Stochastic Game-Based Analysis of the DNS Bandwidth Amplification Attack Using Probabilistic Model Checking

Tushar Deshpande*, Panagiotis Katsaros†, Scott A. Smolka*, and Scott D. Stoller*
* Department of Computer Science, Stony Brook University
E-mail: {tushard, stoller, sas}@cs.stonybrook.edu
† Department of Informatics, Aristotle University of Thessaloniki
Email: katsaros@csd.auth.gr

Abstract—**The Domain Name System (DNS) is an Internet-wide, hierarchical naming system used to translate domain names into numeric IP addresses. Any disruption of DNS service can have serious consequences. We present a formal game-theoretic analysis of a notable threat to DNS, namely the *bandwidth amplification attack* (BAA), and the countermeasures designed to defend against it. We model the DNS BAA as a two-player, turn-based, zero-sum *stochastic game* between an attacker and a defender. The attacker attempts to flood a victim DNS server with malicious traffic by choosing an appropriate number of *zombie* machines with which to attack. In response, the defender chooses among five BAA countermeasures, each of which seeks to increase the amount of legitimate traffic the victim server processes. To simplify the model and optimize the analysis, our model does not explicitly track the handling of each packet. Instead, our model is based on calculations of the rates at which the relevant kinds of events occur in each state. We use our game-based model of DNS BAA to generate optimal *attack strategies*, which vary the number of zombies, and optimal *defense strategies*, which aim to enhance the utility of the BAA countermeasures by combining them in advantageous ways. The goal of these strategies is to optimize the attacker's and defender's *payoffs*, which are defined using probabilistic reward-based properties, and are measured in terms of the attacker's ability to minimize the volume of legitimate traffic that is processed, and the defender's ability to maximize the volume of legitimate traffic that is processed.**

I. Introduction

DNS (Domain Name System) is a critical infrastructure component of the Internet that provides name-to-IP-address resolution; i.e., it translates domain names into numeric IP addresses. For scalability, DNS is implemented by a distributed hierarchical database and a query-response protocol. It is widely acknowledged that the initial DNS design did not take into account the threats that came along with the immense growth of the Internet. Numerous attacks, which can negatively impact DNS performance and robustness (availability), have prompted the development of sophisticated countermeasures.

For the most prevalent threats to DNS, there is no universally deployed countermeasure. A *Bandwidth Amplification Attack* (BAA) exploits a network of computers to flood a DNS server with many large DNS responses to presumed requests that have never been made. If the available bandwidth for legitimate DNS traffic is exhausted, the attack causes a Distributed Denial of Service (DDoS) incident. Countermeasures include packet filtering, random drops, and aggressive retries, but various hybrid solutions may be more effective under certain circumstances.

In [1], we modeled the DNS BAA and its three basic countermeasures—packet filtering, random drops, and aggressive retries—using Continuous-Time Markov Chains (CTMCs), and studied countermeasure effectiveness using a cost-benefit analysis. We defined benefit and cost metrics that reflected a countermeasure's positive and negative effects on a DNS server, and computed their values by model checking probabilistic reachability and reward properties over a bounded time period, before the system reaches the steady-state. This transient analysis reflected the need to evaluate countermeasures, in terms of their ability to mitigate the attack within as short a period of time as possible [2] (in steady-state, the effects of the BAA on bandwidth usage do not depend on the available bandwidth). We employed an automated model-repair process [3] to determine an optimal configuration of the countermeasure parameters required to achieve a desired attack success probability. These parameter settings were used in our cost-benefit analysis, which ranks countermeasures in terms of their net-benefit value.

In recently reported BAA incidents, the attackers often employ tricks to sidestep attack detection mechanisms [4], like diverse types of DNS requests or queries for varying DNS domains [5]. Our CTMC-based study of the BAA is not appropriate for attacks, in which the attack characteristics or the attack intensity are varied dynamically, and in which the defender's optimal strategy may require switching among multiple countermeasures.

Game-theoretic modeling of DNS attacks can automate the synthesis of defense strategies that combine multiple countermeasures, in order to provide optimal protection. The need to find the optimal configuration of countermeasure parameters is eliminated, because those configurations are computed as part of the synthesized strategies. Motivated by these observations, in this paper we model the DNS BAA as a two-player game with perfect information that is played between an attacker and a defender. In reality, the attacker cannot have perfect information for the defender's state. However, this overly pessimistic assumption from the defender's point of view eventually results in a more conservative solution for him.

Our model is formulated in PRISM-games, an extension of the PRISM model checker [6] that allows one to model two-player, turn-based, zero-sum stochastic games and specify interesting probabilistic and reward-based properties. PRISM-games generates the optimal strategies for a player, guaranteeing that the player can optimize a property irrespective of any strategy chosen by the adversary.

In our DNS BAA game, the attacker exploits multiple DNS requests, for amplifying the occupied bandwidth and chooses the number of compromised machines (*zombies*) to launch a BAA. The defender selects the best possible combination of countermeasures. For the attacker, we specify two goals or *payoffs*: one seeks to maximize the difference between bogus packets received and legitimate packets received, and the other seeks to maximize the difference between legitimate packets dropped per zombie and legitimate packets received per zombie. The defender's strategy minimizes the attacker's payoff. For each of these payoffs, we record the optimal attack and defense strategies generated by PRISM-games.

Our model is designed to capture the attacker's and defender's choices and payoffs at the highest suitable level of abstraction, in order to simplify the representation and optimize the analysis. The most straightforward model would explicitly track the handling of each packet as it traverses the network, and would require modeling the capacity and the contents of each buffer and network queue. Such a representation would have a very large state space, and the analysis would be feasible only for very small parameter values. Our model abstracts from these details and is based on calculations of the rates at which the relevant kinds of events occur in each state. Realistic data is used in the rate calculations.

An optimal attack strategy may repeatedly update the number of zombies used or even pause the attack, while an optimal defense strategy may involve switching among basic and composite countermeasures along with updating their parameter settings as the attack progresses. Strategies generated in this manner constitute new optimal strategies for launching and defending against DNS BAA attacks.

The rest of this paper is organized as follows. Section II provides background on DNS and on the BAA attack and its countermeasures. Section III introduces stochastic game-based modeling with PRISM-games. Section IV describes our stochastic game-based model of the DNS BAA. Section V presents our experimental results. Section VI considers related work. Section VII offers our concluding remarks.

II. Domain Name System and Bandwidth Amplification Attack

DNS (Domain Name System) is a hierarchical naming system for the Internet, based on an underlying client-server architecture, which is also hierarchical. The primary function of a DNS server is to perform *URL-resolution*: the process of translating a url or domain name, such as `mail.google.com`, into an IP address, such as 209.85.132.83. DNS is implemented using hierarchically organized domain name servers. At the top of this hierarchy are the DNS servers for the root domain, which know IP addresses of DNS servers for top-level domains such as `com` and `edu`. Top level DNS servers know IP addresses of DNS servers for domains such as `google.com` that belong to individual organizations. Such DNS servers, in turn, know IP addresses of name servers of their subdomains, e.g., `mail.google.com`. Name servers of domains belonging to individual organizations and their subdomains also know the IP addresses of any other networked machines, e.g., web servers like `www.google.com`, that are within those domains or subdomains.

When a DNS server receives a url-resolution query from a client, it first checks to see if it can answer the query *authoritatively* based on a locally maintained database of *resource records* mapping domain names to IP addresses. If the queried name matches a corresponding resource record in its local database, the server gives an *authoritative answer* (AA), using the local resource record to resolve the queried name. If no local information exists for the queried name, the server then checks to see if it can resolve the name using information cached locally from previous queries. If a match is found, the server answers with the appropriate cache entry and the query is completed [7], [8].

If the queried name does not find a matched answer at its preferred server—either from its cache or local database—the query process can continue, using *recursion* to fully resolve the name. Such *recursive queries* involve assistance from other DNS servers to help resolve them. The response to the last recursive query is the AA response (if the url is valid). The DNS servers that support recursive DNS queries are also called *DNS resolvers*. The root and other top-level domains, on the other hand, are configured to be non-recursive. A non-recursive DNS server provides a *referral response* (RR) to a DNS query: a pointer (referral) to another DNS server that presumably has authority for a lower portion of the DNS namespace and can assist in resolving the query.

Domain name queries can also be *iterative*, where a DNS server returns the best possible answer, which can be either an AA response, a RR response, or an error response [9]. If the response is a RR response, then the DNS client itself queries the DNS server pointed by the RR response. On the other hand, a recursive query always returns an AA response or an error response, because a DNS resolver handles recursive queries on behalf of its clients. A DNS resolver can provide faster responses by caching, so DNS servers of Internet Service Providers are usually configured to support recursive queries.

A. DNS Bandwidth Amplification Attack

The DNS BAA is a DDoS attack in which a network of computers floods a DNS resolver with large responses to requests that the resolver never made. A typical DNS response size is 512 bytes. During a BAA, however, the victim DNS resolver can receive DNS responses that are as large as 4,000 bytes. These unwanted responses consume both the bandwidth

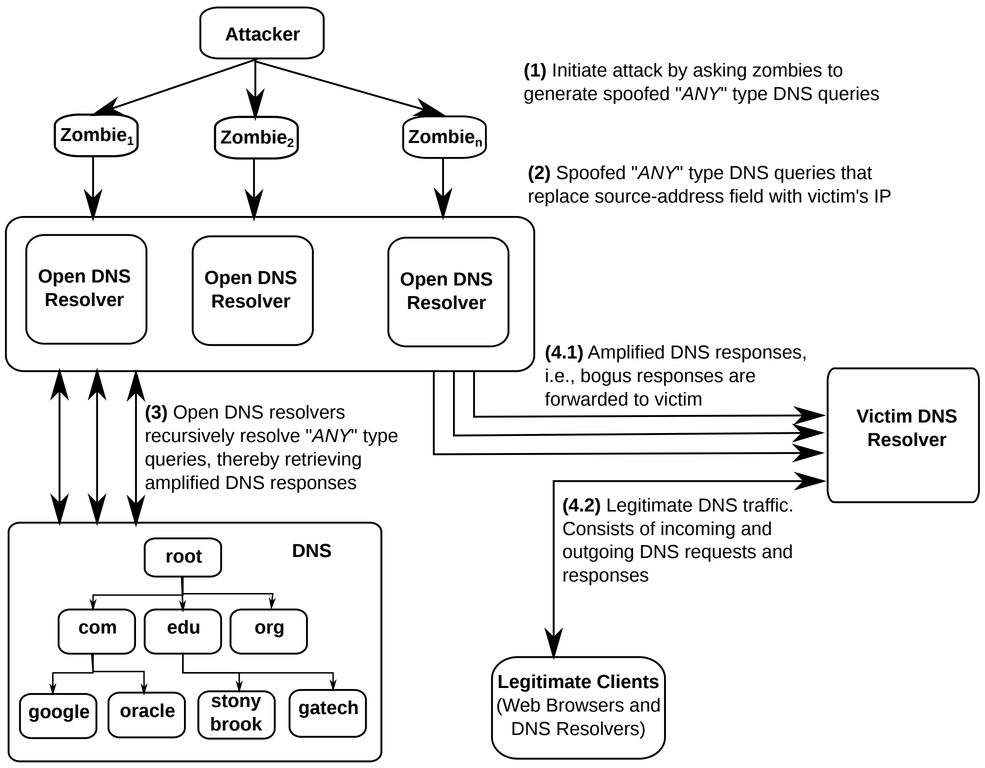

Attacker
(1) Initiate attack by asking zombies to generate spoofed "ANY" type DNS queries
Zombie1
Zombie2
Zombien
(2) Spoofed "ANY" type DNS queries that replace source-address field with victim's IP
Open DNS Resolver
Open DNS Resolver
Open DNS Resolver
(4.1) Amplified DNS responses, i.e., bogus responses are forwarded to victim
Victim DNS Resolver
(3) Open DNS resolvers recursively resolve "ANY" type queries, thereby retrieving amplified DNS responses
(4.2) Legitimate DNS traffic. Consists of incoming and outgoing DNS requests and responses
root
DNS
com
edu
org
google
oracle
stony brook
gatech
Legitimate Clients (Web Browsers and DNS Resolvers)

Fig. 1. Schematic diagram of DNS BAA

and the computational power of the victim DNS server. BAAs are a major cause of DNS disruption; a number of incidents involving BAAs have been reported [10].

A typical DNS query retrieves only the IP address for a single URL. However, an *ANY*-type query asks a DNS server to return various information about the domain. The size of a response to an *ANY*-type query can be 50 times larger than the size of a response to a typical DNS query. In this case, the *amplification factor* (AF), i.e., the ratio of the response size to the request size, is 50 [11]. For DNS, the AF can be as high as 73 [12], [10], while the AF for DNSSec, the security extension of DNS, can be as high as 271.2 [13]. The fact that a small DNS request can generate a substantially larger response makes the BAA possible.

We now describe how an attacker can launch a DNS BAA on a DNS resolver, which is called the *victim server*. The attacker prepares for the BAA by acquiring control of compromised hosts, called zombies, to be used as attack sources, and by acquiring a list of open DNS resolvers. A DNS resolver is *open* if it is able to provide recursive name-resolution service for clients outside of its administrative domain [14].

The steps in a BAA are illustrated in Figure 1. The attacker commands the zombies to send to the previously found open DNS resolvers a number of requests that generate amplified responses (Figure 1, Step 1). For example, the zombies may send *ANY*-type requests to the open DNS resolvers (Figure 1, Step 2). Moreover, these queries are *spoofed*: the source-address fields of these queries have been replaced with the victim's IP address. The open DNS resolvers resolve the spoofed queries (Figure 1, Step 3) and direct the large number of amplified responses that they receive to the victim server (Figure 1, Step 4.1), thereby exhausting the victim's available bandwidth. After some time, the attacker may change the parameters of the attack, by changing the number of zombies used, and by deciding whether to exploit the same set of spoofed DNS requests (a *repeating query attack*) or alternatively prepare new ones (a *varying query attack*) at the cost of some delay, if the attack is being mitigated [5], [4].

B. DNS BAA Countermeasures

Three basic countermeasures are suggested to prevent DNS BAA attacks.

- **Filtering (FTR):** Filtering seeks to identify and block attack traffic. It offers relatively high accuracy with a false-positive rate as low as 10% [15], [16], [17], [18], [19]. The computational demands of FTR depend on the filtering mechanism and the attack strength.
- **Random Drops (RND):** RND regulates incoming traffic by randomly dropping DNS packets [20], [21]. During a BAA, the traffic arriving at the victim mainly comprises bogus packets, and a randomly dropped packet is, therefore, likely to be bogus. RND has negligible computational demands.
- **Aggressive Retries (AGR):** AGR encourages legitimate clients to generate traffic at a higher rate. In AGR, increasing the number of retries by one doubles the amount of legitimate traffic generated during each retry [22], [23]. The downside is the increased server workload and bandwidth consumption.

It is possible to combine the basic BAA countermeasures to obtain better protection.

- **Random Drops with Aggressive Retries (RDR):** RDR is obtained by combining RND with AGR.
- **Aggressive Retries with Filtering (AGF):** AGF is obtained by combining AGR with FTR.

Both RDR and AGF try to filter out attack traffic while explicitly increasing the proportion of legitimate traffic.

III. Stochastic Game-Based Model Checking

A. Stochastic Games and the rPATL Logic

A *stochastic multi-player game* is played by a finite number of players on a finite state space, and, in each state, each player chooses one of finitely many actions; the resulting profile of actions determines a reward for each player and a probability distribution on successor states. In a *turn-based* game, at every state, only one player can choose from the set of actions available in that state. For a finite set X, let $\mathcal{D}(X)$ denote the set of discrete probability distributions over X.

Definition 1 (SMG [24]): A *turn-based stochastic multi-player game* (SMG) is a tuple $\mathcal{G} = \langle \Pi, S, A, (S_i)_{i\in\Pi}, \Delta, AP, \chi\rangle$, where Π is a finite set of players; S is a finite, non-empty set of states; A is a finite, non-empty set of actions; $(S_i)_{P_i\in\Pi}$ is a partition of $S; \Delta : S \times A \to \mathcal{D}(S)$ is a partial transition function; AP is a finite set of atomic propositions; and $\chi : S \to 2^{AP}$ is a labeling function.

In each state $s \in S$ of the SMG $\mathcal{G}$, the set of *available* actions is $A(s) = \{\alpha \in A \,|\, \Delta(s, \alpha) \neq \perp\}$. We assume that $A(s) \neq \emptyset$ for all s. The choice of action to take in s is under the control of exactly one player, namely the player $i \in \Pi$ for which $s \in S_i$. Once action $\alpha \in A(s)$ is selected, the successor state is chosen according to the probability distribution $\Delta(s, \alpha)$. A *path* of $\mathcal{G}$ is a possibly infinite sequence $\lambda = s_0\alpha_0 s_1\alpha_1...$ such that $\alpha_j \in A(s_j)$ and $\Delta(s_j, \alpha_j)(s_{j+1}) > 0$, for all j.

A *strategy* for player $i \in \Pi$ in $\mathcal{G}$ is a function $\sigma_i : (SA)^*S_i \to \mathcal{D}(A)$, which, for each path $\lambda \cdot s$, with $s \in S_i$, assigns a probability distribution $\sigma_i(\lambda{\cdot}s)$ over $A(s)$. A *strategy profile* $\sigma = \sigma_1, \ldots, \sigma_{|\Pi|}$ comprises a strategy for all players in the game. Under a strategy profile σ, the behavior of $\mathcal{G}$ is fully probabilistic and a probability measure can be defined over the set of all paths $\Omega_{\mathcal{G},s}$ starting in state s [24].

SMGs have been augmented with *reward structures* $r : S \to \mathbb{Q}_{\geq 0}$, mapping each state s to a non-negative rational value (reward) that the player controlling s receives as a *payoff*. Payoffs represent the desirability of a game's outcome as perceived by the players in the game [25]. Transition/action rewards are also possible in an SMG. The *total payoff* over a path is the sum of the payoffs over each state in the path [26].

rPATL (Probabilistic Alternating-time Temporal Logic with Rewards) [24] is a CTL-style branching-time temporal logic that can be used to express quantitative properties of SMGs with rewards. rPATL combines the coalition operator $\langle\langle C\rangle\rangle$ of ATL (Alternating-time Temporal Logic) [27], the probabilistic operator $\mathcal{P}_{\bowtie q}$ of PCTL (Probabilistic Computation Tree Logic), where $\bowtie \in \{<, \leq, \geq, >\}$, $q \in \mathbb{Q} \cap [0, 1]$, and a variant of the reward operator $\mathcal{R}^r_{\bowtie x}$ proposed in [28], where r is a reward structure, $x \in \mathbb{Q}_{\geq 0}$, in order to reason about several types of expected reward measures. The semantics of these operators is defined in [24] via a reduction to a two-player game called a coalition game.

Definition 2 (Coalition Game [24]): For a coalition of players $C \subseteq \Pi$ of SMG $\mathcal{G}$, we define the *coalition game* of $\mathcal{G}$ induced by C as the stochastic two-player game $\mathcal{G}_C = \langle\{1, 2\}, S, A, (S'_1, S'_2), \Delta, AP, \chi\rangle$, where $S'_1 = \cup_{i\in C} S_i$ and $S'_2 = \cup_{i\in\Pi\setminus C} S_i$.

rPATL formulas are interpreted over the states of a game $\mathcal{G}$, are used to check for the existence of a strategy that satisfies a given probability/reward bound, or a strategy optimizing some objective. For example, the formula $\langle\langle\{1, 2\}\rangle\rangle\mathcal{P}_{\geq 0.5}[\psi]$ means that players 1 and 2 have a strategy to ensure that the probability of path formula ψ being satisfied is at least 0.5, regardless of the strategies of other players. rPATL allows path formulas with the standard temporal operators X ("next"), $U^{\leq k}$ ("bounded until"), and U ("until"). Also, the F ("eventually") operator is derived from the U operator in the usual way: $F\phi = true\, U\phi$, where ϕ is a state rPATL formula.

rPATL reward formulas are of the form $\langle\langle C\rangle\rangle\mathcal{R}^r_{\bowtie x}[F^\star\phi]$, and are annotated with a reward structure r and a type $\star \in \{0, \infty, c\}$. It states that coalition C has a strategy to ensure that the expected sum of rewards accumulated along a path until a state satisfying ϕ is reached satisfies $\bowtie x$. The type $\star$ comes into play when the target set of states is *not* reached, assigning zero reward ($\star = 0$), infinite reward ($\star = \infty$), or the reward accumulated the whole path. The value chosen for the type parameter depends on the application. For example, if the goal is to maximize the expected reward for reaching a target state, then a type of $\star = 0$ would be appropriate, thereby disincentivizing paths that fall short of this goal. Similarly, a type of $\star = \infty$ would be appropriate if the goal was to minimize the expected reward for reaching a target state.

B. Probabilistic Model Checking in PRISM and PRISM-games

Probabilistic model checking is an automated formal verification technique for modeling and analyzing systems or processes with probabilistic behavior. Model-checking tools like PRISM [6] combine graph-theoretic algorithms for reachability analysis with iterative numerical solvers. They can evaluate properties of the form $\mathcal{P}_{\bowtie q}(\psi)$ or $\mathcal{P}_{=?}(\psi)$, which compute the probability that a path satisfies ψ. Path formula ψ is interpreted over the paths of the probabilistic model, which could be a Discrete Time Markov Chain (DTMC), a Continuous Time Markov Chain (CTMC), or a Markov Decision Process (MDP).

A PRISM model is a parallel composition of *modules*, whose state is determined by a set of variables. A module consists of a collection of *guarded commands*. In an MDP or DTMC, each such command consists of a guard g (i.e., a state predicate) and one or more updates for the module's variables, where each update u_i is labeled by the probability λ_i with which u_i occurs from a state satisfying the guard g. In a model of a CTMC, commands have the same form, except that λ_i is interpreted as the transition rate for u_i.

When a module has a command whose guard is satisfied in the current model state, it can update its variables probabilistically, according to the specified updates. In a DTMC or a CTMC model, only one command can be enabled in a state. In MDPs, multiple commands (actions) may be enabled simultaneously, thus representing a nondeterministic choice between multiple discrete probability distributions over successor states.

The PCTL semantics for the property $\mathcal{P}_{\bowtie q}(\psi)$ over MDPs is that, for all strategies, the probability that ψ is true for a path satisfies $\bowtie q$, where a strategy (or adversary) for an MDP is a function mapping that resolves nondeterminism based on execution history. Model checking such a property in PRISM reduces to the computation over all strategies of either the minimum or maximum probability of ψ holding true. Accordingly, for MDPs, two forms of quantitative properties are supported by PRISM, namely $\mathcal{P}_{min=?}(\psi)$ and $\mathcal{P}_{max=?}(\psi)$.

PRISM also allows the definition of reward structures. Consider a DTMC $D = (S, \overline{S}, P, L)$, where S is a finite set of states, $\overline{S} \in S$ is the initial state, $P : S \times S \rightarrow [0, 1]$ is the transition probability matrix such that $\sum_{s' \in S} P(s, s') = 1$ for all $s \in S$, and $L : S \rightarrow 2^{AP}$ is a labeling function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that hold true in the state. Reward structures are defined as pairs of: (i) state rewards ($\rho : S \rightarrow \mathbb{R}_{\geq 0}$) and (ii) transition rewards ($\iota : S \times S \rightarrow \mathbb{R}_{\geq 0}$).

Reward-based properties have the form $\mathcal{R}\{"\mathit{rewardId}"\} \bowtie x\,[\psi]$ or $\mathcal{R}\{"\mathit{rewardId}"\} =?\,[\psi]$, where the $\mathcal{R}$ operator signifies a reward-based property, *rewardId* identifies the reward structure to be used, and ψ is a path formula. Such a property returns the instantaneous expected reward or the cumulative expected reward for the specified reward structure, until the specified path formula is satisfied.

Consider the expected accumulated values of a reward. $\mathit{ExpReach}(s, T)$, the expected reward accumulated starting from state s until the target states $T \subseteq S$ are reached, is defined using following linear equation system [29]:

$$\mathit{ExpReach}(s,T) = \begin{cases} 0 & \text{if } s \in T \\ \infty & \text{if } \mathit{ProbReach}(s,T) \leq 1 \\ \rho(s) + \sum_{s' \in S} P(s,s') \cdot (\iota(s,s') & \text{otherwise} \\ \qquad\qquad + \mathit{ExpReach}(s',T)) & \end{cases} \tag{1}$$

where $\mathit{ProbReach}(s, T)$ is the probability that a path starting in state s would reach a state in the target set $T \subseteq S$. For MDPs, rewards can be assigned to actions instead of transitions and $\mathcal{R}\{"\mathit{rewardId}"\} =?\,[\psi]$ is replaced by $\mathcal{R}\{"\mathit{rewardId}"\}max =?\,[\psi]$ or $\mathcal{R}\{"\mathit{rewardId}"\}min =?\,[\psi]$.

PRISM-games extends PRISM's MDP model checking capabilities to support coalition games, in order to analyze systems with both *probabilistic* and *competitive* behaviors [24]. PRISM-games supports analysis of turn-based, zero-sum coalition games. Such games are played between only two players (or coalitions of players), and at a given state of the game only one of them is allowed to make a move. Turn-based games can be used to model systems where multiple components execute under the control of a central scheduler and each component chooses an action. Zero-sum means that the gain of one player is exactly balanced by the loss of the other player [30].

In PRISM-games [31], SMGs are described in a modeling language similar to the one for PRISM. A model is composed of *modules*, whose behavior is specified by a set of *guarded commands*, each of which contains an (optional) action label, a guard, and a probabilistic update for the module's variables. For action-labeled commands, multiple modules execute updates synchronously, if all their guards are satisfied. Each probabilistic transition in the model is thus associated with either an action label or a single module.

A model also defines *players*, each of which is assigned modules, as well as a disjoint subset of the model's synchronizing action labels. A module can be part of multiple players as long as each enclosed command is assigned to exactly one player. Thus, each probabilistic transition is assigned to one player. In the current implementation of PRISM-games, all possible probabilistic transitions on a state must belong to the same player; PRISM-games detects and disallows concurrent actions. To satisfy the coalition game constraint, all players must be divided into two groups. These groups of players act as adversaries of each other, thus taking into account their competitive behavior.

PRISM-games supports an extension of rPATL that allows one to write coalition-based properties that identify a strategy that maximizes or minimizes either the expected probability of a path or the expected value of the accumulated reward while reaching a set of states.

C. PRISM-games Optimal Strategies and Nash equilibrium

Consider a reward-based rPATL property in PRISM-games:

$$<< \mathtt{P_1, P_2, \ldots, P_k} >> \mathtt{Rmax/min}\{\text{“}\mathtt{rewardDef}\text{”}\} =?\ [\mathtt{F}\ \phi]$$

This property asks PRISM-games to generate an optimal strategy for the coalition C_1 containing players $\mathtt{P_1}$, $\mathtt{P_2}$, …, $\mathtt{P_k}$, where $C_1 \subset \Pi$. Such an optimal strategy either maximizes or minimizes the expected accumulated value of reward “`rewardDef`” until some state from the target set of states that satisfy state formula ϕ is eventually reached. Since the game is also a zero-sum game, PRISM-games simultaneously generates the optimal strategy for the adversarial coalition C_2 ($C_1 \cup C_2 = \Pi$), which seeks to reduce the reward obtained by coalition C_1. PRISM-games returns both the generated optimal strategy and the expected accumulated reward. In case the target set of states is not reached, the accumulated reward value is set to infinity.

Typically, the goal of a game-theoretic analysis is to determine if there exists a *Nash equilibrium* for a game. A Nash equilibrium represents a complete set of strategies (i.e. for all players) such that no player can hope to better its payoff (reward) by unilaterally switching to another strategy [32].

In game-based models of security attacks, Nash equilibria represent optimal strategies for the defender and attacker, by virtue of which the defender can minimize the impact of the attack and the attacker can cause maximum damage. Therefore, it is interesting to study the relationship between optimal strategies generated by PRISM-games and Nash equilibrium.

Let us assume a two-player zero-sum game with players $\mathtt{P}_1$ and $\mathtt{P}_2$. Consider the property $<< \mathtt{P}_1 >> \mathtt{Rmax}\{\text{“}\mathtt{reward}_1\text{”}\} =? [\mathtt{F}\ \phi_1]$. This causes PRISM-games to generate a strategy for $\mathtt{P}_1$, that maximizes the expected value of reward $\mathtt{reward}_1$ accumulated while reaching a state satisfying ϕ_1. This means that $\mathtt{P}_1$ has identified a strategy that would allow it to maximize the expected accumulated value of $\mathtt{reward}_1$ in face of maximum opposition from $\mathtt{P}_2$. $\mathtt{P}_1$ therefore has no incentive to switch to any other strategy, since it is bound to reduce the expected accumulated value of $\mathtt{reward}_1$.

Similarly, $\mathtt{P}_2$ has identified a strategy that would minimize $\mathtt{P}_1$'s payoff. Since the game is a zero-sum game, any reduction in $\mathtt{P}_1$'s payoff or reward is a gain for $\mathtt{P}_2$. So, $\mathtt{P}_2$ has also maximized its own payoff and it has no incentive to switch to any other strategy. Therefore, we can infer that an optimal strategy generated by PRISM-games is indeed a Nash equilibrium.

IV. Two-player, Turn-based Stochastic Game for DNS BAA

We model the DNS BAA as a two-player, turn-based stochastic game. This section describes our model, which is illustrated by a schematic diagram in Figure 2.

A. Modules and players

The stochastic game for the DNS BAA developed using the PRISM-games model checker consists of modules corresponding to the attacker, defender, and victim server. These three modules are described next.

Attacker (AT). The attacker chooses the number of zombies to launch the BAA. The attacker can stop attacking by setting the number of zombies to zero. To allow modeling of constraints on the temporal behavior of the attacker, we introduce two parameters: MAX_SUCC_ATTACKS, the maximum duration of a continuous attack (i.e., the maximum number of consecutive steps with $\mathtt{zombies} > 0$), and ATTACKER_LATENCY, the maximum number of steps before the attack can resume after MAX_SUCC_ATTACKS steps of continuous attack. For example, an attacker might have a policy of stopping and restarting its attack periodically, in order to help evade detection and tracing [4], using tactics such as identifying and switching to a different set of zombies (we do not model this explicitly, since only the number of zombies being used is modeled), or computing and switching to a different set of DNS queries [5]. The parameter ATTACKER_LATENCY models the time needed for such tactics. Scenarios in which such temporal constraints are not present can also be analyzed, by setting MAX_SUCC_ATTACKS to the length of the run (i.e., maxTime, introduced below).

As seen in Figure 1, the victim receives amplified bogus DNS responses from the open DNS resolvers that resolve the spoofed queries generated by zombies. Because open DNS resolvers simply reflect bogus DNS responses to the victim, while modeling the attack it is reasonable to assume that the bogus traffic originates directly from the zombies. As such, we omit open DNS resolvers from our model. Moreover, since each zombie sends bogus DNS responses at a fixed rate, we assume that bogus traffic originates from the attacker (Step 1.1 in Figure 2). The rate at which the bogus traffic arrives can be computed by multiplying the rate at which each zombie sends bogus DNS responses to the victim by the number of zombies.

Defender (DF). The defender decides whether to disable countermeasures (Nofix) or apply the FTR, AGR, RND, RDR, or AGF countermeasure (Step 2 in Figure 2). After selecting a countermeasure, the defender chooses the countermeasure parameters.

Victim Server (VS). The victim server is a DNS resolver. The VS has a finite network bandwidth, which is shared by legitimate DNS traffic (Step 1.2 in Figure 2) and BAA traffic (Step 1.1 in Figure 2). VS's bandwidth is BW packets per second. If packets arrive at a rate higher than this, the excess packets are dropped (Step 3.2 in Figure 2).

Since PRISM-games supports only two-player games, we divide these three players into two groups or *coalitions*, one containing AT, and the other containing DF and VS.

Model constants and parameters. Since VS is a DNS resolver, it handles both legitimate DNS requests and legitimate DNS responses. To model this as simply as possible, we introduce an abstraction called *legitimate DNS packet*. The size of a legitimate DNS packet is the weighted average of the sizes of typical legitimate DNS requests and typical legitimate DNS responses. The weights are proportional to the frequencies with which requests and responses appear in legitimate traffic. One DNS request generates one response; so, requests and responses typically appear with the same frequency in legitimate traffic. Thus, the size of a legitimate DNS packet is the average of the size of a legitimate DNS request (60 bytes) and the size of a typical legitimate DNS response (512 bytes) [10]. The legitimate DNS packet size is, therefore, $(60 + 512)/2 = 286$ bytes.

As discussed in Section II-A, each bogus DNS response in a BAA is AF times larger than a legitimate DNS request, i.e., AF = $(\text{bogus DNS response size})/(\text{legitimate DNS request size})$. Since our model uses the abstraction of legitimate DNS packets, we need to adapt the definition of AF to use the legitimate DNS packet size instead of the legitimate request size. The modified definition for AF is AF = $(\text{bogus DNS response size})/(\text{legitimate DNS packet size})$. The maximum size of a bogus DNS response is 4380 bytes [10], so $\mathtt{AF} = (4380\ \text{bytes})/(286\ \text{bytes}) = 15.31$.

Since a bogus DNS response is AF times larger than a legitimate DNS packet, one bogus DNS response is equivalent, in terms of bandwidth usage, to AF legitimate DNS packets. The number of bogus DNS responses and the number of legitimate DNS packets, and the bandwidth limit of VS's network can, thus, be measured using a single unit called

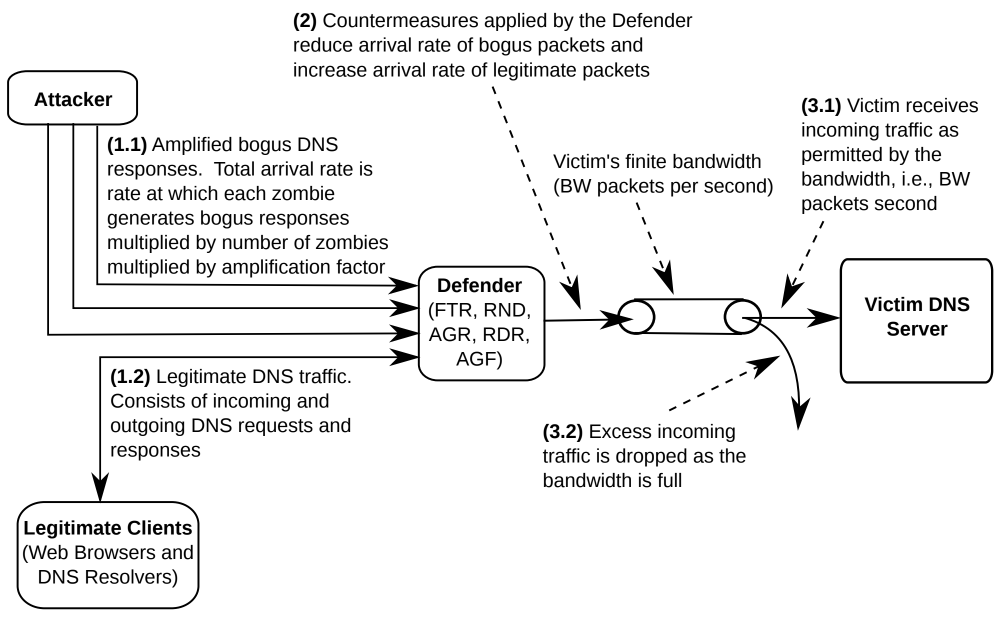

(2) Countermeasures applied by the Defender reduce arrival rate of bogus packets and increase arrival rate of legitimate packets
Attacker
(1.1) Amplified bogus DNS responses. Total arrival rate is rate at which each zombie generates bogus responses multiplied by number of zombies multiplied by amplification factor
Victim's finite bandwidth (BW packets per second)
(3.1) Victim receives incoming traffic as permitted by the bandwidth, i.e., BW packets second
Defender (FTR, RND, AGR, RDR, AGF)
Victim DNS Server
(1.2) Legitimate DNS traffic. Consists of incoming and outgoing DNS requests and responses
(3.2) Excess incoming traffic is dropped as the bandwidth is full
Legitimate Clients (Web Browsers and DNS Resolvers)

Fig. 2. Modules and players in DNS BAA stochastic game

packets, provided we multiply the number of bogus DNS responses by `AF`.

The rate at which the legitimate DNS packets arrive at and flow out of VS is denoted R_l. As the consumption of VS's bandwidth mainly depends on the attack strength, we set R_l to a moderate value of 100 packets per second. Parameter `zombies` represents the number of zombies used in the BAA.

The rate at which each zombie sends bogus DNS packets to VS, denoted `bogus_rate`, equals `AF` times the number of bogus DNS responses sent per second by each zombie. Since the attack strength is controlled mainly by the number of zombies, we assume that each zombie sends a moderate number of 10 bogus DNS responses per second; so, `bogus_rate` = $10 \cdot$ `AF` packets per second. The net arrival rate for the bogus DNS packets, denoted R_b, is $R_b = \texttt{bogus\_rate} \cdot \texttt{zombies}$.

A typical DNS server has approximately 1 Mbps of dedicated bandwidth [33]. Since the legitimate DNS packet size is 286 bytes, 1 Mbps bandwidth can transmit $\frac{1\text{Mb}}{286} = 458$ packets per second. So, we set `BW`, the bandwidth of VS's network, to 458 packets per second in our model. Finally, `maxTime` represents the maximum time for which the model is allowed to execute. Note that a DNS server's bandwidth typically saturates before its CPU saturates, so it is unnecessary to model the rate at which VS can process DNS requests.

Countermeasure parameters. Next we describe the parameters of each countermeasure.

FTR has two parameters: *detection fraction* (`df`) and *false-positive fraction* (`fpf`). Parameter `df` is the fraction of attack traffic identified and filtered, while `fpf` is the fraction of legitimate traffic incorrectly identified as bogus. Studies of different filtering algorithms show that on average FTR has a `df` of 0.9 and `fpf` of 0.1. FTR decreases the legitimate packet rate to $R_l \cdot (1-\texttt{fpf})$ and the bogus packet rate to $R_b \cdot (1-\texttt{df})$.

RND has parameter *random-drop fraction* (`rdf`), the fraction of incoming legitimate and bogus packets randomly dropped. RND reduces the legitimate packet rate to $R_l \cdot (1-\texttt{rdf})$, and the bogus packet rate to $R_b \cdot (1-\texttt{rdf})$.

AGR has parameter `retries`, which controls the number of times the legitimate DNS packets are resent to increase the share of legitimate traffic. AGR increases the legitimate packet rate to $R_l \cdot 2^{\texttt{retries}}$.

RDR is the combination of RND and AGR, so RDR has parameters `rdf` and `retries`. It increases the legitimate packet rate to $R_l \cdot (1-\texttt{rdf}) \cdot 2^{\texttt{retries}}$, and decreases the bogus packet rate to $R_b \cdot (1-\texttt{rdf})$.

AGF is the combination of AGR and FTR, so it has parameters `retries`, `df`, and `fpf`. It increases the legitimate packet rate to $R_l \cdot (1-\texttt{fpf}) \cdot 2^{\texttt{retries}}$ and decreases the bogus packet rate to $R_b \cdot (1-\texttt{df})$.

B. Sequence of moves

PRISM-games supports turn-based games, so we need to ensure that in any state, only one player can make a move. We achieve this by introducing a scheduling variable named `turn`, whose value indicates whether AT, DF, or VS is enabled. This leads to the following sequence of moves:

1) Initially, AT is enabled (`turn` = AT), and `time`=0.
2) AT chooses the value of `zombies`.
3) AT enables DF, setting `turn` = DF.
4) DF chooses one of the five countermeasures or decides to disable the countermeasures.
5) DF selects values for the parameters of the selected countermeasure (if any).
6) DF enables VS, setting `turn` = VS.
7) VS receives legitimate and bogus packets with probabilities that reflect the selected countermeasure, the arrival

rates of the legitimate and bogus DNS packets, and VS's bandwidth.

8) VS enables AT, setting `turn` = AT, and increments time by 1.
9) If time is less than `maxTime`, go to step 2.

C. DNS BAA SMG

Figure 3 shows the SMG for the DNS BAA. In state AT : s_{init}, AT chooses the value of `zombies`. In state DF : s_2, DF decides which, if any, countermeasure to turn on. If DF turns on a countermeasure, DF next selects values for the countermeasure parameters. For example, in Figure 4, we see that the DF sets `df` to 0.85, 0.9, or 0.95. The state DF : $s^{\texttt{df}=0.9}_{cm=FTR}$ indicates that DF set `df` to 0.9. Next, DF sets `fpf` to 0.05, 0.1, or 0.15. VS's behavior is fully probabilistic, as it receives legitimate and bogus packets with probabilities that reflect the selected countermeasure, the arrival rates of legitimate and bogus DNS packets, and VS's bandwidth. In state VS : s^{cm}_4, the VS executes transition *Receive Packet* and with probability $p^{cm}_{b\_receive}$, a bogus packet is received and the next state is VS : $s^{cm}_{b\_received}$; with probability $p^{cm}_{l\_receive}$, a legitimate packet is received and the next state is VS : $s^{cm}_{l\_received}$; with probability $p^{cm}_{b\_drop}$, a bogus packet is dropped and the next state is VS : $s^{cm}_{b\_dropped}$; and with probability $p^{cm}_{l\_drop}$, a legitimate packet is dropped and the next state is VS : $s^{cm}_{l\_dropped}$. Note that $\left(p^{cm}_{b\_receive} + p^{cm}_{l\_receive} + p^{cm}_{b\_drop} + p^{cm}_{l\_drop}\right) = 1$. We now derive expressions for these four probabilities.

As shown in Figure 2, all traffic to VS passes first through DF and then through VS's network, so there are two places where a packet can be dropped: (1) at DF, if DF uses countermeasure FTR, RND, RDR, or AGF, and (2) at VS (i.e., VS's network), if VS's bandwidth is full.

First, we compute the rates $R^{cm}_{b\_in}$ and $R^{cm}_{l\_in}$ at which bogus and legitimate packets, respectively, pass through DF, and the rates $R^{cm}_{b\_out}$ and $R^{cm}_{l\_out}$ at which bogus and legitimate packets, respectively, are dropped at DF.

$$R^{cm}_{b\_in} = \begin{cases} R_b & \text{if } cm \in \{\text{Nofix}, \text{AGR}\} \\ R_b \cdot (1 - \texttt{df}) & \text{if } cm \in \{\text{FTR}, \text{AGF}\} \\ R_b \cdot (1 - \texttt{rdf}) & \text{if } cm \in \{\text{RND}, \text{RDR}\} \end{cases} \tag{2}$$

$$R^{cm}_{l\_in} = \begin{cases} R_l & \text{if } cm = \text{Nofix} \\ R_l \cdot (1 - \texttt{fpf}) & \text{if } cm = \text{FTR} \\ R_l \cdot (1 - \texttt{rdf}) & \text{if } cm = \text{RND} \\ R_l \cdot 2^{\texttt{retries}} & \text{if } cm = \text{AGR} \\ R_l \cdot 2^{\texttt{retries}} \cdot (1 - \texttt{rdf}) & \text{if } cm = \text{RDR} \\ R_l \cdot 2^{\texttt{retries}} \cdot (1 - \texttt{fpf}) & \text{if } cm = \text{AGF} \end{cases} \tag{3}$$

$$R^{cm}_{b\_out} = \begin{cases} 0 & \text{if } cm \in \{\text{Nofix}, \text{AGR}\} \\ R_b \cdot \texttt{df} & \text{if } cm \in \{\text{FTR}, \text{AGF}\} \\ R_b \cdot \texttt{rdf} & \text{if } cm \in \{\text{RND}, \text{RDR}\} \end{cases} \tag{4}$$

$$R^{cm}_{l\_out} = \begin{cases} 0 & \text{if } cm \in \{\text{Nofix}, \text{AGR}\} \\ R_l \cdot \texttt{fpf} & \text{if } cm = \text{FTR} \\ R_l \cdot \texttt{rdf} & \text{if } cm = \text{RND} \\ R_l \cdot 2^{\texttt{retries}} \cdot \texttt{rdf} & \text{if } cm = \text{RDR} \\ R_l \cdot 2^{\texttt{retries}} \cdot \texttt{fpf} & \text{if } cm = \text{AGF} \end{cases} \tag{5}$$

Next we compute the probabilities $p^{cm}_{b\_in}$ and $p^{cm}_{l\_in}$ with which a bogus packet and a legitimate packet, respectively, pass through DF, and the probabilities $p^{cm}_{b\_out}$ and $p^{cm}_{l\_out}$ with which a bogus packet and a legitimate packet, respectively, are dropped at DF. They are $p^{cm}_{t\_d} = R^{cm}_{t\_d}/(R^{cm}_{t\_in} + R^{cm}_{t\_out})$ where the packet type t is b or l, and the direction d is in or out.

Once a packet passes through DF, VS receives it subject to VS's bandwidth availability. So, $p^{cm}_{b\_receive}$ can be computed as the probability that bogus packet arrives at VS (i.e., VS's network) and is not dropped by VS due to bandwidth limit (i.e., network congestion). The $p^{cm}_{b\_drop}$ can be computed as the probability that a bogus packet is dropped by DF, or a bogus packet arrives at VS and is dropped by VS due to bandwidth limit. The probabilities $p^{cm}_{l\_receive}$ and $p^{cm}_{l\_drop}$ can be computed similarly. This leads to the following equations, where p^{cm}_{drop} is the probability that a (bogus or legitimate) packet transmitted by DF gets dropped by VS due to bandwidth limit, and $p^{cm}_{receive}$ is the probability that a (bogus or legitimate) packet transmitted by DF is not dropped by VS due to bandwidth limit. Note that all of these probabilities implicitly depend on state variables such as `zombies` and `df`.

$$\begin{aligned} R^{cm}_{in} &= R^{cm}_{b\_in} + R^{cm}_{l\_in} \\ p^{cm}_{drop} &= \begin{cases} 0 & \text{if } \texttt{BW} \geq R^{cm}_{in} \\ (R^{cm}_{in} - \texttt{BW})/R^{cm}_{in} & \text{otherwise} \end{cases} \\ p^{cm}_{receive} &= 1 - p^{cm}_{drop} \\ p^{cm}_{t\_receive} &= p^{cm}_{t\_in} \cdot p^{cm}_{receive} \\ p^{cm}_{t\_drop} &= p^{cm}_{t\_in} \cdot p^{cm}_{drop} + p^{cm}_{t\_out} \end{aligned}$$

where the packet type t in the last two equations is b or l.

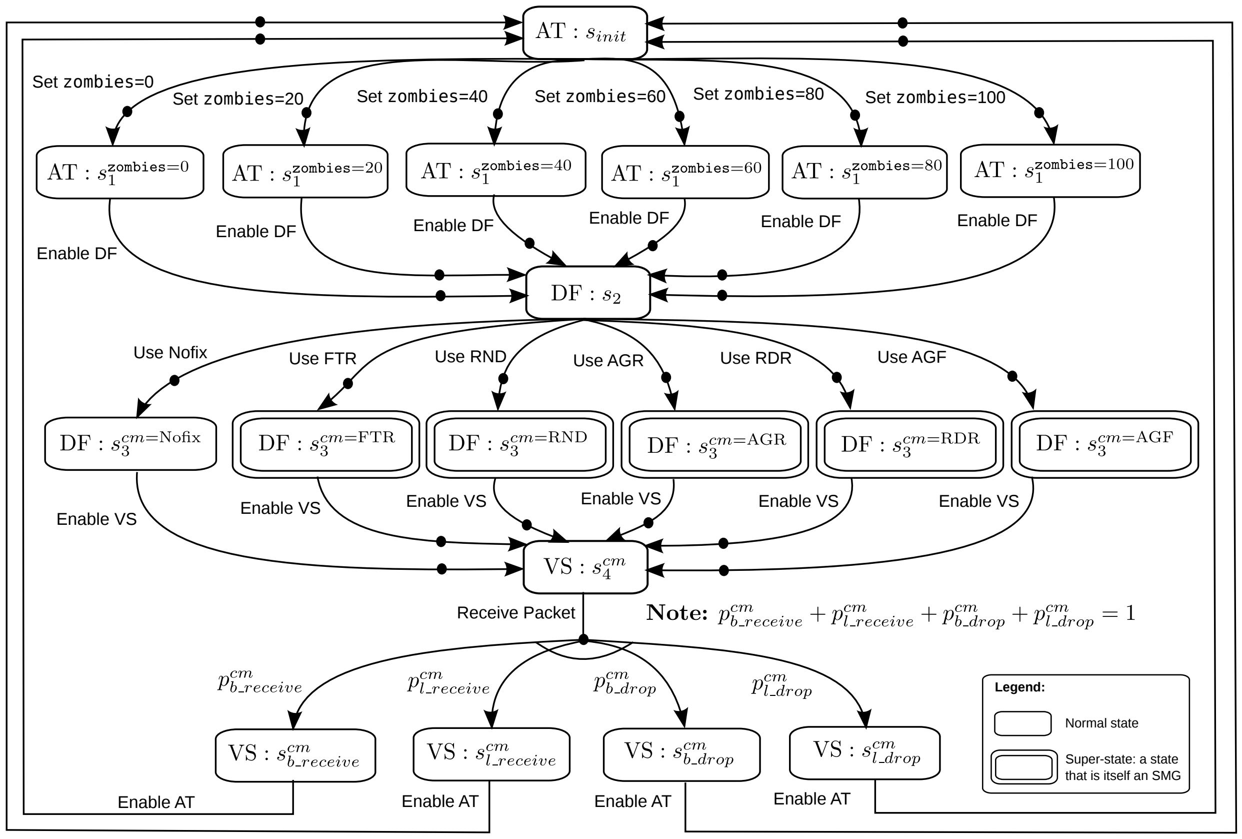
D. Attacker and Defender Payoffs and Optimal Attack and Defense Strategies

Attacker's payoffs. We define two payoff functions for the attacker, to illustrate how the payoff function affects the generated optimal strategies.

`AttackerPayoff_1` maximizes the difference between bogus and legitimate packets received by the victim (VS).

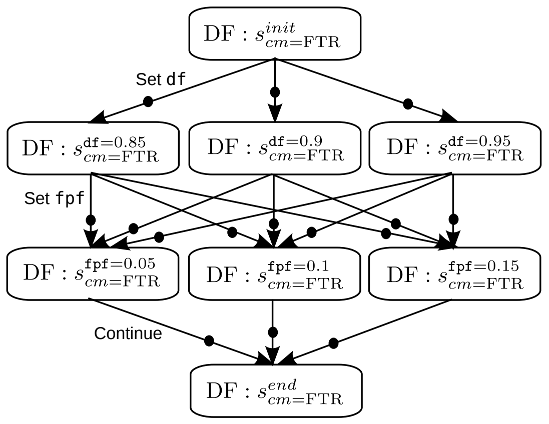
```
rewards "AttackerPayoff_1"
 turn=VS & legitPktRcvd : -1;
 turn=VS & bogusPktRcvd : 1;
endrewards
```

`AttackerPayoff_2` is a refinement of `AttackerPayoff_1` that takes the attack cost into account by dividing by the "benefit" by the number of `zombies`, which is a measure of attack cost. So, `AttackerPayoff_2`



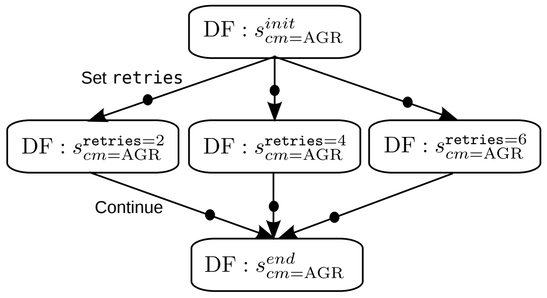
AT : s_{init}
Set zombies=0
Set zombies=20
Set zombies=40
Set zombies=60
Set zombies=80
Set zombies=100
AT : $s_1^{\texttt{zombies}=0}$
AT : $s_1^{\texttt{zombies}=20}$
AT : $s_1^{\texttt{zombies}=40}$
AT : $s_1^{\texttt{zombies}=60}$
AT : $s_1^{\texttt{zombies}=80}$
AT : $s_1^{\texttt{zombies}=100}$
Enable DF
DF : s_2
Use Nofix
Use FTR
Use RND
Use AGR
Use RDR
Use AGF
DF : $s_3^{cm=\text{Nofix}}$
DF : $s_3^{cm=\text{FTR}}$
DF : $s_3^{cm=\text{RND}}$
DF : $s_3^{cm=\text{AGR}}$
DF : $s_3^{cm=\text{RDR}}$
DF : $s_3^{cm=\text{AGF}}$
Enable VS
VS : s_4^{cm}
Receive Packet
Note: $p^{cm}_{b\_receive} + p^{cm}_{l\_receive} + p^{cm}_{b\_drop} + p^{cm}_{l\_drop} = 1$
$p^{cm}_{b\_receive}$
$p^{cm}_{l\_receive}$
$p^{cm}_{b\_drop}$
$p^{cm}_{l\_drop}$
VS : $s^{cm}_{b\_receive}$
VS : $s^{cm}_{l\_receive}$
VS : $s^{cm}_{b\_drop}$
VS : $s^{cm}_{l\_drop}$
Enable AT
Legend:
Normal state
Super-state: a state that is itself an SMG

Fig. 3. The DNS BAA SMG. A state s controlled by player P is shown as $P : s$. Each transition is labeled with an action and a transition probability. A solid dot separates the action from the probability. When the transition probability is not shown on an edge, the probability is 1. Single ovals denote any states that are reachable via preceding actions. Double ovals denote super-states, which are themselves SMGs. Figures 4 and 5 expand the super-states DF : $s_3^{cm=\text{FTR}}$ and DF : $s_3^{cm=\text{AGR}}$, respectively. The expansions of the super-states DF : $s_3^{cm=\text{RND}}$, DF : $s_3^{cm=\text{RDR}}$, and DF : $s_3^{cm=\text{AGF}}$ are not shown due to space constraints.



DF : $s^{init}_{cm=\text{FTR}}$
Set df
DF : $s^{\texttt{df}=0.85}_{cm=\text{FTR}}$
DF : $s^{\texttt{df}=0.9}_{cm=\text{FTR}}$
DF : $s^{\texttt{df}=0.95}_{cm=\text{FTR}}$
Set fpf
DF : $s^{\texttt{fpf}=0.05}_{cm=\text{FTR}}$
DF : $s^{\texttt{fpf}=0.1}_{cm=\text{FTR}}$
DF : $s^{\texttt{fpf}=0.15}_{cm=\text{FTR}}$
Continue
DF : $s^{end}_{cm=\text{FTR}}$

Fig. 4. Expansion of super-state DF : $s_3^{cm=FTR}$ from Figure 3. This shows how DF chooses values for parameters of FTR.



DF : $s^{init}_{cm=\text{AGR}}$
Set retries
DF : $s^{\texttt{retries}=2}_{cm=\text{AGR}}$
DF : $s^{\texttt{retries}=4}_{cm=\text{AGR}}$
DF : $s^{\texttt{retries}=6}_{cm=\text{AGR}}$
Continue
DF : $s^{end}_{cm=\text{AGR}}$

Fig. 5. Expansion of super-state DF : $s_3^{cm=AGR}$ from Figure 3. This shows how DF chooses a value for the parameter of AGR.

time	zombies	Countermeasure	df	fpf	rdf	retries
1	100	RDR			0.99	6
2	100	RDR			0.99	6
3	0	Nofix				
4	100	RDR			0.99	6
5	100	RDR			0.99	6
6	0	Nofix				
7	100	AGF	0.90	0.05		6
8	100	RDR			0.99	6
9	0	Nofix				
10	100	RDR			0.99	6
11	100	AGF	0.85	0.05		6
12	0	Nofix				
13	100	RDR			0.99	6
14	100	AGF	0.85	0.05		6
15	100	RDR			0.99	6

TABLE I
OPTIMAL ATTACK AND DEFENSE STRATEGIES GENERATED FOR `AttackerPayoff_1`. EMPTY CELLS INDICATE THAT THE CORRESPONDING COUNTERMEASURE PARAMETER IS NOT APPLICABLE FOR THE SELECTED COUNTERMEASURE.

time	zombies	Countermeasure	df	fpf	rdf	retries
1	100	FTR	0.95	0.05		
2	100	FTR	0.95	0.55		
3	0	Nofix				
4	100	FTR	0.95	0.05		
5	100	FTR	0.95	0.05		
6	0	Nofix				
7	100	FTR	0.95	0.05		
8	100	FTR	0.95	0.05		
9	0	Nofix				
10	100	FTR	0.95	0.05		
11	100	FTR	0.95	0.05		
12	0	Nofix				
13	100	FTR	0.95	0.05		
14	100	FTR	0.95	0.05		
15	100	FTR	0.95	0.05		

TABLE II
OPTIMAL ATTACK AND DEFENSE STRATEGIES GENERATED FOR `AttackerPayoff_2`. EMPTY CELLS INDICATE THAT THE CORRESPONDING COUNTERMEASURE PARAMETER IS NOT APPLICABLE FOR THE SELECTED COUNTERMEASURE.

maximizes the difference between legitimate packets dropped per zombie and legitimate packets received by the victim (VS) per zombie. If `zombies` > 0, `AttackerPayoff_2` assigns a positive unit reward per zombie when a legitimate packet is dropped, and it assigns a negative unit reward per zombie when a legitimate packet is received. When `zombies` $= 0$, i.e., if the attack is paused, rewards per zombie are not used, in order to avoid the *Division by zero*. However, when the attack is paused, more and more legitimate packets are received by the victim, which is not beneficial to the attacker. So, a negative unit reward is assigned when a legitimate packet is received. Similarly, a small number of legitimate packets may be dropped, if the incoming legitimate packets will exhaust the victim's bandwidth. This behavior is beneficial to the attacker. So, a positive unit reward is assigned when a legitimate packet is dropped.

```
rewards "AttackerPayoff_2"
turn=VS & legitPktDrop & zombies>0 : 1/zombies;
turn=VS & legitPktRcvd & zombies>0 : -1/zombies;
turn=VS & legitPktDrop & zombies=0 : 1;
turn=VS & legitPktRcvd & zombies=0 : -1;
endrewards
```

Defender's payoffs. Since our game is a zero-sum game, the defender's payoff is exactly opposite to the attacker's payoff. The defender always chooses a move that would be most disadvantageous to the attacker. So, we do not need to explicitly define the defender's payoffs.

Generating optimal strategies. To generate optimal attack and defense strategies, we define rPATL reward-based properties for the coalition of players containing attacker AT. These reward-based properties have the form:

```
<<AT>>Rmax{"AttackerPayoff_n"} =? [F time=maxTime]
```

When PRISM-games evaluates such properties, it returns the expected accumulated reward value and the optimal strategies for the two coalitions of players.

V. EXPERIMENTAL RESULTS

To limit the number of reachable states, we set `maxTime` to 15, and we allow the attacker and defender to select among a few values for each parameter. Specifically, AT can set `zombies` to 0, 20, 40, 60, 80, or 100, and DF can set `df` to one of the representative values 0.85, 0.9, or 0.95. Similarly, DF can set `fpf` to 0.05, 0.1, or 0.15, `rdf` to 0.81, 0.9, or 0.99, and `retries` to 2, 4, or 6. To illustrate the interesting temporal behaviors that may appear in the generated optimal strategies when the attacker is subject to limits on attack duration, we set `MAX_SUCC_ATTACKS` to 3 and `ATTACKER_LATENCY` to 2; these values are relatively small but allow interesting temporal behavior to occur within the time limit imposed by `maxTime`.

With these settings, PRISM-games constructs the state graph, consisting of 773,906 states and 1,591,981 transitions, in 20.2 seconds. PRISM-games model checking algorithms compute fixpoints up to a desired *convergence threshold*, i.e., the maximum difference between solution values observed in successive iterations. We use the default threshold of 10^{-6}. The lower bound on the number of iterations required to achieve the desired convergence threshold is exponential in the game size [24]. PRISM-games computed the optimal strategies for `AttackerPayoff_1` and `AttackerPayoff_2` in 71.0 seconds and 52.4 seconds, respectively.

Results for `AttackerPayoff_1`. Table I presents the optimal attack and defense strategies for `AttackerPayoff_1`. We observe that the attacker never attacks for three continuous steps, so as to escape the latency, i.e., so that it can attack again after one step instead of `ATTACKER_LATENCY` steps. We also see that the attacker uses the highest possible number of zombies. The defender tries to drop as many bogus packets as possible by using RDR with `rdf` and `retries` set to their highest possible values (0.99 and 6, respectively). With these settings, RDR reduces the number of bogus packets received more effectively than FTR. On a few occasions, we see that defender also uses AGF with `df` set to less than its maximum value of 0.95, `fpf` set to its lowest possible value of 0.05, and `retries` set to its highest possible value of 6. Finally, we observe that when the attacker pauses the attack, the defender turns off all countermeasures.

Results for `AttackerPayoff_2`**.** Table II presents the optimal attack and defense strategies generated for `AttackerPayoff_2`. From Table II, we again see that the attacker never attacks for `MAX_SUCC_ATTACKS` consecutive steps. We also observe that, as one might expect, the defender sets `df` to the highest possible value and `fpf` to the lowest possible value, and the attacker sets `zombies` to the highest possible value. We once again see that when the attacker pauses the attack, the defender turns off all countermeasures.

Note that none of the optimal strategies generated for this payoff function use RND or RDR, because when RND is used, the values of $p^{\mathrm{RND}}_{l\_receive}$ are always less than the corresponding probability values observed for FTR. Moreover, $p^{\mathrm{RND}}_{l\_receive} = p^{\mathrm{Nofix}}_{l\_receive}$, because RND drops `rdf` of legitimate and bogus traffic, so selecting RND or RDR does not help the victim to increase the fraction of legitimate packets received.

Note also that none of the optimal strategies generated for this payoff function use AGR, because increasing the number of retries leads to bandwidth saturation, thereby increasing the probability of legitimate packets being dropped, and this offsets the benefit of increasing the volume of legitimate traffic; in contrast, FTR drops only a small percentage of legitimate packets, while dropping a large percentage of bogus packets, allowing significantly more legitimate packets to get through, improving the defender's payoff.

We also ran a variant of this experiment in which we fixed `df` to 0.9 and `fpf` to 0.1. In this case, the attacker uses only 75 zombies, because `AttackerPayoff_2` assigns unit reward *per zombie* when a legitimate packet is dropped and a unit negative reward *per zombie* when a legitimate packet is received. Therefore, increasing the number of zombies is advantageous for the attacker only if it increases the legitimate drop rate by a sufficiently large amount to compensate for the additional cost, which is reflected in the larger denominator in the payoff function.

VI. Related Work

Game-theoretic techniques have been previously applied to DoS and DDoS attacks, where an attack is modeled as a two-player game between an attacker and a defender that resorts to filtering and rate limiting as attack countermeasures. The majority of these approaches [34], [35], [36], [37], [38] analyze the effectiveness of individual countermeasures; however, [39] studies multi-layer protection, where filtering, rate-limiting, and bandwidth capacity extension are used in conjunction.

A one-shot, zero-sum game between an attacker and a defender is also presented in [40]. The attacker's goal is to find optimal values for the number of attacking nodes, the malicious traffic probability distributions for each node, and other related parameters.

In all aforementioned works, payoff functions defined for the attacker and defender reflect the benefits obtained and the costs incurred when choosing a particular strategy. The typical metrics used as benefits and costs are: the bandwidth share utilized by the legitimate traffic, the number of legitimate packets incorrectly dropped, the number of bogus packets allowed to pass through the filter, and the costs associated with adding more bandwidth or using more zombies. The payoffs are defined as functions of various thresholds and drop rates (DR) used by filtering mechanism, rate at which legitimate packets arrive (LR), number of zombies used by the attacker (Z), rate at which each zombie sends bogus traffic (BR), and the manner in which the attack traffic is generated (ATG), i.e., whether the attack traffic is sent continuously or in bursts.

Once payoff functions are defined, they are used to determine the Nash equilibria strategies for the attacker and defender. Nash equilibria are computed by various game solvers and the results are verified by simulating the attack using network simulation tools such as NS2. The Nash equilibria return the values of DR that would maximize the defender's payoff, and values of values of Z, BR, and ATG that would maximize attacker's payoff. Neither the defender nor the attacker has any incentive to deviate from the Nash equilibria.

Our work is different in several respects. Along with FTR and RND, we analyze AGR and composite countermeasures RDR and AGF that contain AGR. Our two-player SMG model of the DNS BAA results in the generation of more interesting optimal attack and defense strategies, where the number of zombies employed in the attack and the countermeasures selected by the defender vary over the duration of the attack. We implemented the two-player game for DNS BAA using PRISM-games, which uses reward-based probabilistic model checking to generate optimal strategies. Therefore, we do not need to use a separate game solver. Also, reward-based properties in PRISM offer a convenient way to define payoff functions, allowing us to easily explore the effects of different payoff functions on the optimal strategies so generated.

VII. Conclusions

We developed a stochastic game-based model of the DNS BAA attack at a suitable level of abstraction to allow efficient analysis. We formally analyzed the BAA by formulating the model in the PRISM-games probabilistic model checker. This approach allowed us to generate new optimal attack and defense strategies corresponding to different payoff functions. An optimal attack strategy generated in this manner may vary the number of zombies used to launch the attack, or even periodically pause the attack. An optimal defense strategy generated in this manner may involve switching among basic and composite countermeasures, along with updating countermeasure parameter settings, as the attack progresses.

One direction for future work is to consider more complex payoff functions. For example, the attacker may try to maximize the percentage of legitimate packets dropped per zombie. This payoff function needs to be evaluated over the entire run, which is not directly supported by PRISM-games. In order to define such a payoff function, we need to extend the model with additional state variables, e.g., counters for the numbers of legitimate packets received and legitimate packets dropped. Furthermore, if the number of zombies may vary during the run, and we want to use the average number of zombies when computing these percentages per zombie, then an additional integer variable is needed. Such counters would significantly increase the number of reachable states and therefore the time and space needed for the analysis. New optimizations may be needed to make such analysis feasible. Another direction for future work is to validate the results of our model on a real network or by using network simulation tools like NS2. Finally, we are also interested to analyze the DNS BAA as a two-player game with imperfect information [41].

Acknowledgements: Research supported in part by NSF grants CNS-0831298, CCF-0926190, CCF-1018459, and CCF-1248184 and AFOSR grant FA0550-09-1-0481.

References

[1] T. Deshpande, P. Katsaros, S. Basagiannis, and S. A. Smolka, "Formal analysis of the DNS bandwidth amplification attack and its countermeasures using probabilistic model checking," in *Proc. 13th IEEE Int. Symp. on High-Assurance Systems Engineering (HASE)*, 2011, pp. 360–367.

[2] B. Al-Duwairi, "Mitigation and traceback countermeasures for DDoS attacks," Ph.D. dissertation, Iowa State University, 2005.

[3] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka, "Model repair for probabilistic systems," in *Proc. 17th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11)*, ser. LNCS, vol. 6605, 2011, pp. 326–340.

[4] M. Anagnostopoulos, G. Kambourakis, P. Kopanos, G. Louloudakis, and S. Gritzalis, "Dns amplification attack revisited," *Computers & Security*, vol. 39, Part B, no. 0, pp. 475 – 485, 2013.

[5] T. Rozekrans and J. De Koning, "Defending against DNS reflection amplification attacks," University of Amsterdam, Tech. Rep., Feb. 2013, http://www.nlnetlabs.nl/downloads/publications/report-rrl-dekoning-rozekrans.pdf.

[6] M. Kwiatkowska, G. Norman, and D. Parker, "PRISM 4.0: Verification of probabilistic real-time systems," in *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, ser. LNCS, G. Gopalakrishnan and S. Qadeer, Eds., vol. 6806. Springer, 2011, pp. 585–591.

[7] S. Friedl, "An illustrated guide to the Kaminsky DNS vulnerability," Aug 2008, http://unixwiz.net/techtips/iguide-kaminsky-dns-vuln.html.

[8] Microsoft TechNet, "How DNS query works," Jan. 2005, http://technet.microsoft.com/en-us/library/cc775637(WS.10).aspx.

[9] "Recursive and iterative queries," Microsoft, Tech. Rep., http://technet.microsoft.com/en-us/library/cc961401.aspx.

[10] R. Vaughn and G. Evron, "DNS amplification attacks," 2006, http://www.isotf.org/news/DNS-Amplification-Attacks.pdf.

[11] M. Prince, "Deep inside a DNS amplification DDoS attack," Oct 2012, http://blog.cloudflare.com/deep-inside-a-dns-amplification-ddos-attack.

[12] G. Kambourakis, T. Moschos, D. Geneiatakis, and S. Gritzalis, "Detecting DNS amplification attacks," in *Critical Information Infrastructures Security*, ser. LNCS, 2008, vol. 5141, pp. 185–196.

[13] A. Cowperthwaite and A. Somayaji, "The futility of DNSSec," in *Proc. 5th Annual Symp. Information Assurance (ASIA'10)*, 2010, pp. 2–8.

[14] The Measurement Factory, "DNS survey: Open resolvers," 2013, http://dns.measurement-factory.com/surveys/openresolvers.html.

[15] H. Sun, Y. Zhaung, and H. J. Chao, "A principal components analysis-based robust DDoS defense system," in *Proc. 2008 IEEE Int. Conf. on Communications (ICC'08)*, May 2008, pp. 1663–1669.

[16] T. Peng, C. Leckie, and K. Ramamohanarao, "Protection from distributed denial of service attacks using history-based IP filtering," in *Proc. 2003 IEEE Int. Conf. on Communications (ICC'03)*, vol. 1, May 2003, pp. 482–486.

[17] C. Jin, H. Wang, and K. G. Shin, "Hop-count filtering: an effective defense against spoofed DDoS traffic," in *Proc. 10th ACM Conf. on Computer and Communications Security (CCS'03)*, 2003, pp. 30–41.

[18] S. Ioannidis, K. G. Anagnostakis, J. Ioannidis, and A. D. Keromytis, "xPF: packet filtering for low-cost network monitoring," in *Workshop on High Performance Switching and Routing Merging Optical and IP Technologies*, 2002, pp. 116–120.

[19] Z. Wu, M. Xie, and H. Wang, "Swift: a fast dynamic packet filter," in *Proc. 5th USENIX Symposium on Networked Systems Design and Implementation (NSDI'08)*, 2008, pp. 279–292.

[20] A. Mankin, "Random drop congestion control," in *Proc. ACM Symposium on Communications Architectures & Protocols (SIGCOMM'90)*, 1990, pp. 1–7.

[21] V. Jacobson, "Minutes of the performance working group," in *Proc. Cocoa Beach Internet Engineering Task Force*, Apr 1989.

[22] M. Walfish, M. Vutukuru, H. Balakrishnan, D. Karger, and S. Shenker, "DDoS defense by offense," *ACM Transactions on Computer Systems*, vol. 28, pp. 3:1–3:54, Aug 2010.

[23] S. Khanna, S. S. Venkatesh, O. Fatemieh, F. Khan, and C. A. Gunter, "Adaptive selective verification," in *Proc. 27th IEEE Conf. on Computer Communications (INFOCOM'08)*, Apr 2008, pp. 529–537.

[24] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, "Automatic verification of competitive stochastic systems," *Formal Methods in System Design*, vol. 43, no. 1, pp. 61–92, 2013.

[25] M. Shor, "Payoff," in *Dictionary of Game Theory Terms, Game Theory.net*, 2005, http://www.gametheory.net/dictionary/Payoff.html.

[26] A. Neyman and S. Sorin, *Stochastic Games and Applications*, ser. Nato Science Series. Kluwer Academic Press, 2003, vol. 570.

[27] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic," *Journal of the ACM*, vol. 49, no. 5, pp. 672–713, Sep. 2002.

[28] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker, "Automated verification techniques for probabilistic systems," in *Formal Methods for Eternal Networked Software Systems (SFM'11)*, ser. LNCS, vol. 6659. Springer, 2011, pp. 53–113.

[29] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, ser. LNCS (Tutorial Volume), M. Bernardo and J. Hillston, Eds., vol. 4486, 2007, pp. 220–270.

[30] "Zero-sum game," *Wikipedia, The Free Encyclopedia*, 2013, http://en.wikipedia.org/w/index.php?title=Zero-sum_game&oldid=576815550.

[31] T. Chen, V. Forejt, M. Kwiatkowska, D. Parker, and A. Simaitis, "PRISM-games: A model checker for stochastic multi-player games," in *Proc. 19th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'13)*, ser. LNCS, N. Piterman and S. Smolka, Eds., vol. 7795. Springer, 2013, pp. 185–191.

[32] "Nash equilibrium," *Wikipedia, The Free Encyclopedia*, 2013, http://en.wikipedia.org/w/index.php?title=Nash_equilibrium&oldid=560155666.

[33] Y. Xuebiao, W. Xin, L. Xiaodong, and Y. Baoping, "DNS measurements at the .CN TLD servers," in *Proc. 6th Int. Conf. on Fuzzy Systems and Knowledge Discovery (FSKD'09)*, vol. 7, Aug 2009, pp. 540–545.

[34] M. E. Snyder, R. Sundaram, and M. Thakur, "A game-theoretic framework for bandwidth attacks and statistical defenses," in *Proc. the 32nd IEEE Conference on Local Computer Networks (LCN '07)*. IEEE Computer Society, 2007, pp. 556–566.

[35] P. Shi and Y. Lian, "Game-theoretical effectiveness evaluation of DDoS defense," in *Proc. Seventh International Conference on Networking (ICN '08)*, 2008, pp. 427–433.

[36] W. Zang, P. Liu, and M. Yu, "How resilient is the Internet against DDoS attacks? – A game theoretic analysis of signature-based rate limiting," in *International Journal of Intelligent Control and Systems*, vol. 12, no. 4, December 2007, pp. 307–316.

[37] H. Bedi, S. Roy, and S. Shiva, "Game theory-based defense mechanisms against DDoS attacks on TCP/TCP-friendly flows," in *Proc. IEEE Symposium on Computational Intelligence in Cyber Security (CICS'11)*, 2011, pp. 129–136.

[38] A. Agah and S. Das, "Preventing DoS attacks in wireless sensor networks: A repeated game theory approach," *International Journal of Network Security*, pp. 145–153, 2007.

[39] G. Yan, R. Lee, A. Kent, and D. Wolpert, "Towards a Bayesian network game framework for evaluating DDoS attacks and defense," in *Proc. 2012 ACM Conference on Computer and Communications Security (CCS'12)*, 2012, pp. 553–566.

[40] T. Spyridopoulos, G. Karanikas, T. Tryfonas, and G. Oikonomou, "A game theoretic defence framework against DoS/DDoS cyber attacks," *Computers & Security*, vol. 38, pp. 39 – 50, 2013.

[41] Y. Soupionis, R.-A. Koutsiamanis, P. Efraimidis, and D. Gritzalis, "A game-theoretic analysis of preventing spam over internet telephony via audio CAPTCHA-based authentication," *(To appear in) Journal of Computer Security*, 2014.