Reinforcement Learning for Vulnerability Assessment in Peer-to-Peer Networks

Scott Dejmal and **Alan Fern** and **Thinh Nguyen**
School of Electrical Engineering and Computer Science
Oregon State University
Corvallis, OR 97330
Email: {dejmal, afern, thinhq}@eecs.oregonstate.edu

Abstract

Proactive assessment of computer-network vulnerability to unknown future attacks is an important but unsolved computer security problem where AI techniques have significant impact potential. In this paper, we investigate the use of reinforcement learning (RL) for proactive security in the context of denial-of-service (DoS) attacks in peer-to-peer (P2P) networks. Such a tool would be useful for network administrators and designers to assess and compare the vulnerability of various network configurations and security measures in order to optimize those choices for maximum security. We first discuss the various dimensions of the problem and how to formulate it as RL. Next we introduce compact parametric policy representations for both single attacker and botnets and derive a policy-gradient RL algorithm. We evaluate these algorithms under a variety of network configurations that employ recent fair-use DoS security mechanisms. The results show that our RL-based approach is able to significantly outperform a number of heuristic strategies in terms of the severity of the attacks discovered. The results also suggest some possible network design lessons for reducing the attack potential of an intelligent attacker.

Introduction

Vulnerability assessment (VA) is an area of computer security that involves proactively discovering security weaknesses in a computer network or system before those weaknesses are exploited by an attacker. As an example, consider designing a security measure for a peer-to-peer (P2P) network to prevent malicious flooding by single attackers or distributed botnets. Such systems are highly susceptible to such attacks and it is impossible to foresee, for a particular security measure, the range of weaknesses that might be exploited by future attacks. One way to help evaluate a security measure is to consult an independent security expert (or team) to try to hypothesize potential weaknesses. This approach, however, is expensive, and often fails to discover many vulnerabilities. A critical direction in computer security is to develop automated VA techniques that can identify weaknesses in a proposed network configuration and security measure.

Most automated VA software has been developed to prevent intrusions and utilizes a library-based approach, where a network is tested against a library of known vulnerabilities or attacks (e.g. buffer overflow). This approach, however, does not prevent the exploitation of vulnerabilities outside of the library (perhaps specific to a given network). Thus, importantly, the approach can effectively prevent attacks such as those arising from automated attack software, but is still susceptible to more sophisticated, knowledgeable, and determined attackers. In addition, in VA domains such as P2P networking, it is difficult and unnatural to produce a sizeable library. What type of library could be used to represent the wide range of possible distributed denial-of-service (DDoS) attacks?

In this paper, we advocate for a complementary approach to library-based vulnerability assessment where AI techniques are used to discover novel weaknesses in a network or system. Once discovered the weaknesses can then be used to guide an administrator toward a more secure network design. We believe that AI techniques offer significant impact potential in this wide application domain and our work in this paper is a first step toward demonstrating this promise.

We focus on the problem of learning to optimize DDoS attacks on a P2P network to achieve maximal damage. Given such a tool, a network designer or administrator could evaluate various combinations of network configurations and security measures in simulation with respect to their susceptibility to DDoS-style attacks. Currently it can be quite difficult to arrive at such evaluations due to the huge space of possible attacks. For example, in recent work on passive P2P network security mechanisms (Daswani & Garcia-Molina 2002; KaZaA 2008), comparative experimental results are only provided for small networks and somewhat artificial configurations where humans are able to determine the optimal DDoS attack. From these results it is difficult to know the relative performance of these mechanisms against an intelligent adversary in more realistic networks. AI techniques provide the opportunity to obtain such comparisons.

In this work, we show how to cast our problem of proactive VA in P2P networks as a reinforcement learning (RL) problem. RL is the general problem of learning control policies based on interactions with an environment and observations of reward signals that result from the interaction. In our application, the environment will correspond to a specific computer network, where perhaps a candidate security measure is installed. The reward signal will provide positive reward for activity that is detrimental to the network, and negative reward for activity that is detected as malicious by the security measure. The learned control policies will select actions for a botnet of one or more malicious nodes in order to maximize the reward, or damage to the network. The discovery of such a policy indicates an effective way for one or more attackers to compromise the network, which can then be analyzed and used to suggest an improved network structure or security measure.

There has been previous work on applying RL techniques to computer networking including examples such as network routing (Littman & Boyan 1993; Tao, Baxter, & Weaver 2001), mobilized ad-hoc network policies (Chang, Ho, & Kaelbling 2004), and peer selection in P2P networks (Bernstein *et al.* 2003). These successes show that RL is capable of improving the performance of computer networks. However, we are unaware of any work that applies RL to proactive network security, which has the opposite goal of maximally hurting network performance.

We describe an RL solution based on online policy-gradient optimization. Our experimental results across a variety of simulated P2P networks indicate that this approach is able to significantly improve on some natural heuristic solutions in terms of the amount of network damage. Furthermore, our initial results suggest some high-level features of computer networks that are particularly exploitable by an intelligent attacker compared to a naive attacker. These features can potentially be used as guiding principles by network designers.

In what follows, we first describe our problem of proactive vulnerability assessment in P2P networks in more detail. Next, we describe our RL formulation of this problem followed by a derivation of a policy-gradient RL algorithm. Next, we present our experimental results followed by a summary and discussion of future work.

Problem Description

Since there are many varieties of P2P networks and defense mechanisms, we formulate the problem in terms of an abstract model that captures the key properties of the P2P DDoS attack problem. A P2P network is composed of a set of nodes and a set of files that are distributed across the nodes. Each node has a set of peers, typically changing over time, which can be directly queried when content is desired. When a node receives a query from a peer, if the node is below its query-processing capacity it will either return the requested content if it is present at the node, or otherwise forward the query to its own set of peers. This process of forwarding a query when it cannot be served continues for at most a maximum number of hops as specified by the P2P network configuration. In cases where a node is above its query processing capacity and receives a query, it can either choose to process the query as described above or to drop the query according to some policy.

In our work, we will consider a common abstraction of P2P networks in terms of supernodes (Daswani & Garcia-Molina 2002). P2P networks are often organized around a set of supernodes, which act as query hubs for normal nodes. The normal network nodes are partitioned and the partitions are associated with distinct supernodes which process the queries originating from the partition. The above query forwarding process is then conducted at the level of supernodes. In this model, we can abstract away from normal nodes and focus on the supernode P2P network, where each supernode is viewed as originating a large number of queries, and each will often be operating at close to maximum query processing capacity. In the remainder of this paper we will adopt this model and simply use the term node rather than supernode.

In our P2P network model, query processing is considered to be the most constrained resource and the key to mounting a successful DDoS attack is to inject malicious queries into the network in such a way that they are processed by the maximum number of (super) nodes, where here a malicious query is simply any query that originates from an attacker. The key for a network to mitigate a DDoS attack is to drop as many malicious queries as possible, with different defense mechanisms varying according to how they make dropping decisions.

One possible dropping mechanism is to actively classify incoming queries as either malicious or normal and then drop accordingly. However, it is extremely difficult to establish trust in a P2P network, and active classification of query-generator intent appears to be very difficult against a botnet with many physical addresses. Thus, most DDoS defense mechanisms for P2P networks, and the ones we consider in this work, are passive, fair-use schemes rather than active. As one example, a security mechanism might allocate its query-processing resources proportionally based on the number of its immediate neighbors rather than on the number of incoming queries from those neighbors. Even this simple defense mechanism represents a considerable improvement over the policy of allocating based on the proportion of raw incoming queries (Daswani & Garcia-Molina 2002). In particular, it significantly diminishes the effects of a naive attack that bombards a single high degree node. However, given two such mechanisms it is difficult to determine which will be more robust to intelligent, sometimes non-obvious attack strategies.

Our objective is to proactively assess the vulnerability of a particular P2P network under a particular set of security mechanisms. More precisely, we will be provided with a simulation of a P2P network, which in practice would reflect properties of the real network of interest, including its defense mechanisms, general structure, and typical traffic patterns. In this simulation we will be given control of a botnet of one or more malicious nodes, each with a fixed query bandwidth. Our objective is to determine a control policy for the botnet that maximizes the number of malicious queries that are processed by non-malicious nodes in the P2P network. Given a way to generate such policies, different network structures and measures can be compared with respect to the susceptibility to intelligent attacks.

Our goal, in this work, will not be to exploit and attack application or protocol bugs, but rather to measure the limits of the resiliency of a passive defense mechanism under reasonable observability conditions for an attacker. In this sense, our DDoS attackers will be restricted to behave very much within the boundaries of normal P2P network use, but with very different intentions compared to non-malicious nodes. Thus, the basic actions available to our malicious nodes will be to send a malicious query to various nodes across the network and it can choose its actions based only on information that would be available to it under reasonable assumptions about what can be observed by an attacker operating in the actual network.

It can be easily shown that the problem of finding an optimal attack policy is NP-Hard even for very simple passive defense mechanisms. This is the case even under idealized conditions where the network is deterministic and we are provided with complete knowledge of the network structure, traffic, and security mechanisms. Furthermore, the problem can be shown to be APX-complete, which indicates that there is likely no arbitrarily precise constant factor approximation algorithm. These intractability results lead us to consider a heuristic approach to solving the problem via the application of RL.

Reinforcement Learning Formulation

RL Background. RL (Sutton & Barto 1998) studies algorithms for learning to effectively control a system by interacting with the system and observing the resulting rewards, or reinforcements. Thus, RL techniques are a natural choice for problems where it is difficult to precisely specify an explicit software solution, but where it is easy to provide a reward signal, which is exactly the case in our P2P DDoS attack problem. Here we will formulate the RL problem in the framework of Partially Observable Markov Decision Processes (POMDPs). POMDPs are commonly used to describe dynamic systems that can be controlled and observed such as computer networks. A POMDP is defined by a set of states S, representing the possible states of the system, a set of observations O, representing the observable parts of the system, a set of control actions A, a transition function T, a reward function R, and an observation distribution D. The Markovian transition function specifies the dynamics of the system, and gives the probability $T(s, a, s')$ of transitioning to state s' after taking action a in state s. The reward function assigns real numbers to states, so that $R(s)$ represents the immediate reward of being in state s. The observation distribution gives the probability $D(o, s)$ of observing observation o when in state s.

At any moment the POMDP is in a particular system state and when an action is selected by the controller the system transitions to a new state according to T and then generates a reward according to R and an observation according to D which can be observed by the controller. Note that the underlying system's state may not be directly observable by the controller. Formally, a controller is represented as a policy π which is a possibly stochastic function from sequences of past observations to actions. Each policy π defines a distribution over infinite sequences of rewards when run from a fixed initial state of a POMDP, and we denote the expected time-averaged reward to be $\rho(\pi)$. Given a POMDP, the goal is to return a policy that maximizes $\rho(\pi)$.

In RL, we are not provided with the POMDP, but rather are allowed to select action in the POMDP and observe the resulting reward and observation sequences. Based on this interaction the general goal of RL is to find a policy that maximizes $\rho(\pi)$ as quickly as possible. Typically maximizing $\rho(\pi)$ in a reasonable time frame is not practical and we instead will focus on finding a good, but perhaps suboptimal policy, quickly.

There are many ways to formulate our DDoS attack problem as RL. This primarily involves selecting the reward function, the action space, and the observation space. Below we describe the formulation used in this paper. We first focus on the case of a single malicious node and then describe the formulation for multi-node botnets.

DDoS Reward Function. We will define the reward signal at any time step to be equal to the number of times that any malicious query was processed by any non-malicious node in the network during the execution of the previous action. Note that this reward signal will measure the damage caused not just by the most recent action, but also by previous actions. Maximizing the expected average reward then corresponds to maximizing the average number of malicious queries processed by the network.

Note that the use of this reward function assumes that there is some way for our malicious node(s) to observe it in the network. In cases where we are applying RL to simulated networks this information is readily available. However, we also argue that the reward information could likely be approximately observed in a real network, and thus it is reasonable to assume that a real attacker might have access to the same information. There are several ways that this could be accomplished. For example, using back channel communications, malicious nodes in a botnet could communicate their application layer unique IDs to each other. Each query will have such a return address and each malicious node can estimate local damage by simply keeping track of the ratio of bad to total queries that arrive. Malicious nodes can periodically communicate this information to each other, either directly or via a centralized botnet controller, and read back the aggregated reward signal.

DDoS Action Space. Next, we must define the action space. One possibility would be to have an action for each non-malicious node that when taken sends a malicious packet to that node. While this is a reasonable choice, this action space results in a difficult RL problem. To see this, recall that in an RL scenario after each action is selected the agent receives an immediate reward and an observation. The fundamental problem in RL is one of credit assignment, where it is necessary to learn how to assign credit for the most recent reward to actions selected in the past. In this case, where the actions correspond to sending single malicious queries, the effect of any individual action is quite small and the reward integrates the effect of a long string of prior actions, leading to a very ambiguous credit assignment problem. The result will be very long learning times.

With the above motivation, we consider the use of a richer action space for P2P DDoS attacks where the credit assignment problem is easier. Specifically, the action space will correspond to the possible ways of allocating B malicious queries across nodes in the network. That is, each individidual action a can be viewed as an allocation vector with each component a_i equal to the number of malicious queries to be sent to node i, where $\sum_i a_i = B$. Upon selecting an action a the malicious node then proceeds to send all of the malicious queries as dictated by the allocation vector and then receives a reward after this step has completed which is equal to the total number of bad queries that were processed by the nodes in the network during the time the action was executing.

The allocation vector actions can be viewed as coarsening the time scale of basic actions and increasing the direct influence that single actions have on the reward signal. This results in an easier learning problem from a credit assignment point of view. However, now rather than having a small action space, one action per network node, there are an exponentially large set of actions: one for each possible allocation vector. While many RL algorithms are not able to cope with such large action spaces, we show in the next section a policy representation and learning algorithm that can.

DDoS Observation Space. Finally, for simplicity, in our current experiments, we use an empty observation space. That is, the malicious nodes do not observe anything about the network other than the reward signal. Our experimental results show that very good results can be achieved just by observing rewards. We note, however, that the learning algorithm we describe allows for arbitrary observation spaces and this is an interesting direction for future work.

Multi-Node Botnets. The above discussion was for a single malicious node. The formulation is similar in the case of a multi-node botnet. The action space for the botnet is simply the cross-product of the action spaces for the individual nodes. That is, each botnet action is a sequence of allocation vectors: one for each malicious node. An important issue that arises in the extension to botnets is that of centralized versus distributed control. In the centralized case, there is a central controller or policy that issues actions to each of the malicious nodes. In this case, it is theoretically possible for the centralized controller to explicitly coordinate the actions of the individual node in an optimal way.

Rather, in the distributed case, there is no such central controller and the nodes execute localized policies, possibly in collaboration with a subset of other nodes in the botnet. This provides only limited coordination among the nodes in the botnet, possibly leading to sub-optimal results. The advantage of the distributed setting is that the demands on communication among the nodes are lessened. However, it is relatively easy to justify the availability of back channel communication among the botnet since we are considering an application layer attack, and the peer to peer software operates within an overlay network allowing malicious nodes to communicate with each other via other ports, either directly or through a centralized controller. Thus, in our current work we assume a centralized control setting.

Policy Gradient Reinforcement Learning

In this section we describe a specific policy representation for DoS attacks and a corresponding algorithm for learning those policies based on policy-gradient RL.

The key idea behind policy-gradient RL is to describe a parametric class of policies with parameters θ and to then perform some form of online gradient descent in the space of policy parameters in order to optimize the expected time-averaged reward. More formally, assume a class of parametric stochastic policies such that $\pi(a \mid \theta, o)$ gives the probability that the policy selects action a given previous observation sequence o and current parameters θ. Later in this section we will define such a policy class for our application. We will denote the expected time-averaged reward of the policy with parameters θ by $\rho(\theta)$ and our goal is to adjust the parameters in order to maximize this function.

Typically the function $\rho(\theta)$ will be non-convex with many local optima and thus the typical goal is to develop learning algorithms that will converge to one of those local optima. If it were possible to compute the gradient of the expected reward $\nabla_\theta \rho(\theta)$ then one could simply perform gradient descent by iteratively moving the parameter vector in the direction of the gradient until arriving at a local optima. Unfortunately, it is generally not feasible to derive a useful closed form of this gradient and thus we must estimate it based on our interaction with the environment. One algorithm for doing this is OLPOMDP (Baxter & Bartlett 2000) which carries out a form of stochastic gradient descent, in which a noisy estimate of the gradient is computed at each time step and used to update the parameter vector. Algorithm 1 gives pseudo-code for OLPOMDP.

Each time through the main loop, the algorithm first reads the current observation from the environment and then samples an action from the policy under the current parameters. The action is then executed, resulting in an observed reward. Next a vector e, called the eligibility trace, is updated, which stores a discounted sum of gradient directions in which previous actions can be made more likely, where β is the discount factor. Finally, the parameter vector is updated in the direction of the eligibility vector scaled by the most recent reward received. Here α is a learning rate, typically a small constant less than one. Intuitively the effect of the eligibility trace is to move the parameter vector in a direction that makes more recent actions more likely if the reward is positive and less likely if the reward is negative, with the magnitude of the movement related to the magnitude of the reward.

Algorithm 1: OLPOMPD
```
1: $e = \theta = 0$ // initialize eligibility trace and parameters
2: **while** continue? **do**
3:     read current observation $o$
4:     sample action $a$ from $\pi(a \mid o, theta)$
5:     execute action $a$ and observe reward $r$
5:     $e \leftarrow \beta e + \nabla_\theta \log(\pi(a \mid o, \theta))$
6:     $\theta \leftarrow \theta + \alpha \, r \, e$
```

The main requirement of this algorithm is that we must be able to sample actions from the policy and compute the gradient $\nabla_\theta \log\left(\pi(a \mid o, \theta)\right)$ of the logarithm of the probability of the policy selecting an action given the current parameters and observations. Below we describe a policy representation where these requirements are met for our exponentially large action space described in the previous section.

Policy Representations. Recall that for our DoS problem formulation we have an exponentially large action space consisting of all possible allocation vectors over B queries and no observations. Thus, our policy $\pi\left(a|\,\theta\right)$ is a distribution over the space of allocation vectors. A simple way to define such a policy is by defining a multinomial distribution over the network nodes and then making B independent draws from the distribution with replacement. The i'th component of the allocation vector is then equal to the number of times node i was drawn. With this approach we can parameterize our policy in terms of the parameters of a multinomial distribution. We use the following, common exponential form of a multinomial distribution with parameters θ,

$$P_\theta(i) = \frac{e^{\theta_i}}{Z\left(\theta\right)}, \quad Z\left(\theta\right) = \sum_j e^{\theta_j}$$

where $P_\theta(i)$ is the probability that the distribution assigns to node i and the relative value of θ_i controls this probability. Note that this specifies a well defined distribution over nodes for any value of θ. Given this form and the fact that our policy makes independent draws from this distribution to generate allocation vectors, we get the following form for the probability assigned to any allocation vector a,

$$\pi\left(a\,|\theta\right) = \frac{\exp\left(\sum_i a_i\theta_i\right)}{\left(Z\left(\theta\right)\right)^B}$$

where a_i is the number of queries assigned to node i. Given this form, the gradient of the logarithm of this probability is given by

$$\frac{\partial \log\left(\pi\left(a\,|\theta\right)\right)}{\partial\theta_i} = a_i - B\cdot P_\theta(i)$$

From this we see that the gradient component corresponding to node i is simply the number of queries allocated to node i minus the expected number of queries to be allocated to node i given the current parameter vector. Given this closed form for the gradient, it is straightforward to implement OLPOMDP.

The above policy representation is for a single malicious node. In the case of multi-node botnets, we consider a particularly simple and restricted approach in this paper. We define a multi-node policy by simply learning a single policy of the above form that is shared by all of the nodes in the botnet. This is far from optimal as all of the nodes are restricted to the same policy, however, it is an extremely simple approach that requires little modification to the basic single agent approach. In the future we plan to study more sophisticated multi-node policies, for example allowing each node to learn a distinct set of independent policy parameters as in (Peshkin *et al.* 2000) or more sophisticated representations that allow for explicit coordination among the agent policies such as (Guestrin, Lagoudakis, & Parr 2002).

DoS Simulation Environment

Simulation was performed in a synchronous discrete time network simulator which we designed for this research. P2P networks are modeled as graphs with vertices representing peers and edges representing immediate neighbors. Edges are bidirectional, and because we are only considering constraints at the nodes, no additional attributes are associated with the links themselves. Each peer has several attributes: a query processing capacity, damage factor, base query generation rate, allocation and drop strategies, and the aforementioned set of links to its immediate neighbors.

During a single time step, each node gets a chance to generate and forward its own queries (subject to its background query generation rate parameter) and to process its incoming queue. Processing the queue consists of examining each query which was enqueued in the previous time step and deciding whether to accept that query or to drop it. If it accepts the query, it will remember it (store its globally unique identifier in a hash table), and forward it to all its neighbors (enqueue it in their incoming queue, to be processed in the next time step). Each time a query is forwarded, its time-to-live (TTL) is decremented. Queries start with a TTL of 7 and when they reach 0, they are dropped regardless of other factors. TTL of 7 is the default value of many Gnutella-based P2P networks.

There are two points that may be worth clarifying here. First, timestamps are added during the enqueue process which dictates when the associated peer will process them — between the current step and the next step. This maintains the behavior of independent processors acting synchronously, even in a single threaded application. It also means the order in which nodes are allowed to act is not relevant. Second, it may seem that a lot of work is being done at each time step; it is very possible that this amount of work would not be viable for a lower level bandwidth attack, but we are considering an application layer attack where query acceptance may incur a resource requirement several orders of magnitude greater.

The damage factor describes the value of having a query–good or bad–processed by this node. The allocation and drop strategies are used to determine how many queries to accept from each neighbor, and once this is determined, which particular queries to drop. In these experiments, both remained constant. The allocation strategy used was "fractional", and queries forwarded were those with the highest time-to-live. The fractional allocation strategy is a fairly straightforward fair-use scheme with the following recursive algorithm. If there is some remaining query processing capacity, it divides that remaining capacity by the number of immediate neighbors, accepts up to that number of queries from each neighbor, and subtracts the number of queries it has accepted from its remaining capacity. Fractional queries are allocated to neighbors arbitrarily. This process repeats until either all queries have been accepted, or all capacity has been consumed. This process determines the number of queries to accept from each neighbor. The particular queries are chosen based on their remaining TTL. In these experiments, those with the highest remaining TTL are forwarded. This fair-use scheme defines a very basic DDoS defense mechanism, but is significantly better than allocating resources proportional to the number of incoming queries. Moreover, it is simpler and immune to false positives, compared to an active mechanism.

Experimental Evaluation

In the following experiments we compared our OLPOMDP approach against several other heuristic approaches of varying complexity. Below we describe each of the heuristic policies we consider, followed by the experimental setup, and the results.

Heuristic Policies

Random. The random policy simply draws its allocation vector from a uniform random distribution where each peer is equally likely.

Uniform. The uniform policy allocates the same number of queries to each peer in the network. Fractional queries are assigned arbitrarily, so in any given time step, the allocation is only approximately equal.

High Degree. The high degree heuristic is commonly used in social networking research when studying spread-of-influence phenomena. It allocates queries proportionally to the fan out at each particular node. Its critical weakness in the DoS setting is that it does not account for redundancy. In our model, a peer will not forward a query that it has previously seen. Thus, often the damage caused by high degree nodes that are topologically adjacent will *not* be strictly additive since there will be a high level of query redundancy in their neighbors.

Greedy Search. The greedy algorithm is a simulation-based approach that requires complete knowledge of the network in order to simulate it. In order to select allocation vectors, the greedy algorithm performs offline simulations starting with a query budget of one, allocating it to each node in turn and settling on the best single query allocation. It continues to add one query at a time in a greedy manner until it has used its full allocation budget. At this point it sends queries in the real network according to the final allocation vector.

This approach has several practical problems. First, it is not clear that a botnet will be able to infer the full network topology and be able to form accurate simulations of a network which also requires inferring many other network parameters such as capacities, cutoff thresholds, etc. Also the greedy mechanism is quite computationally expensive due to the large number of network simulations that must be run, which makes real-time use in a network problematic.

Although greedy search is not a particularly realistic attack mechanism, it provides a useful point of reference as it provides a potentially optimistic estimate of the maximum damage an attacker might inflict given oracle knowledge of the network and significant amounts of processing time.

Beam Search. Beam search is a simple generalization of the above greedy approach that maintains a beam of the b best allocation vectors during the search and outputs the best one in the end. Greedy search is a special case where only the single best allocation vector is maintained. This approach requires about b times as much memory as and time as greedy search, but will typically find better solutions. Like the greedy algorithm, it needs full knowledge of all relevant network parameters. In all of our experiments we used a beam width of $b = 3$.

Evaluation Method.

Our OLPOMDP approach has two parameters: α, the learning rate, and β, the discount factor. After some initial experiments on validation networks we found that $\alpha = 0.01$ and $\beta = 0.8$ worked well and used these values for all of the experiments reported in this section. The results were relatively insensitive to small variations around these values.

We ran OLPOMDP on a variety of networks described below. For each network we conducted 10 independent trials for a 4 node botnet where each node had a query budget of 7 per time step. Each trial consisted of 5000 decision epochs and the average damage during the last 100 epochs of each trial was recorded in Table 1 under the column labeled RL. We also ran 10 independent evaluations of the heuristic policies described above and recorded the average damage they achieved in the appropriately labeled columns of Table 1. Here damage is measured as the total number of bad packets processed by the network divided by the total number of packets processed, and hence always lies in the interval $[0, 1]$.

Each row of Table 1 corresponds to a particular network. We considered three network topologies. The first two were generated by the BRITE (BRITE 2008) topology generator in two different schemas: Barabási (scale free) and Waxman (random). Barabási networks are labeled with "BA" and Waxman with "WAX". The number of supernodes in each network ranged from size 10 to 40 as indicated in the table. We also included a 20 node grid topology "20Grid" where the nodes were arranged in a 4 x 5 lattice.

	Beam(3)	Greedy	High Degree	Random	Uniform	RL
10BA	0.64	0.62	0.52	0.52	0.54	0.57
10WAX	0.51	0.52	0.44	0.39	0.45	0.44
15BA	0.62	0.58	0.48	0.47	0.48	0.54
15WAX	0.43	0.42	0.36	0.35	0.36	0.37
20BA	0.43	0.42	0.33	0.32	0.33	0.41
20WAX	0.34	0.34	0.27	0.25	0.26	0.28
30BA	0.47	0.47	0.28	0.29	0.28	0.41
30WAX	0.40	0.38	0.20	0.20	0.22	0.33
40BA	0.45	0.45	0.25	0.25	0.26	0.40
40WAX	0.35	0.33	0.17	0.19	0.18	0.28
20Grid	0.42	0.42	0.26	0.23	0.28	0.36

Table 1: Damage Achieved by 4 Node Botnet

Results

Learning Curves. Figure 1 shows the averaged learning curve for the 40 node Barabási Network, which is representative of other results. Generally we found that larger networks have much noisier learning curves, but offer more potential for improvement over random and uniform strategies. For this curve and all other cases, the majority of learning

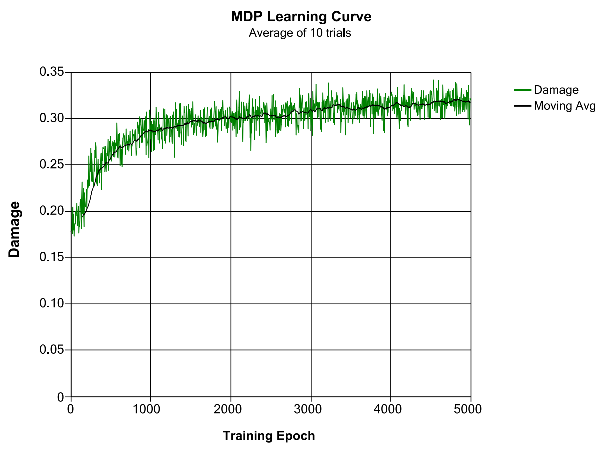

Figure 1: Learning Curve - 40 Node Barabási Network

was achieved within 2000 epochs, with larger networks continuing to improve by a small amount over the next several thousand epochs. The 20 node lattice network had a particularly steep learning curve, improving its strategy by roughly 60% in the first 1000 training epochs. We speculate that the steep rise in damage is due to the highly structured nature of this network which makes it easier to find exploitable attack patterns.

Comparing RL to Naive Policies. Random and Uniform are both naive policies in the sense that they do not require any knowledge of the network structure or parameters in order to make their decisions, as is the case for our RL approach. We see that in many cases RL is able to inflict a significantly larger amount of damage on the networks than either naive policy. For the smaller networks we see that RL does not improve much over the naive strategies, but it is never significantly worse.

The High Degree heuristic is less naive as it requires knowledge of the network structure in terms of the out degree of each node. Despite the use of this knowledge we see that it rarely does significantly better than uniform or random and hence its relationship to RL is the same as those naive policies. This shows that RL was able to significantly outperform a common heuristic used in the study of spread-of-influence phenomena, while using no knowledge of the network structure.

Comparing RL to Oracle Policies. In all cases the oracle policies are able to outperform our RL approach. In many cases the increase in performance is significant. This indicates that either: 1) using oracle knowledge of the network increases the theoretical limit on the average damage a policy can possible inflict, or 2) the utility of oracle knowledge is less than observed and there is significant room to improve our RL algorithm. We believe that a combination of these two points is actually the case, since our RL algorithm is relatively simple and can likely be improved in a number of ways as discussed in the future work section. It is important to note, however, that even our simple RL approach is able to make up a large part of the performance difference between the naive approaches and oracle approaches, while using only information about the damage, which is quite plausible to obtain in a real network.

Possible Lessons for Network Design

While the primary goal of our experiments was to demonstrate that RL could improve over naive attacks in a variety of networks, our initial results suggest some basic network design principles. The first lesson suggested by the results is that more regular network structures are more susceptible to intelligent attacks. In particular, there appears to be a bigger difference between our intelligent RL-based attacks and the naive attacks for Barabási (scale free) and regular grids than for the random Waxman topologies. Furthermore this difference seems to increase as the total number of supernodes increases. The second lesson suggested by the results is that as the number of supernodes in a network grows, the disparity between naive and intelligent attackers seems to broaden.

In these preliminary experiments, we did not scale the capacity of attackers with the size of the network. Since damage is a normalized quantity relative to network size (ratio of bad/total packets), we hypothesize that when resources are highly constrained, the attacker stands to gain the most by allocating those resources intelligently. This seems to be a realistic scenario; in real-world applications, the ratio of malicious to total nodes is likely to be quite low. In these cases, we expect to see an even greater relative improvement in damage of intelligent agents versus naive attackers.

It is important to note that the above observations are only speculative at this time. An important piece of follow-up work is to conduct a much broader and more thorough evaluation of a variety of network structures, sizes, and traffic conditions in order to verify the above observations and identify additional design principles.

Toward Applicability

The principal goal of this initial study was to show that with a plausible level of knowledge, intelligent agents can use reinforcement learning to increase the effectiveness of attacks. We have demonstrated positive results and are now interested in moving them toward real application software. Until now, developers of new defense mechanisms often demonstrate their effectiveness against naive attacks; however, our work shows that there may be plausible attacks which are significantly more damaging. We suggest this framework as a tool which can be used to evaluate the resiliency of new protocols against more sophisticated attacks than those typically presented. In this scenario, design choices for new protocols can be compared and tradeoffs can be evaluated. Thus, we expect that one customer of our software will be security researchers who will use it to provide more substantial evaluations of new security ideas.

Another customer will be network designers who aim to select and employ existing security protocols and design network topologies. For example, a protocol may require setting connection acceptance ratio, trust threshold, and other tunable parameters. In addition, in certain applications the network designer may have some control over the topology and number of supernodes. Given our software all of these choices can be evaluated more objectively against plausible intelligent attacks, which are not generally straightforward for a network designer to anticipate.

Our current simulator and RL system requires several improvements before release to a more general audience. First, network configuration should be centralized and made more explicit and configurable by an end-user. Second, we need to design a general API for inserting arbitrary security mechanisms and protocols. Third, our current infrastructure is a prototype and needs to be further optimized for speed and memory so that larger scale networks can be simulated and attacked in a reasonable time frames.

Summary and Future Work

We introduced the problem of proactive security in the domain of DoS attacks in P2P networks. We showed how to formulate this problem as reinforcement learning and described a solution approach based on policy-gradient descent. Across a variety of networks this approach was able to significantly outperform baseline heuristics and approach the performance of search-based strategies that use oracle knowledge of the network. There is still a significant gap between the oracle search-based approaches and RL, which suggest that there is still room to improve the RL approach and/or that there is a theoretical gap between the achievable performance of policies with and without oracle knowledge.

There is ample opportunity for additional future work in this area. The experimental setup contains many parameters, ranging from the distribution from which network topologies are drawn to the patterns of query generation and learning rates. Understanding better how these choices relate to the performance of various methods is an important task.

We also plan to consider a much broader policy space for botnet control, where the malicious nodes have individual policies that are learned with various mechanisms for achieving coordination among the nodes. One such example would be to use coordination graph structures (Guestrin, Lagoudakis, & Parr 2002). Furthermore, it will be interesting to consider extending the observation space of the nodes to include aspects of the network environment that might be realistically observed.

Finally, the current model assumes that clients will always accept a connection from a peer, but will not always accept all queries. A more realistic model for many P2P networks would also have a peer acceptance policy used by well-behaved nodes, to determine whether they even accept an incoming connection request from a new peer. This may have a significant impact on the optimal attack strategy. As the dynamics of the system become increasingly realistic, the potential to improve on naive policies grows. At the same time, the plausibility of obtaining the more complete knowledge required by simulation techniques such as beam search becomes less realistic. The policy gradient method is well positioned to take on more complex models, since learning is achieved without knowledge of the underlying dynamics.

Acknowledgments

This work was supported by NSF grants IIS-0524831 and IIS-0546867.

References

Baxter, J., and Bartlett, P. L. 2000. Infinite-horizon policy-gradient estimation. *Journal of Artificial Intelligence Research* 15.

Bernstein, D. B.; Zhengzhu, F.; Levine, B.; and Zilberstein, S. 2003. Adaptive peer selection. *Proceedings of the 2nd International Workshop on Peer-to-Peer Systems*.

BRITE. 2008. Boston University Representative Internet Topology Generator. http://www.cs.bu.edu/brite/.

Chang, Y.; Ho, T.; and Kaelbling, L. 2004. All learning is local: Multi-agent learning in global reward games. *Advances in Neural Information Processing Systems*.

Daswani, N., and Garcia-Molina, H. 2002. Query-flood dos attacks in gnutella. *Proceedings of the 9th ACM Conference on Computer and Communications Security*.

Guestrin, C.; Lagoudakis, M.; and Parr, R. 2002. Coordinated reinforcement learning. *International Conference on Machine Learning*.

KaZaA. 2008. www.kazaa.com.

Littman, M., and Boyan, J. 1993. A distributed reinforcement learning scheme for network routing. Technical Report CMU-CS-93-165, Robotics Institute, Carnegie Mellon University.

Peshkin, L.; Kim, K.; Meuleau, N.; and Kaelbling, L. 2000. Learning to cooperate via policy search. *International Conference on Uncertainty in Artificial Intelligence*.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. MIT Press.

Tao, N.; Baxter, J.; and Weaver, L. 2001. A multi-agent, policy-gradient approach to network routing. *International Conference on Maching Learning*.