Proceedings of the Twenty-Fifth Innovative Applications of Artificial Intelligence Conference

Multiagent Router Throttling: Decentralized Coordinated Response Against DDoS Attacks

Kleanthis Malialis and **Daniel Kudenko**
Department of Computer Science
University of York, UK
{malialis,kudenko}@cs.york.ac.uk

Abstract

Distributed denial of service (DDoS) attacks constitute a rapidly evolving threat in the current Internet. In this paper we introduce Multiagent Router Throttling, a decentralized DDoS response mechanism in which a set of upstream routers independently learn to throttle traffic towards a victim server. We compare our approach against a baseline and a popular throttling technique from the literature, and we show that our proposed approach is more secure, reliable and cost-effective. Furthermore, our approach outperforms the baseline technique and either outperforms or has the same performance as the popular one.

Introduction

One of the most serious threats in the current Internet is posed by distributed denial of service (DDoS) attacks, which target the availability of a system (Mirkovic and Reiher 2004). A DDoS attack is a highly coordinated attack where the attacker (or attackers) takes under his control a large number of hosts, called the botnet (network of bots), which start bombarding the target when they are instructed to do so. Such an attack is designed to exhaust a server's resources or congest a network's infrastructure, and therefore renders the victim incapable of providing services to its legitimate users.

Router Throttling (Yau et al. 2005) is a popular approach to defend against DDoS attacks, where the victim server signals a set of upstream routers to throttle traffic towards it. In this paper, we introduce Multiagent Router Throttling, a novel throttling approach where multiple independent reinforcement learning agents are installed on a set of upstream routers and learn to throttle traffic towards the victim server. Our contributions in this paper are the following:

- **Decentralized response:** One of the novel characteristics of our approach is its decentralized architecture and response to the DDoS threat. We compare our approach against the Baseline Router Throttling and AIMD Router Throttling (Yau et al. 2005) techniques, a baseline and a popular throttling technique respectively from the literature, and we show that our proposed approach is more secure, reliable and cost-effective.
- **Multiagent learning:** Due to the high complexity and multidimensionality of the DDoS threat, multiagent reinforcement learning creates an automated and effective response against DDoS attacks. The network environment is highly dynamic and our approach provides adaptable behaviors over frequent environmental changes. We evaluate our approach in a series of attack scenarios with increasing sophistication and we show that our approach outperforms the Baseline (Yau et al. 2005) technique, and either outperforms or has the same performance as the AIMD (Yau et al. 2005) technique.
- **Intrusion response:** There is an extensive literature regarding the application of machine learning to intrusion detection, specifically anomaly detection where no action is performed beyond triggering an intrusion alarm when something anomalous is detected. Our work investigates the applicability of machine learning to intrusion response. In this paper, we investigate the applicability of multiagent reinforcement learning to DDoS response.

Background

Reinforcement Learning

Reinforcement learning is a paradigm in which an active decision-making agent interacts with its environment and learns from reinforcement, that is, a numeric feedback in the form of reward or punishment (Sutton and Barto 1998). The feedback received is used to improve the agent's actions. Typically, reinforcement learning uses a Markov Decision Process (MDP) as a mathematical model.

An MDP is a tuple $\langle S, A, T, R \rangle$, where S represents the state space, A represents the action space, $T(s, a, s') = Pr(s'|s, a)$ is the transition probability function which returns the probability of reaching state s' when action a is executed in state s, and $R(s, a, s')$ is the reward function which returns the immediate reward r when action a executed in state s results in a transition to state s'. The problem of solving an MDP is to find a policy (i.e. a mapping from states to actions) which maximises the accumulated reward. When the environment dynamics (transition probabilities and reward function) are available, this task can be solved using dynamic programming (Bertsekas and Tsitsiklis 1996).

In most real-world domains, the environment dynamics are not available and therefore the assumption of perfect

problem domain knowledge makes dynamic programming to be of limited practicality. The concept of an iterative approach constitutes the backbone of the majority of reinforcement learning algorithms. These algorithms apply so called temporal-difference updates to propagate information about values of states, $V(s)$, or state-action, $Q(s, a)$, pairs. These updates are based on the difference of the two temporally different estimates of a particular state or state-action value. The SARSA algorithm is such a method (Sutton and Barto 1998). After each real transition, $(s, a) \rightarrow (s', r)$, in the environment, it updates state-action values by the formula:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)] \quad (1)$$

where α is the rate of learning and γ is the discount factor. It modifies the value of taking action a in state s, when after executing this action the environment returned reward r, moved to a new state s', and action a' was chosen in state s'.

The exploration-exploitation trade-off constitutes a critical issue in the design of a reinforcement learning agent. It aims to offer a balance between the exploitation of the agent's knowledge and the exploration through which the agent's knowledge is enriched. A common method of doing so is ϵ-greedy, where the agent behaves greedily most of the time, but with a probability ϵ it selects an action randomly. To get the best of both exploration and exploitation, it is advised to reduce ϵ over time (Sutton and Barto 1998).

Applications of reinforcement learning to multiagent systems typically take one of two approaches; multiple individual learners or joint action learners (Claus and Boutilier 1998). The former is the deployment of multiple agents each using a single-agent reinforcement learning algorithm. The latter is a group of multiagent specific algorithms designed to consider the existence of other agents.

Multiple individual learners assume any other agents to be a part of the environment and so, as the others simultaneously learn, the environment appears to be dynamic as the probability of transition when taking action a in state s changes over time. To overcome the appearance of a dynamic environment, joint action learners were developed that extend their value function to consider for each state the value of each possible combination of actions by all agents.

The consideration of the joint action causes an exponential increase in the number of values that must be calculated with each additional agent added to the system. Typically, joint action learning algorithms have only been demonstrated in trivial problem domains whilst applications in complex systems most often implement multiple individual learners. For these reasons, this work will focus on multiple individual learners and not joint action learners.

Distributed Denial of Service Attacks

DDoS attacks (Mirkovic and Reiher 2004) constitute a major and evolving problem in the current Internet. Such an attack targets the availability of computer or network resources, thus not allowing its legitimate users to access resources in a timely and efficient manner. The most common type of DDoS attacks is the flooding attacks, where the attacker starts bombarding the victim by sending large volumes of traffic towards it, thus causing severe congestion to the victim.

A DDoS attack is a highly coordinated attack; the strategy behind it is described by the agent-handler model (Mirkovic and Reiher 2004). The model consists of four elements, the attackers, handlers, agents and victim. A handler (or master) and the agent (or slave or deamon or zombie) are hosts compromised by the attackers, which constitutes the botnet. Specifically, the attackers install a malicious software called Trojan on vulnerable hosts to compromise them, thus being able to communicate with and control them. The attackers communicate with the handlers, which in turn control the agents in order to launch a DDoS attack. Typically, the users of the agent systems are not aware of their involvement in a coordinated DDoS attack.

The DDoS threat is challenging because of the following reasons (Mirkovic, Robinson, and Reiher 2003):

- **Distributed traffic:** The traffic flows originate from agent machines spread all over the Internet, which they all aggregate at the victim.
- **Large volume:** The large volume of the aggregated traffic is unlikely to be stopped by a single defense point near the victim.
- **Large number of agents:** The number of compromised agent machines is large, thus making an automated response a necessary requirement.
- **Similarity to legitimate packets:** DDoS packets appear to be similar to legitimate ones, since the victim damage is caused by the aggregated volume and not packet contents. A defense system cannot make an accurate decision based on a packet-by-packet basis. It requires to keep some statistical data in order to correlate packets and detect anomalies, for example, "all traffic directed towards a specific destination address".
- **Difficult to traceback:** It is difficult to discover even the agent machines, let alone the actual attackers, firstly because of the agent-handler model's architecture, and secondly because of IP spoofing. With IP spoofing an attacker hides his true identity by placing a fake source address in the IP packet's source address.

It is evident that to combat the distributed nature of these attacks, a distributed coordinated defense mechanism is necessary, where many defensive nodes, across different locations cooperate in order to stop or reduce the flood.

Related Work

This section describes work related to ours, focusing on distributed, cooperated defense mechanisms. One of the first and most influential work in the field is the Aggregate-based Congestion Control (ACC) and Pushback mechanisms by Mahajan et al. (2002). The authors view the DDoS attacks as a router congestion problem. The **aggregate** is defined as the traffic that is directed towards a specific destination address i.e. the victim (note that source addresses cannot be trusted due to IP spoofing). A local ACC agent is installed on the victim's router which monitors the drop history. If the drop history deviates from the normal, the local ACC reduces the

throughput of the aggregate by calculating and setting a rate-limit.

Pushback is an optional cooperative mechanism. The local ACC can optionally request from adjacent upstream routers to rate limit the aggregate according to a max-min fashion, a form of equal-share fairness, where bandwidth allocation is equally divided among all adjacent upstream routers. Rate limiting of the aggregate recursively propagates upstream towards its sources. Pushback relies on a contiguous deployment and requires a secure and reliable communication. The major limitation of Pushback is that is causes collateral damage, that is, when legitimate traffic is punished, in this case rate limited, along with the attack traffic. This is because the resource sharing starts at the congested point, where the traffic is highly aggregated and contains a lot of legitimate traffic within it.

Another popular work is the Router Throttling mechanism by Yau et al. (2005). The authors view the DDoS attacks as a resource management problem, and they adopt a proactive, server-initiated approach, which according to Douligeris and Mitrokotsa (2004) similar techniques to this are used by network operators. The approach is as follows. When a server operates below an upper boundary U_s, it needs no protection (this includes cases of weak or ineffective DDoS attacks). When the server experiences heavy load, it requests from upstream routers to install a throttle on the aggregate. In case the server load is still over the upper boundary U_s, the server asks from upstream routers to increase the throttle. If the server load drops below a lower boundary L_s, the server asks the upstream routers to relax the throttle. The goal is to keep the server load within the boundaries $[L_s, U_s]$ during a DDoS attack. Router Throttling requires a secure and reliable communication. However, unlike Pushback, it does not require a contiguous deployment; it is more of an end-to-end approach initiated by the server and therefore collateral damage is significantly reduced.

The authors present the Baseline Router Throttling approach in which all upstream routers throttle traffic towards the server, by forwarding only a fraction of it. This approach penalizes all upstream routers equally, irrespective of whether they are well behaving or not. The authors then propose the AIMD (additive-increase/multiplicative-decrease) Router Throttling algorithm, which installs a uniform leaky bucket rate at each upstream router, and guarantees the max-min fairness.

Multiagent Router Throttling

Network Model and Assumptions

The network model is similar to the one used by Yau et al. (2005). A network is a connected graph $G = (V, E)$, where V is the set of nodes and E is the set of edges. All leaf nodes are hosts and denoted by H. Hosts can be traffic sources and are not trusted because of IP spoofing. An internal node represents a router, which forwards or drops traffic received from its connected hosts or peer routers. The set of routers are denoted by R, and they are assumed to be trusted, i.e. not to be compromised. This assumption is rational since it is much more difficult to compromise a router than an end host or server, because routers have a limited number of potentially exploitable services (Keromytis, Misra, and Rubenstein 2002). The set of hosts $H = V - R$ is partitioned into the set of legitimate users and the set of attackers. A leaf node denoted by S represents the victim server. Consider for example the network topology shown in figure 1. It consists of 20 nodes, these are, the victim server denoted by S, 13 routers denoted by $R1 - R13$ and six end hosts denoted by $H1 - H6$, which are traffic sources towards the server.

A legitimate user sends packets towards the server S at a rate r_l, and an attacker at a rate r_a. We assume that the attacker's rate is significantly higher than that of a legitimate user, that is, $r_a >> r_l$ [1]. This assumption is based on the rationale that if an attacker sends at a similar rate to a legitimate user, then the attacker must recruit a considerably larger number of agent hosts in order to launch an attack with a similar effect (Yau et al. 2005). A server S is assumed to be working normally if its load r_s is below a specified upper boundary U_s, that is, $r_s \leq U_s$ (this includes cases of weak or ineffective DDoS attacks). The rate r_l of a legitimate user is significantly lower than the upper boundary i.e. $r_l << U_s$, where U_s can be determined by observing how users normally access the server.

Design

Agent Selection Different selection methods exist, all based on the condition that ideally all of the aggregate traffic passes through the selected agents, otherwise the attack cannot be handled effectively, or even at all. For reasons of direct comparison we have chosen a similar selection method to the one used by Yau et al. (2005). Reinforcement learning agents are installed on locations that are determined by a positive integer k, and are given by $R(k) \subseteq R$. $R(k)$ is defined as the set of routers that are either k hops away from the server, or less than k hops away but are directly attached to a host. The effectiveness of throttling increases with an increasing value of k, although there is a limit since routers in $R(k)$ must belong to the same administrative domain (or collaborative domains). We emphasize that no topology modeling is required.

Consider for example the network topology shown in figure 1. Reinforcement learning agents are installed on the set $R(5)$, which consists of routers $R6, R7$ and $R10$. Router $R6$ is included in the set $R(5)$, although it is only 4 hops away from the server, because it is directly attached to the host $H1$.

State Space Each agent's state space consists of a single state feature, which is its average **load**. Recall that an aggregate is defined as the traffic that is directed towards the victim. The average load is defined as the aggregate traffic arrived at the router over the last T seconds, which we call the monitoring window. We set the time step of the reinforcement learning algorithm to be the same as the monitoring window size.

[1] Dropping traffic based on host addresses can be harmful because, as mentioned, hosts cannot be trusted.

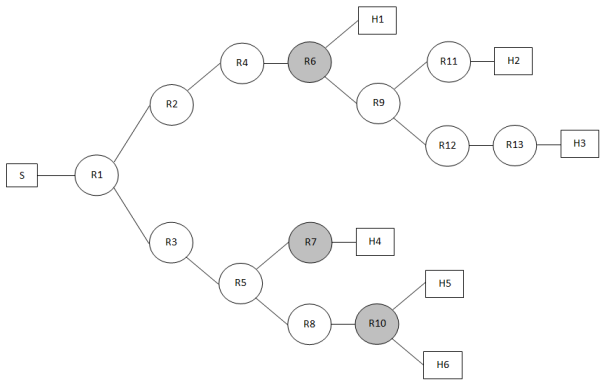

Figure 1: Network topology showing defensive routers

Action Space Each router applies throttling via probabilistic traffic dropping. For example action 0.4 means that the router will drop (approximately) 40% of its aggregate traffic towards the server, thus setting a throttle or allowing only 60% of it to reach the server. The action is applied throughout the monitor window.

Completely shutting off the aggregate traffic destined to the server by a router is prohibited, that is, the action 1.0 (which corresponds to 100% drop probability) is not included in the action space of any of the routers. The reason being that the incoming traffic likely contains some legitimate traffic as well, and therefore dropping all the incoming traffic facilitates the task of the attacker, which is to deny legitimate users access to the server.

Reward Function Each agent has the same reward function and therefore receives the same reward or punishment. Our proposed system has two important goals, which are directly encoded in the reward function.

The first goal is to keep the server operational, that is, to keep its load below the upper boundary U_s. When this is not the case, our system receives a punishment of -1. The second goal of our system is to allow as much legitimate traffic as possible to reach the server during a period of congestion. In this case, the system receives a reward of $L \in [0, 1]$, where L denotes the rate of the legitimate traffic that reached the server during a time step.

Simulation Results

To train and then evaluate our approach against others, we carried out realistic simulations using the ns-2 network simulator. ns-2 is the most widely used open source network simulator (Issariyakul and Hossain 2011). It is an advanced network simulator which offers, among others, abstraction, scenario generation and extensibility (Breslau et al. 2000). Abstraction refers to the simulation at different levels of granularity. Scenario generation refers to the creation of complex traffic patterns, topologies and dynamic events. Extensibility allows users to add new functionality to the simulator.

During the training of our system, we keep track of, and distinguish between legitimate and attack traffic (requirement for the reward function). However, we particularly emphasize that this is not the case during the evaluation of our system. The rationale behind this is that the defensive system can be trained in simulation, or in any other controlled environment (e.g. small-scale lab, wide-area testbed), where legitimate and DDoS packets are known a priori, and then deployed in a realistic network, where such knowledge is not available.

System Training

The network topology used for training our system is shown in figure 1. As a convention, bandwidth and traffic rates are measured in $Mbit/s$. The bottleneck link $S - R1$ has a limited bandwidth of $U_s = 8$, which constitutes the upper boundary for the server load. The rest of the links have an infinite bandwidth, and all links have a delay of $10ms$. Defensive agents are installed on the set $R(5)$, i.e. routers $R6, R7$ and $R10$.

Our network model is based on the model used by Yau et al. (2005). Legitimate users and attackers are evenly distributed, specifically each host is independently chosen to be a legitimate user with probability p and an attacker with probability $q = 1 - p$. We have chosen p and q to be 0.6 and 0.4 respectively. Each host sends fixed size UDP packets towards the server. Legitimate and attack packets are sent at constant rates, randomly and uniformly drawn from the range $[0, 1]$ and $[2.5, 6]$ respectively.

The Multiagent Router Throttling approach uses a linear decreasing ϵ-greedy exploration strategy with an initial $\epsilon = 0.4$ and the learning rate is set to $\alpha = 0.1$. We have used Tile Coding (Sutton and Barto 1998) for the representation of the state space and have discretized the action space into ten actions: $0.0, 0.1, ..., 0.9$ which correspond to $0\%, 10\%..., 90\%$ packet drop probabilities. We have also decided to use the popular SARSA (Sutton and Barto 1998) reinforcement learning algorithm, which was described earlier. We are only interested in immediate rewards, therefore we have set the discount factor to $\gamma = 0$; the resulting SARSA update rule is shown in formula 2:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r - Q(s, a) \right] \tag{2}$$

The system is trained for 62500 episodes, with no exploration after the 50000th episode. All traffic starts at time $t = 0s$ and each episode has a duration of $60s$. The monitoring window is set to $2s$. We particularly emphasize that training attempts to cover all instances of the network model, in other words, each episode is most likely to be different from each other. This is because at the start of each episode a new network instance is generated i.e. we re-choose the legitimate users, attackers and their rates according to the model. Consider the network topology shown in figure 1. In the first episode for example, there may be two attackers, let's say $H1$ and $H3$. In the second episode there may also be two attackers, but different ones, let's say $H2$ and $H5$. The third episode may have the same attackers as the first episode, but their sending rates are different. Therefore, we have trained our system in a very dynamic environment. It is very important to note that during the training phase, our

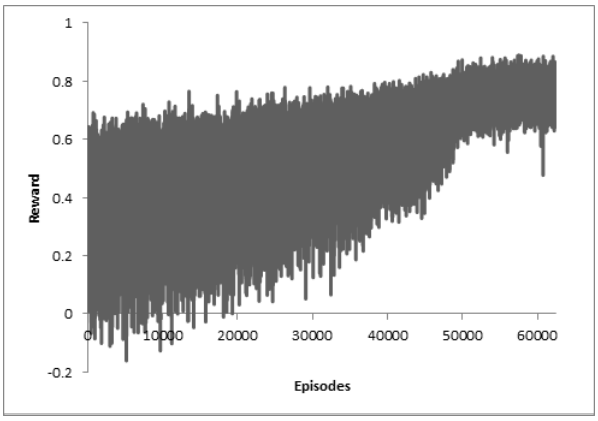

Figure 2: System training

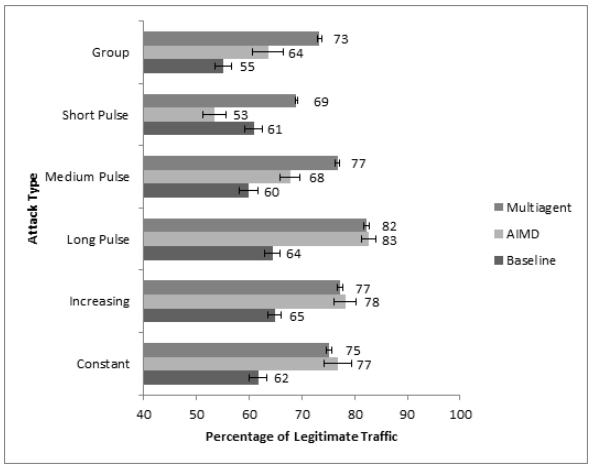

Figure 3: Evaluation results

system learns the "best" policy for all network instances, which can be different from the optimal policy of a single network instance.

We plot the last reward of each episode. Figure 2 presents the average rewards over 20 system trainings. It is clear that the system learns and improves over time until it finally converges. It is important to note that even if the corresponding "best" actions are performed in each episode, it is likely that different rewards will be yielded (and hence the shape of the graph).

Evaluation Results

We decided to evaluate our approach against the Baseline and the popular AIMD Router Throttling (Yau et al. 2005) approaches described earlier. These approaches use a lower boundary $L_s = 6$ and the same upper boundary $U_s = 8$ as our approach. Other control parameters are configured based on values and ranges recommended by their authors. As far as our approach is concerned, each reinforcement learning agent uses its policy learned during the system training.

For evaluation we randomly sampled 50 episodes (includes only effective DDoS attacks) each of a duration of $120s$. We use the same monitoring window size as previously, that is, $2s$. Legitimate traffic is started at $t = 0s$ and stopped at $t = 120s$. Attack traffic lasts for $100s$; it is started at $t = 10s$ and stopped at $t = 110s$. Evaluation is performed in six scenarios with different attack dynamics (Mirkovic and Reiher 2004):

- **Constant rate attack:** The maximum rate is achieved immediately when the attack is started.
- **Increasing rate attack:** The maximum rate is achieved gradually over $100s$.
- **Pulse attack:** The attack rate oscillates between the maximum rate and zero. The duration of the active and inactive period is the same and represented by T. We create 3 different attacks namely the long, medium and short pulse attacks which use a period of $T = 10s$, $T = 4s$ and $T = 2s$ respectively. Note that a short pulse has the same value as the system's monitoring window.
- **Group attack:** Attackers are split into two groups and each group performs simultaneously a different attack pattern. We have chosen the first group to perform a constant rate attack, and the second to perform a medium pulse attack. For this case, we retook a random sample of 50 episodes such that in each episode there are at least two attackers, one for each group.

We emphasize that these patterns have not been previously seen by our system during the training period. Furthermore, to make the evaluation more realistic, the legitimate packets' interarrival times follow a Pareto distribution. Again, this is different from what has been used for the training of our system.

Performance is measured as the percentage of legitimate traffic that reached the server. Figure 3 shows the average performance over the 20 policies learned during the system training, for the six types of attacks; error bars show the standard error around the average. In all scenarios, the Multiagent approach outperforms the Baseline Router Throttling approach (the differences are statistically significant). Similarly, in almost all scenarios, the AIMD approach also outperforms the Baseline approach. Furthermore, our approach has the same performance as the AIMD approach for the scenarios involving the constant rate, increasing rate and long pulse attacks (the differences are not statistically different). Most importantly, Multiagent Router Throttling outperforms the AIMD approach in the scenarios involving the medium pulse, short pulse and group attacks. Noteworthy is the fact that in the case of short pulse attacks, where their period is the same as the monitoring window, the AIMD approach performs worse than the Baseline.

Discussion

Advantages

Security The Baseline and AIMD throttling approaches are server-initiated, that is, the server controls and sends the throttle signals to the upstream routers. However, they are based on the assumption that either the server and the net-

work infrastructure remain operational during the DDoS attack, or that a helper machine is introduced to deal with the throttle signaling (Yau, Liang, and Lui 2001). The first assumption is unlikely to be valid in a real-world scenario. As far as the second assumption is concerned, the helper machine can also become a target of the attack.

In essence, the problem arises because existing approaches are server-initiated, in other words, they have a single point of failure. One of the novel characteristics of Multiagent Router Throttling is its decentralized architecture and response to the DDoS threat. Each agent independently throttles traffic, thus our approach is more secure and does not have a single point of failure.

Reliability and Cost-effectiveness Existing throttling approaches require an efficient and reliable communication between the server and the upstream routers. Specifically, they require throttle message authentication, priority transmission for throttle messages and packet retransmission in case of packet loss (Yau, Liang, and Lui 2001).

Our approach requires no communication at all between routers due to its decentralized architecture and response. This makes our approach not only more secure and reliable, but more cost effective since authentication, priority transmission and packet retransmission are not needed.

Adaptability and Performance Existing throttling approaches can suffer from stability, and potential convergence problems because of system oscillations in order to settle the server load to a desired level within the lower L_s and upper U_s boundaries. Performing throttling becomes challenging as the range $[L_s, U_s]$ becomes smaller. Even if convergence is obtained, oscillations can cause an increase in the time required for the server load to converge to the desired level.

In contrast, the Multiagent Router Throttling approach learns the router throttles, therefore it does not require system oscillations, and as a result it does not suffer from stability issues. Our approach is highly adaptable and this is exactly the reason it outperforms the AIMD algorithm in the highly dynamic scenarios that involve the medium and short pulse attacks and the group attack.

Limitations and Future Work

"Meek" Attackers Our approach does not consider the case of "meek" attackers, i.e. where an attacker's sending rate is similar to the rate of a legitimate user. As already discussed, this requires that an attacker compromises and recruits a really high number of host machines. Effectively tackling this problem would require the enrichment of the state space of an agent, that is, to introduce more statistical features other than the local router load. This is necessary because in the case of "meek" attackers, the system cannot differentiate between legitimate and attack traffic by just taking into account router loads.

Scalability Scalability is an important challenge which we plan to study in our future work by examining different mechanisms. Specifically, we will investigate how Multiagent Router Throttling can be extended to incorporate tens of learning agents.

Network Model The performance of Multiagent Router Throttling depends on an accurate network model. As already discussed, during the training phase, the reward function requires to distinguish between legitimate and attack traffic. However, the network's behavior, especially the attacker's, is unpredictable; capturing this into a model is very difficult. Unlike non-learning approaches, one of the core advantages of reinforcement learning is its capability for online learning. Online learning would allow the system to continue learning without a model. Further investigation is required to examine the potential of online learning.

Acknowledgments

Thanks go to Sam Devlin and Gareth Andrew Lloyd for their time and input to this work.

References

Bertsekas, D. P., and Tsitsiklis, J. N. 1996. *Neuro-Dynamic Programming*. Athena Scientific.

Breslau, L.; Estrin, D.; Fall, K.; Floyd, S.; Heidemann, J.; Helmy, A.; Huang, P.; Mccanne, S.; Varadhan, K.; Xu, Y.; and Yu, H. 2000. Advances in network simulation. *Computer* 33(5):59–67.

Claus, C., and Boutilier, C. 1998. The dynamics of reinforcement learning in cooperative multiagent systems. In *AAAI/IAAI*, 746–752.

Douligeris, C., and Mitrokotsa, A. 2004. Ddos attacks and defense mechanisms: classification and state-of-the-art. *Computer Networks* 44(5):643–666.

Issariyakul, T., and Hossain, E. 2011. *Introduction to Network Simulator NS2*. Springer, 2nd edition.

Keromytis, A. D.; Misra, V.; and Rubenstein, D. 2002. Sos: secure overlay services. In *SIGCOMM*, 61–72.

Mahajan, R.; Bellovin, S. M.; Floyd, S.; Ioannidis, J.; Paxson, V.; and Shenker, S. 2002. Controlling high bandwidth aggregates in the network. *Computer Communication Review* 32(3):62–73.

Mirkovic, J., and Reiher, P. L. 2004. A taxonomy of ddos attack and ddos defense mechanisms. *Computer Communication Review* 34(2):39–53.

Mirkovic, J.; Robinson, M.; and Reiher, P. L. 2003. Alliance formation for ddos defense. In *NSPW*, 11–18.

Sutton, R. S., and Barto, A. G. 1998. *Introduction to Reinforcement Learning*. MIT Press Cambridge, MA, USA.

Yau, D. K. Y.; Lui, J. C. S.; Liang, F.; and Yam, Y. 2005. Defending against distributed denial-of-service attacks with max-min fair server-centric router throttles. In *IEEE/ACM Transactions on Networking*, 29–42.

Yau, D. K. Y.; Liang, F.; and Lui, J. C. S. 2001. On defending against distributed denial-of-service attacks with server-centric router throttles. Technical Report CSD TR #01-008, Department of Computer Science, Purdue University.