AZ, USA, November 2014.

[15] L. Kleinrock, *Queueing Systems. Volume 1: Theory*. Wiley-Interscience, 1975.

[16] A. K. Bangalore and A. K. Sood, "Securing web servers using self cleansing intrusion tolerance (SCIT)," in *2nd Intl. Conf. Dependability (DEPEND'09)*. IEEE, 2009, pp. 60–65.

[17] R. Zhuang, A. Bardas, S. DeLoach, and X. Ou, "A theory of cyber attacks: A step towards analyzing MTD systems," in *Proc. 2nd ACM Workshop on Moving Target Defense (MTD)*, Denver, CO, Oct. 2015.

[18] J. P. Buzen, "Fundamental operational laws of computer system performance," *Acta Informatica*, vol. 7, no. 2, 1976.

[19] W. Walsh, G. Tesauro, J. Kephart, and R. Das, "Utility functions in autonomic computing," in *Proc. IEEE Intl. Conf. Autonomic Computing*, ser. ICAC '04. IEEE Computer Society, 2004.

[20] J. Xu, P. Guo, M. Zhao, R. Erbacher, M. Zhu, and P. Liu, "Towards a theory of moving target defenses," in *Proc. First ACM Workshop on Moving Target Defense (MTD)*, Scottsdale, AZ, USA, November 2014.

[21] Q. Duan, E. Al-Shaer, and H. Jafarian, "Efficient random route mutation considering flow and network constraints," in *Proc. IEEE Conf. Communications and Network Security (IEEE CNS 2013)*. Washington, DC, USA: IEEE, October 2013, pp. 260–268.

[22] J. H. Jafarian, E. Al-Shaer, and Q. Duan, "OpenFlow random host mutation: Transparent moving target defense using software defined networking," in *Proc. 1st Workshop on Hot Topics in Software Defined Networks (HotSDN 2012)*. Helsinki, Finland: ACM, August 2012, pp. 127–132.

[23] G. F. Lyon, *Nmap Network Scanning: The Official Nmap Project Guide to Network Discovery and Security Scanning*. Insecure, 2009.

[24] D. Watson, M. Smart, G. R. Malan, and F. Jahanian, "Protocol scrubbing: Network security through transparent flow modification," *IEEE/ACM Transactions on Networking*, vol. 12, no. 2, pp. 261–273, April 2004.

[25] T. Carroll, M. Crouse, E. Fulp, and K. Berenhaut, "Analysis of network address shuffling as a moving target defense," in *2014 Intl. Conf. Communications (ICC)*, June 2014, pp. 701–706.

[26] J. H. H. Jafarian, E. Al-Shaer, and Q. Duan, "Spatio-temporal address mutation for proactive cyber agility against sophisticated attackers," in *Proc. First ACM Workshop on Moving Target Defense (MTD)*, New York, NY, USA, 2014, pp. 69–78.

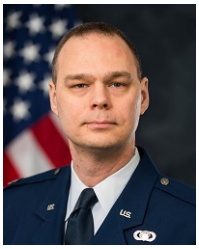
[27] K. Zaffarano, J. Taylor, and S. Hamilton, "A quantitative framework for moving target defense effectiveness evaluation," in *Proc. 2nd ACM Workshop on Moving Target Defense (MTD)*, New York, NY, USA, 2015, pp. 3–10.

[28] H. Maleki, S. Valizadeh, W. Koch, A. Bestavros, and M. van Dijk, "Markov modeling of moving target defense games," in *Proc. 2016 ACM Workshop on Moving Target Defense (MTD)*, New York, NY, USA, 2016, pp. 81–92.

[29] K. A. Farris and G. Cybenko, "Quantification of moving target cyber defenses," in *SPIE Defense+ Security*. International Society for Optics and Photonics, 2015, pp. 94 560L–94 560L.

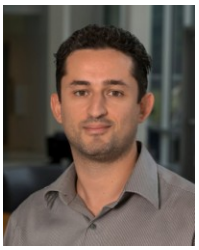
[30] M. Alia, M. Lacoste, R. He, and F. Eliassen, "Putting together QoS and security in autonomic pervasive systems," in *Proc. 6th ACM Workshop on QoS and Security for Wireless and Mobile Networks*. New York, NY, USA: ACM, 2010, pp. 19–28.

[31] F. Alomari and D. A. Menascé, "An autonomic framework for integrating security and quality of service support in databases," in *Software Security and Reliability (SERE), 2012 IEEE Sixth Intl. Conf.*, June 2012, pp. 51–60.



Warren Connell is a 2017 graduate of the PhD program in Information Technology at George Mason University. He is an active-duty Air Force officer with a variety of computer security-related assignments throughout his 21-year career. He is currently assigned as Operational Test Deputy, F-35 Joint Program Office, Crystal City, Virginia, and following his current tour, he is slated for assignment to a faculty position at the Air Force Institute of Technology, Dayton, Ohio. His research interests include moving target defense, anti-tamper techniques, and data aggregation.

Daniel A. Menascé is a University Professor of Computer Science at George Mason University where he was Senior Associate Dean of the Volgenau School of Engineering for seven years. He received his Ph.D. in Computer Science from UCLA, is a Fellow of the ACM and of the IEEE, a recipient of the 2017 Outstanding Faculty Award from the Commonwealth of Virginia, and the recipient of the 2001 lifetime A.A. Michelson Award from the Computer Measurement Group. He is the author of over 260 peer-reviewed papers and five books on analytic models of computer systems published by Prentice Hall and translated into Russian, Korean, and Portuguese. His research interests include autonomic computing, security performance tradeoffs, analytic modeling of computer systems, and software performance engineering.



Massimiliano Albanese is an Associate Professor in the Department of Information Sciences and Technology at George Mason University, where he serves as Associate Director of its Center for Secure Information Systems (CSIS) and Codirector of the Laboratory for IT Entrepreneurship (LITE). He received his Ph.D. degree in Computer Science and Engineering in 2005 from the University of Naples Federico II and was a Postdoctoral Researcher at the University of Maryland. His research interests include information and network security, cyber situational awareness, moving target defense, and adaptive cyber defense. Dr. Albanese holds a U.S. Patent and has co-authored a book and over 60 papers in peer-reviewed journals and conference proceedings. Dr. Albanese is a recipient of the 2014 Mason Emerging Researcher/Scholar/Creator Award, a most prestigious honor at Mason.