In-Network Filtering of Distributed Denial-of-Service Traffic with Near-Optimal Rule Selection

Devkishen Sisodia
University of Oregon
dsisodia@cs.uoregon.edu

Jun Li
University of Oregon
lijun@cs.uoregon.edu

Lei Jiao
University of Oregon
jiao@cs.uoregon.edu

ABSTRACT

A recent trend to mitigate large-scale distributed denial-of-service (DDoS) attacks is in-network filtering, where victims can deploy traffic-filtering rules in networks other than their own. However, given multiple constraints, such as the number of rules a victim can afford to deploy, the set of rules that DDoS defense entities allow a victim to deploy, and the amount of collateral damage to limit, the selection of rules has a large impact on the efficacy of an in-network filtering solution.

In this paper, we introduce a new, offer-based operational model for in-network DDoS defense and formulate the *NP*-hard rule selection problem for this model. We then design an algorithm that overcomes the fundamental limitations of the classical ACO framework and transform it with several key changes to make it applicable to the domain of in-network DDoS defense. Finally, we use a real-world-based Internet routing topology and two real-world DDoS traces, along with one synthetic trace that follows the attack distribution of the recent Mirai DDoS attack, to evaluate the efficacy and runtime of our algorithm against four other rule selection algorithms, and show our algorithm is near-optimal.

CCS CONCEPTS

• **Networks → Denial-of-service attacks**; • **Theory of computation → Bio-inspired optimization**.

KEYWORDS

distributed denial-of-service (DDoS); in-network DDoS filtering; DDoS-filtering rule selection; rule selection optimization

ACM Reference Format:
Devkishen Sisodia, Jun Li, and Lei Jiao. 2020. In-Network Filtering of Distributed Denial-of-Service Traffic with Near-Optimal Rule Selection. In *Proceedings of the 15th ACM Asia Conference on Computer and Communications Security (ASIA CCS '20), October 5–9, 2020, Taipei, Taiwan.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3320269.3384755

1 INTRODUCTION

While the size, frequency, and complexity of distributed denial-of-service (DDoS) attacks have increased sharply over the years, a recent trend to mitigate large-scale DDoS attacks is **in-network filtering**. Once a DDoS attack has been detected, the victim (or a dedicated machine acting on behalf of the victim), which we call the **DDoS defense agent** (or simply, defense agent), generates DDoS-filtering rules and places them at **DDoS-filtering networks**, which can be scrubbing centers or strategically located transit networks between the sources and victim that use the rules to filter DDoS traffic.

We focus on the **offer-based** operational model for in-network DDoS filtering. Every in-network filtering solution we surveyed assumes DDoS-filtering networks are ready to accept and execute arbitrary DDoS-filtering rules dispatched to them, which is not the case in the real-world. However, in the offer-based model, each DDoS-filtering network inspects candidate rules that the defense agent needs to deploy and makes one or multiple offers, where each offer is a subset of rules that the filtering network is willing and able to deploy. The defense agent then inspects all the offers from all the filtering networks and accepts certain offers, where rules in every accepted offer will then be placed at the filtering network that made the offer. With this model, DDoS-filtering networks have complete autonomy on which, what types of, and how many rules to be deployed in their networks. This autonomy is particularly useful in avoiding rule overflow troubles when a filtering network receives more rules than it can afford to deploy. Furthermore, the process of DDoS-filtering networks making offers eliminates the defense agent's burden of deciding or verifying, for every rule, which DDoS-filtering networks can deploy the rule.

A problem at the center of the offer-based in-network defense model is that the defense agent needs to decide which offer(s) to take. Given every offer is composed of DDoS-filtering rules, we call this problem the **rule selection problem**. The optimal solution to this problem is to *maximize the amount of DDoS traffic to be filtered*, with two main constraints: (1) assuming every filtering network places a price on every offer it makes, the total price of the offers finally selected must be within the defense agent's budget; and (2) assuming every rule has an associated collateral damage, or the amount of legitimate traffic that may be filtered due to the rule, the total collateral damage from all the selected rules must be within a certain limit.

At first glance, this problem is similar to the classic *NP*-complete 0-1 multidimensional knapsack problem, which is to place items into a knapsack to maximize the total *value* of items in the knapsack, while the total *weight* of the items does not break the knapsack. Here, we want to maximize the volume of DDoS traffic that all the rules in selected offers can filter, while still meeting the aforementioned constraints.

However, the rule selection problem has one key difference from the knapsack problem: While the knapsack problem assumes the items are independent, in the rule selection problem, offers may *overlap* (i.e., the same rule may appear in more than one offer), thus are not independent from each other. This difference makes it completely infeasible to directly apply those approximate knapsack solutions, including the dynamic programming algorithm that requires every offer to be exclusive with each other and the ant colony optimization (ACO) framework that has the same requirement [16]. Specifically, with the ACO framework, selecting offers is equivalent to selecting a path in a complete graph in which every node is an offer of a set of rules, and to select an optimal set of offers is to discover the path whose offers have the maximum total amount of DDoS traffic filtered according to the rules in offers. A basic principle here is that the traffic filtered by every rule should be counted only once, even if the rule appears in more than one offer on the path. However, the classical ACO framework violates this basic principle as its path discovery does not take into account the overlapping nature of offers.

In this paper, we inspect possible solutions and design a near-optimal solution to the rule selection problem. In particular, we make the following contributions:

- We introduce a new, offer-based operational model for in-network DDoS defense and formulate the *NP*-hard rule selection problem for this model. The offer-based model allows a victim to express candidate rules to all participating DDoS defense entities and every entity to decide which rules they deploy.
- We design an algorithm that overcomes the fundamental limitations of the classical ACO framework, by transforming it with several key modifications to make it applicable to the domain of in-network DDoS defense.
- We use a real-world-based Internet routing topology and two real-world DDoS traces, along with one synthetic trace that follows the attack distribution of the recent Mirai DDoS attack [10], to evaluate the efficacy and runtime of our algorithm against four other rule selection algorithms, and show our algorithm is near-optimal.

2 RELATED WORK

All of the in-network filtering solutions that we surveyed follow what we call the directive-based model, in which each DDoS-filtering network, given its filtering capabilities (i.e., number of routers capable of filtering traffic and amount of memory to deploy rules on capable routers), are obliged to deploy filtering rules on behalf of a defense agent [9, 11, 15, 21, 22, 27, 31, 33, 34, 37, 38, 41]. In the directive-based model, in most cases, the defense agent does not concern itself with rule selection because, as long as the filtering capabilities at the DDoS-filtering networks are not exhausted, all rules generated by the defense agent will be deployed at filtering networks. As a result, the rule selection problem is not well studied in these defense solutions.

Also as mentioned in Section 1, the problem of generating and deploying rules is outside the scope of this paper, and are orthogonal to rule selection. Unlike the rule selection problem, the rule generation and deployment problems are well studied. For example, El Defrawy et al. [17], Soldo et al. [43], and Kallola et al. [25] present dynamic programming, prefix tree-based, and hierarchical heavy hitter (HHH) algorithms, respectively, for solving the rule generation problem at either single or multiple routers on the Internet. Along with assuming that DDoS-filtering networks are willing and able to deploy generated rules, these works also assume that the defense agent has complete knowledge of the filtering capabilities at the filtering networks. This is a strong assumption, since most autonomous systems (ASes), or DDoS-filtering networks, would treat their filtering capabilities as private and sensitive information. Therefore, while the defense agent can leverage such works to generate rules, it still needs to choose only a subset of rules offered to be deployed at the filtering networks. Cooke et al. [14] and Xu et al. [46] study and solve problems related to rule deployment, such as deployment of DDoS monitoring sensors and rules, while Armbruster et al. [12] present a solution to the deployment of traffic filtering rules for DDoS attacks that exploit IP spoofing. Furthermore, Zhang et al. [49] present an adaptable rule deployment solution for Software-Defined Networks (SDNs). Due to the fairly comprehensive research conducted on rule generation and deployment, we instead focus our attention on rule selection. To our knowledge, this work is the first to study this problem in depth.

Finally, although the rule selection problem is not a main focus for existing papers related to in-network DDoS defense, there are papers in the domain of DDoS defense via cloud scrubbing services that tackle similar optimization problems. For example, Jiao et al. [24] determine DDoS traffic diversion and cloud resource allocation under dynamic DDoS attacks, and You et al. [47] design an online auction mechanism to incentivize DDoS-protection service providers to collaboratively scrub DDoS traffic. The general approach these papers take in solving their *NP*-hard optimization problems is by regularizing the objective function and relaxing the constraints to form a new problem solvable in polynomial time. In future work, we plan on investigating the impact of such techniques on the offer-based rule selection problem.

3 OFFER-BASED OPERATIONAL MODEL

3.1 Overview

The offer-based operational model, like other operational models, allows the defense agent to express its filtering needs in any way it sees fit. This means that the defense agent has the freedom to choose from a plethora of mechanisms for filtering DDoS traffic, including, but not limited to, access control lists (ACLs) [4], Remotely Triggered Black Hole (RTBH) signals [13], BGP FlowSpec rules [1], or SDN rules [36]. In this paper, we are not tied to a specific mechanism, but without losing generality we choose to focus on filtering rules based on source IP prefixes (e.g., 162.243.141.0/24). However, it is important to note that the algorithm presented in this paper can be generalized to any other type of filtering rule. In the following subsection, we detail the offer-based model for in-network DDoS defense that allows victims to express their filtering needs by deploying source IP prefix-based rules on networks other than their own.

3.2 Operational Model

Fig. 1 shows an overview of the offer-based model. Once a DDoS attack is detected, the mitigation process begins with the generation of DDoS traffic filtering rules. In most cases, the victim of an attack will generate these rules because it has the best vantage point to observe its specific traffic patterns and the most knowledge on what

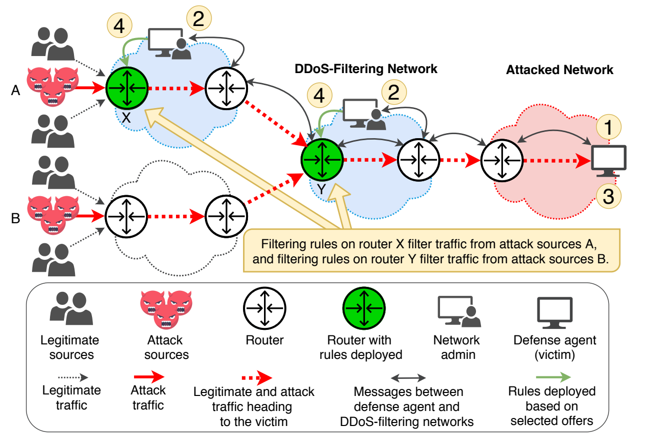

4
2
A
X
DDoS-Filtering Network
Attacked Network
4
2
1
Y
3
B
Filtering rules on router X filter traffic from attack sources A, and filtering rules on router Y filter traffic from attack sources B.
Legitimate sources
Attack sources
Router
Router with rules deployed
Network admin
Defense agent (victim)
Legitimate traffic
Attack traffic
Legitimate and attack traffic heading to the victim
Messages between defense agent and DDoS-filtering networks
Rules deployed based on selected offers

Figure 1: An overview of the offer-based model for in-network DDoS filtering. The defense agent (victim) begins by detecting the DDoS attack and generating rules to filter the attack traffic (circle 1). It then sends those rules to the DDoS-filtering networks, which create offers based on the set of rules each are willing to deploy (circles 2). The filtering networks then send the candidate offers back to the defense agent, who selects the offers to be deployed (circle 3). After the filtering networks of the selected offers are notified by the defense agent, they deploy the rules that make up the selected offers in their networks (circles 4), thus filtering DDoS traffic on behalf of the victim.

traffic is desired and undesired (we will later discuss, in Section 5.5, a scenario when this may not be the case). For generality, the defense agent is the entity that is generating rules. Every rule will indicate how to filter DDoS traffic directed to the victim of the DDoS attack (i.e., what attributes or packet fields to use in order to differentiate attack traffic from legitimate traffic). *Note that rules can only affect the traffic targeted to the victim—the destination IP address of the traffic being filtered must belong to the victim's network.*

The defense agent then distributes these rules to the DDoS-filtering networks (or defending networks) that may be willing to deploy the rules. Each filtering network, given its limited filtering capabilities, then decides which subsets of rules it can or wants to deploy at its discretion; here, every filtering network may run a different decision process and we leave its specifics out of the scope of this work. A subset of rules is called an **offer**, and each filtering network can provide the defense agent with multiple offers.

Specifically, offers are made up of a set of rules, where each rule filters a set of attack flows and, potentially, a set of flows that may be legitimate (i.e., flows that are not attack flows). The value or efficacy of an offer represents the total amount of attack traffic that its rules can filter. Each offer also has two weights associated with it: one weight represents the collateral damage of the offer (or the total amount of legitimate traffic that its rules filter) and the other weight is the cost of the offer. Note, that it does not make sense for a *single* offer to be made up of rules that filter a shared subset of attack flows (or legitimate flows). In other words, it is illogical for an offer Q to include two rules r and s, where rule r filters an IP prefix that is a subset of an IP prefix filtered by rule s. For

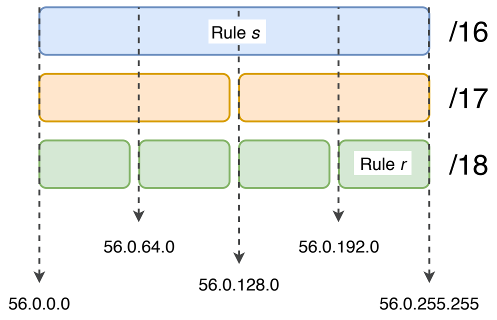


Figure 2: Rules from a single offer should not filter a shared subset of traffic. Since rule s dominates rule r, only one of these rules should be included in the same offer.

example, if s filters 56.0.0.0/16 (or all IP addresses between 56.0.0.0 and 56.0.255.255), and r filters 56.0.192.0/18 (or all IP addresses between 56.0.192.0 and 56.0.255.255), s dominates r, and therefore only one of them should belong to the same offer, as can be seen in Fig. 2 (each block represents a DDoS filtering rule that filters traffic from a specific IP range – the four /18 rules are dominated by the two /17 rules, which are in turn dominated by the single /16 rule). However, *multiple* separate offers *can* be made up of rules that filter a shared subset of traffic flows, or even share the same rules, as explained in Section 4.3.

Furthermore, there is no limit on the number of offers a filtering network can make. However, to simplify this selection process by reducing the total number of offers that a defense agent must consider, we restrict the number of offers a defense agent can select per filtering network to one. This constraint, however, does not negatively affect the overall efficacy of the defense because a filtering network can make offers in such a way as to represent all possible combinations of rules that it is willing to deploy. For example, if a filtering network is willing to deploy rules x, y, and z, but, due to resource limitations, can only deploy a maximum of two rules, it could make 6 different offers: $\{x\}$, $\{y\}$, $\{z\}$, $\{x, y\}$, $\{x, z\}$, and $\{y, z\}$. To reduce the overhead of sending these combinations of offers to the defense agent, the filtering network can simply notify the defense agent that it is willing to deploy any offer with maximum rule size of two that includes rules x, y, and z, along with the price of each rule (so that the defense agent can calculate the price of each offer). Note, making an offer would not reveal a filtering network's capacity; it will only reveal the number of rules a network is willing to deploy for a particular attack.

Once the filtering networks make their offers, the defense agent then selects which offers to buy (among all filtering networks) that will most effectively mitigate the attack, given a limited budget and threshold amount of collateral damage that it is willing to incur. Here, measuring the collateral damage that an offer will incur can depend on the type of filtering rules and could be further subjected to what metric of collateral damage that the defense agent sees fit. For example, if rules are based on source IP prefixes, the defense agent may choose to calculate the collateral damage based on the amount of legitimate traffic filtered, the number of legitimate /32 source IPs filtered (which is what we use in evaluating our selection algorithm in Section 6), or importance of the filtered traffic. After the defense agent selects offers, it notifies the filtering networks that provided those offers to start executing those offers. The filtering networks will then deploy the rules pertaining to the selected offers in their networks to mitigate the attack. The cycle of generating, selecting, and deploying rules will continue until the attack is mitigated.

3.3 The Rule Space Constraint in In-Network Filtering and the Need for Rule Selection

Although source IP-based filtering rules allow the defense agent more fine-grained filtering of DDoS traffic, a significant drawback to this method of filtering is the limited number of rules that can be deployed at defending networks. This limitation is caused by the scarcity of memory space on routers and switches for deploying filtering rules. Specifically, network routers rely on expensive ternary content-addressable memory (TCAM) to forward or discard traffic with low latency, and most high-end routers today only have enough TCAM space to support a few thousand filtering rules without significantly reducing performance [2, 5, 39].

The problem of limited memory space for filtering rules is compounded by the sheer scale of botnets today. The Mirai botnet was found to have control over 50 million unique IP addresses spread all across the world [10]. A victim who is under attack from such a large portion of the Internet simply cannot deploy a filtering rule for each /32 IP address. In fact, even deploying /32 filtering rules (a /32 rule refers to a rule that drops traffic from a specific /32 IP address, thereby achieving the finest filtering granularity possible) to filter a portion of Mirai's IP addresses would cost an astronomical amount of money, assuming the victim would have to pay for each rule deployed on each defending network's routers.

Therefore, the victim would need to *aggregate* /32 prefix rules into more coarse-grained prefixes (e.g., /24 or /16), in order to reduce the number of rules deployed. However, by doing so, the victim loses the ability to prevent filtering legitimate traffic, or incurring collateral damage, that may originate from legitimate source IPs within the more coarse-grained prefixes. In fact, there are examples of DDoS attacks that try to exploit this problem, one of which was proposed in recent literature [42]. Furthermore, recent studies attempt to analyze how in-network DDoS defense solutions attempt to handle this problem, albeit not from a rule selection perspective [48].

Another factor to consider in the problem of balancing collateral damage and monetary cost, is the location at which offers are made, and ultimately, where rules are deployed. The location of defending ASes chosen for filtering affects the collateral damage and monetary cost incurred by a victim. For example, the closer defending ASes are to the attack sources, the victim has a lower chance of incurring collateral damage, but will probably incur a relatively large monetary cost due to more rules being deployed. On the other hand, the closer defending ASes are to the victim, the victim has a higher chance of incurring collateral damage, but will probably incur a relatively small monetary cost. Fig. 3 depicts how the location of deployed rules can impact the collateral damage and monetary cost incurred by a victim. If filtering rules were to be deployed as close to the attack sources as possible, as seen in Fig. 3a, the victim could deploy /32-granular rules at each source AS, thereby dropping all attack traffic, without incurring any collateral damage (only attack flows A_1, A_2, A_3, A_4, A_5, and A_6 would be dropped). However, the victim would need to deploy a total of 6 separate rules (corresponding to each attack flow) at ASes 1, 2, 7, 3, 9, and 4. If the rules were to be deployed on ASes somewhere between the attack sources and victim (e.g., at ASes 11, 12, 13, and 14), as seen in Fig. 3b, the victim could deploy a more coarse-grained rule at ASes 12 to filter A_2 and A_3, and another coarse-grained rule at AS 13 to filter A_4 and A_5. The victim could still deploy two /32-granular rules at AS 11 and 14 to filter A_1 and A_6, respectively. Therefore, the victim would only need to deploy a total of 4 rules, instead of 6. However, the coarse-grained rules at AS 12 and 13 may also filter legitimate flows L_1 and L_2, causing collateral damage. Finally, if filtering rules were deployed at the victim's AS, as seen in Fig. 3c, then the victim, at the very least, could deploy 1 very coarse-grained rule to filter all of the attack traffic. However, this coarse-grained rule could also filter L_3 (along with L_1 and L_2), causing even more collateral damage than the previous scenario (of Fig. 3b). Note also that under link-flooding attacks, where a link several hops upstream to the victim is DDoSed, deploying filtering rules at the victim will be futile, and instead, filtering rules must be deployed further upstream to alleviate the attacked link.

The conundrum of balancing the cost of filtering legitimate traffic with the cost of deploying filtering rules leads to an interesting, and not well-studied optimization problem: maximize the amount of DDoS traffic filtered, while limiting the amount of collateral damage incurred and money spent on deploying filtering rules. In the next section, we formalize this optimization problem, which we call the rule selection problem.

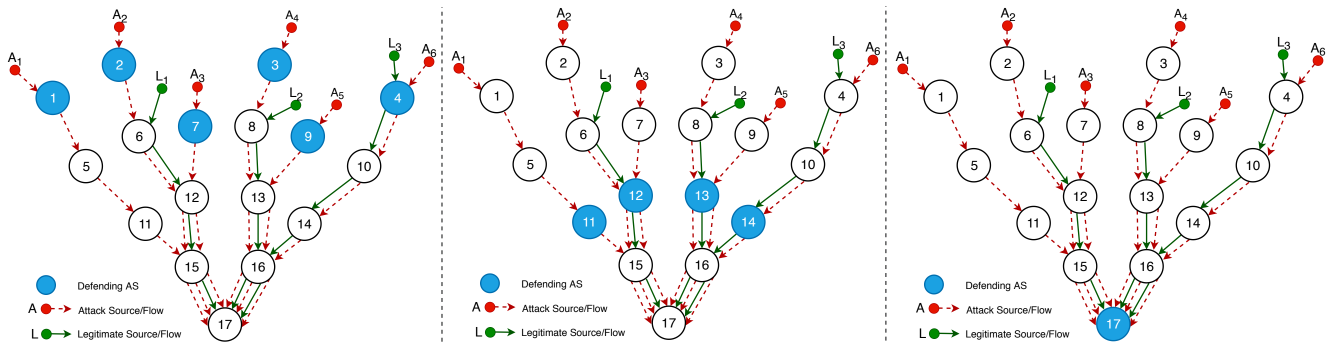
4 RULE SELECTION PROBLEM DEFINITION

4.1 Overview

After the defense agent generates rules and the DDoS-filtering networks provide offers, the defense agent must select a set of offers that most effectively mitigates the DDoS attack. However, there are a number of factors that the defense agent must consider before selecting an offer. Specifically, the defense agent must consider the efficacy of the offer (i.e., the amount of attack traffic or number of attack sources filtered), the collateral damage incurred by the offer (i.e., the amount of legitimate traffic or number of legitimate sources filtered), and the price of the offer.

The defense agent has the freedom to decide which factors to treat as objectives and which to treat as constraints. In this paper, we focus on maximizing the defense efficacy, while keeping the maximum total collateral damage and the maximum amount of money spent on defense as constraints. Thus, we formulate the rule selection problem as Equation 1. Table 1 summarizes all of the notations we use.

Some issues are outside the scope of this paper. In particular, this paper is not concerned with how rules are generated and simply assumes the rules generated for traffic filtering are the input to the rule selection problem. It is not concerned with how rules may be deployed at filtering networks either, except that it determines which filtering networks should deploy different rules as the output of the solution to this problem.



(a) At the attack sources. **(b) In the middle.** **(c) At the victim.**

Figure 3: Three example scenarios of how the location of selected DDoS-filtering networks can impact the collateral damage and monetary cost incurred by a victim.

Table 1: Notations.

Notations	Definitions
$j \in J$	network j in set of all networks J
$i \in I$	offer i in set of all offers I
$I_j \subseteq I$	set of all offers provided by network j
$k \in K$	attack flow k in set of all attack flows K
$u \in U$	legitimate flow u in set of all legitimate flows U
$J_k \subseteq J$	set of networks that flow k passes through
$J_u \subseteq J$	set of networks that flow u passes through
a_k	amount of traffic belonging to attack flow k
b_{ijk}	binary: whether offer i from network j can filter k
a_u	amount of traffic belonging to legitimate flow u
b_{iju}	binary: whether offer i from network j can filter u
x_{ij}	binary: whether to select offer i from network j
P_{ij}	price of offer i from network j
W_c	defense agent's collateral damage threshold
W_b	defense agent's budget

4.2 Formulation

$$
\begin{aligned}
\max \; & \sum_{k \in K} a_k \max_{j \in J_k, i \in I_j} \{x_{ij} b_{ijk}\} \\
\text{s.t.} \; & \sum_{u \in U} a_u \max_{j \in J_u, i \in I_j} \{x_{ij} b_{iju}\} \le W_c \\
& \sum_{j \in J} \sum_{i \in I_j} P_{ij} x_{ij} \le W_b \\
& \sum_{i \in I_j} x_{ij} \le 1, \quad \forall j \in J \\
& x_{ij} \in \{0, 1\}, \quad \forall j \in J, \forall i \in I.
\end{aligned} \tag{1}
$$

4.2.1 *Optimization Objective.*

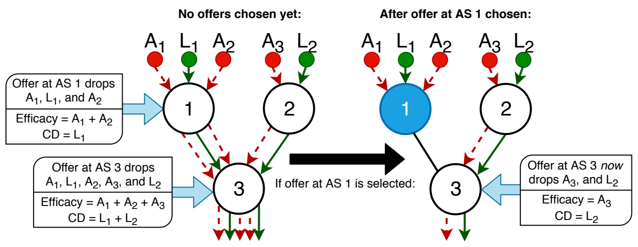
First, we formulate the objective function of the rule selection problem. Let K be the set of all attack flows, where k represents an attack flow in K, and let a_k be the amount of traffic of attack flow k. Let J_k be the set of filtering networks that attack flow k passes through, where j represents a filtering network, and let I_j be the set of all offers provided by filtering network j, where i represents a single offer. Also, let b_{ijk} denote a binary value that represents whether offer i from filtering network j filters attack flow k. Last but not least, the binary variable x_{ij} denotes whether to select offer i from filtering network j for deployment. Note, if there exists an offer i from filtering network j that can filter attack flow k and it has been selected, then $x_{ij} b_{ijk} = 1$, otherwise 0. Note then, if there exists at least one offer i from all of the filtering networks in J_k that can filter attack flow k and it has been selected, then $\max_{j \in J_k, i \in I_j} \{x_{ij} b_{ijk}\} = 1$, otherwise 0. We take the max of $x_{ij} b_{ijk}$ over $j \in J_k$ and $i \in I_j$ because we should not credit multiple offers for filtering the same attack flow k. Therefore, to maximize the total amount of attack traffic filtered, the objective function is written as $\max \sum_{k \in K} a_k \max_{j \in J_k, i \in I_j} \{x_{ij} b_{ijk}\}$.

4.2.2 *Optimization Constraints.*

Next, we formulate the constraints of the rule selection problem. The first constraint ensures that the total amount of collateral damage is within tolerance. Let U be the set of all legitimate flows, where u represents a legitimate flow in U, and let a_u be the amount of traffic of legitimate flow u. Analogously, let b_{iju} be a binary value that represents whether offer i from filtering network j filters legitimate flow u, and let W_c be the threshold for the total amount of collateral damage the defense agent is able to tolerate. We again do not credit multiple offers for filtering the same legitimate flow u. Therefore, we have the collateral damage constraint as $\sum_{u \in U} a_u \max_{j \in J_u, i \in I_j} \{x_{ij} b_{iju}\} \le W_c$. The second constraint ensures that the total cost of the selected offers respects the total budget. Let P_{ij} represent the price of offer i from AS j and W_b represent the defense agent's budget. Then, we have $\sum_{j \in J} \sum_{i \in I_j} P_{ij} x_{ij} \le W_b$. Without loss of generality, the third constraint, $\sum_{i \in I_j} x_{ij} \le 1, \forall j \in J$, limits the number of offers that can be selected per AS to one. Finally, the fourth constraint ensures the atomicity of an offer (i.e., the defense agent can either select or refuse to select an offer, and cannot split an offer), which we express as $x_{ij} \in \{0, 1\}, \forall j \in J, \forall i \in I$.

4.3 Challenges

The rule selection problem looks similar to the NP-complete 0-1 multidimensional knapsack problem [35]. The rule selection problem is 0-1 because an offer can either be selected or not and cannot be broken down into smaller offers, and it is multidimensional because it has multiple constraints. Similarly to the classic knapsack problem, the objective of the rule selection problem is to select



No offers chosen yet:
After offer at AS 1 chosen:
A1 L1 A2 A3 L2
Offer at AS 1 drops A1, L1, and A2
Efficacy = A1 + A2
CD = L1
Offer at AS 3 drops A1, L1, A2, A3, and L2
Efficacy = A1 + A2 + A3
CD = L1 + L2
If offer at AS 1 is selected:
Offer at AS 3 now drops A3, and L2
Efficacy = A3
CD = L2
1
2
3

Figure 4: An example illustrating how offers are value-dependent items. The term "efficacy" represents the amount of DDoS traffic filtered, and "CD", or collateral damage, represents the amount of legitimate traffic filtered.

items (i.e., offers) so that the total value of the items is maximized (i.e., the amount of DDoS traffic filtered), while the weight of the selected items (i.e., the price of all selected offers and the total amount of collateral damage incurred) are within the constraints. Note that no fully polynomial-time approximation scheme exists for 0-1 multidimensional knapsack problem, unless $P = NP$ [29].

However, the rule selection problem is NP-hard and may not be in NP [40] due to the overlapping nature of offers. Offers from the same filtering network or different filtering networks can overlap in terms of the traffic flows that they filter. For example, let us assume there are two filtering networks, y and z, that are on the path of flow f. Let us also assume that both filtering networks y and z provide two separate offers that both contain a rule which filters f, neither of which have been selected yet by the defense agent. With overlapping offers, the selection of one item has an impact on which other items may be selected (e.g., selecting the offer from y decreases the efficacy – if f is an attack flow – or the collateral damage – if f is a legitimate flow – of the offer from z). Each time an item is considered for selection, one would need to check for overlap among the item under consideration and all of the items selected so far, in order to calculate the knapsack's new value (i.e., efficacy) and weights (i.e., price and collateral damage) if the item under consideration is chosen. Due to the existence of overlapping rules, or, in other words, *value-dependent items*, the complexity of the problem increases significantly.

Fig. 4 illustrates an example of value-dependency in two offers. Offers from AS 1 and 3 both filter attack flows A_1 and A_2, and legitimate flows L_1. If the offer from AS 1 is selected, then the offer at AS 3 no longer filters A_1, A_2, and L_1. Therefore, both its efficacy (value) and collateral damage (weight) correspondingly decrease.

The value-dependent and correlated offers of the rule selection problem make it similar to the NP-hard 0-1 multidimensional knapsack problem with value-dependent items, and is unlikely to be solved in pseudo-polynomial time [32, 40]. Polynomial and pseudo-polynomial time heuristics must restrict the problem to a single linear constraint which makes them impractical to directly apply to the rule selection problem [19]. However, there are algorithms for the general 0-1 knapsack problem that can be used as a basis to create a unique, near-optimal solution for the rule selection problem, one such being the ACO algorithm.

5 SOLUTION

5.1 Analysis of Classical Algorithms

We analyze several classical algorithms for the knapsack problem and transform them for the rule selection problem. Many classical algorithms, such as the branch-and-bound and dynamic programming algorithms, cannot be directly applied to the rule selection problem, and therefore we must modify them to handle the unique properties of our problem.

5.1.1 Greedy and Naive Algorithms: The greedy and naive algorithms simply order the set of offers by a heuristic (the greedy algorithm orders the offers by their value or the amount of attack traffic filtered and the naive orders them by their value-to-weight ratio or the ratio of amount of attack traffic filtered to collateral damage incurred). These algorithms run in linear time, and therefore have the shortest runtimes among all other algorithms we analyzed. Unfortunately, the ordering step favors certain offers which may cause the algorithms' failure to consider offers, that if selected, may have lead to a better solution. As a result, the greedy and naive algorithms perform relatively poorly in most cases.

5.1.2 Branch-and-Bound-Based Algorithm: The optimal branch-and-bound algorithm can be adapted to the rule selection problem with only slight modifications. This algorithm essentially enumerates all candidate solutions by building and branching out a binary tree, while estimating the bounds on the optimal solution at each iteration. In the case of the rule selection problem, each node in the tree represents a combination of selected offers, and each node contains a value (the amount of attack traffic filtered), two weights (collateral damage and price), and an upper bound (the maximum amount of value that can be achieved at the current node if it were to be completely branched out). This algorithm always returns the optimal solution. However, its time complexity is exponential in the number of offers. Therefore, running the branch-and-bound-based algorithm for optimal rule selection is not feasible in a real-world scenario for DDoS defense.

5.1.3 Dynamic Programming-Based Algorithm: The classical dynamic programming algorithm can optimally solve the 0-1 knapsack problem, however it cannot solve the rule selection problem because of its multiple constraints and the overlapping nature of offers. In order to apply the classical dynamic programming algorithm to the rule selection problem, we must handle the potential for overlapping offers by performing a pre-processing step. In this step, if there exists two overlapping offers, the algorithm removes the overlapping attack sources from the offer with the lower value-to-weight ratio, solely for the purpose of this algorithm (it does not actually modify the offer). This required pre-processing step prevents the dynamic programming-based algorithm from guaranteeing an optimal solution. Similarly to the greedy and naive algorithms, the pre-processing step potentially eliminates offers from consideration, that if selected, would lead to an optimal solution. However, in most cases, it achieves significantly better efficacy than the greedy and naive algorithms and significantly faster runtimes than the branch-and-bound algorithm, due to its pseudo-polynomial time complexity in the number of offers.

Due to the limitations of the greedy, naive, branch-and-bound-based, and dynamic programming-based algorithms, we look into the classical ACO framework. What follows is a detailed explanation of the classical ACO framework, why it cannot be directly applied to the rule selection problem, and our new algorithm which overcomes the classical ACO framework's fundamental limitations to be applicable to the rule selection problem.

5.2 Overview of the ACO Framework

To address the rule selection problem, we employ the ACO framework. The ACO framework is inspired by the foraging behavior of some ant species, which deposit pheromone trails on the ground in order to mark favorable paths that should be followed by other members of the colony [16]. The ACO framework leverages this behavior for solving optimization problems, such as the NP-hard traveling salesman problem (TSP).

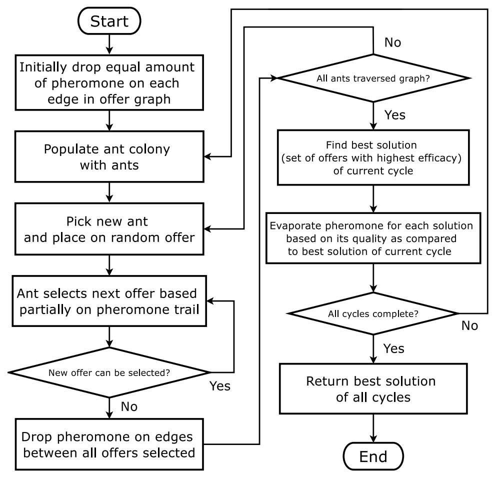
The ACO algorithm is an iterative algorithm. During each iteration, or cycle, a number of ants traverse a graph (e.g., a graph of cities for TSP). Each ant builds a solution (i.e., a sequence of nodes that create a path from the first to the last visited node) by walking from node to node on the graph. An ant chooses the next node in its walk according to a stochastic mechanism that is biased by the amount of pheromone between the current node and all possible next nodes. For example, if an ant is on node i and there is a node j adjacent to i, then the probability of the ant choosing j as its next destination is proportional to the amount of pheromone associated with edge (i, j).

At the end of a cycle, a certain amount of pheromone at each edge is evaporated, based on the quality of the solutions constructed by the ants – more pheromone is evaporated between the nodes that make up low quality solutions (for TSP, a solution represents a route and its quality is indirectly proportional to the length of the route). This causes ants in future cycles to be more attracted to solutions similar to the best ones previously constructed. At the end of the last cycle, the overall best solution is chosen.

There are many existing papers that tackle the multidimensional knapsack problem using ACO algorithms [8, 18, 23, 28]. However, like the classic solutions, the solutions from these papers cannot be directly applied to the rule selection problem. This is again due to the correlated nature of offers and their potential for overlapping. The ACO-based algorithm presented in this paper accounts for this property. Aside from transforming the rule selection problem into a graph-based problem, we make two main changes to the traditional ACO framework: we modify how ants traverse the graph, and how pheromone is dropped onto the graph. To the best of our knowledge, this is the first time the classical ACO framework has been adapted and applied to the domain of in-network DDoS defense.

5.3 ACO-Based Rule Selection Algorithm

5.3.1 Graph Construction for Rule Selection. In order to apply the ACO framework to tackle the rule selection problem, there are several distinctions that need to be made between the rule selection problem and other classical problems, such as TSP. For the rule selection problem, the ant colony will traverse a complete graph, where each node will represent an offer. Unlike in TSP, for the rule selection problem an edge in the graph has no concrete real-world representation. However, the amount of pheromone dropped along an edge will impact the probability of choosing the offers that share that edge. Also, this algorithm needs to take into account the potential of overlapping offers. That is, if two offers filter the same portion of traffic, choosing one offer should affect the probability of choosing the other offer. Lastly, the way ants lay pheromone throughout the graph will be unique to the rule selection problem. The process of laying pheromone in our ant colony system is explained in more detail later in this section. The rest of this section will be devoted to detailing the ACO-based algorithm for rule selection.



Start
Initially drop equal amount of pheromone on each edge in offer graph
Populate ant colony with ants
Pick new ant and place on random offer
Ant selects next offer based partially on pheromone trail
New offer can be selected?
Yes
No
Drop pheromone on edges between all offers selected
All ants traversed graph?
No
Yes
Find best solution (set of offers with highest efficacy) of current cycle
Evaporate pheromone for each solution based on its quality as compared to best solution of current cycle
All cycles complete?
No
Yes
Return best solution of all cycles
End

Figure 5: Flow chart of the rule selection algorithm.

5.3.2 Input Parameters. Along with the aforementioned graph of generated offers, G, the ACO-based rule selection algorithm will take two main parameters as input: the total number of cycles, C_{max}, and the total number of ants per cycle, A_{max}. The total number of cycles and ants per cycle are proportional to the quality of the final solution and the runtime. Additionally, the algorithm requires the tuning parameters, m, n, α, β, ρ, τ_c, τ_{max}, and τ_{min}, which will be explained later. These parameters will be based on the victim's needs. For example, the victim can decrease C_{max} and A_{max}, to keep the runtime fast while sacrificing efficacy, especially if it has many offers to select (more offers translates to longer runtimes). On the other hand, the victim can increase C_{max} and A_{max}, to increase efficacy while also slightly increasing the runtime, especially if it only has a handful of offers to select.

5.3.3 Initialization. Now, let's walk through the entire algorithm from beginning to end. A summary of the algorithm's work flow is depicted in Fig. 5. The algorithm is outlined in Algorithm 1. Each offer will have a value and two weights associated with it. The value is the amount of attack traffic the offer can potentially filter (Equation 2 shows the value of an offer r from an DDoS-filtering network j). One weight represents the amount of collateral damage potentially incurred by the offer (Equation 3), and the other represents the offer's price (Equation 4).

$$value(r) = \max \sum_{k \in K} a_k \max_{j \in J_k, r \in I_j} \{x_{rj} b_{rjk}\} \tag{2}$$

$$weight_c(r) = \sum_{u \in U} a_u \max_{j \in J_u, r \in I_j} \{x_{rj} b_{rju}\} \quad (3)$$

$$weight_b(r) = P_{rj} \quad (4)$$

Algorithm 1: ACO-Based Rule Selection Algorithm

```
Input: $G$, $C_{max}$, $A_{max}$, $\tau_{min}$, $\tau_c$ and other tuning parameters
Output: $s_{sol}$
1  while $c \le C_{max}$ do
2      $s_{loc} = \varnothing$;
3      $\tau_{xy} = \tau_{min}, \forall xy \in G$;
4      while $a \le A_{max}$ do
5          $s_a = \varnothing$;
6          choose random offer $x$;
7          add $x$ to $s_a$;
8          update $l_{avl}$;
9          while $l_{avl}$ is not empty do
10             choose next offer $y \in l_{avl}$ based on $p_{xy}$;
11             add $y$ to $s_a$;
12             update $l_{avl}$;
13             update $\tau_{xy}$ by dropping $\tau_c$ between $x$ and $y$;
14         end
15         add $s_a$ to $s_{loc}$;
16         $a = a + 1$;
17     end
18     add $s_{bst} \in s_{loc}$ to $s_{glo}$;
19     update pheromone for each edge to $\tau_{xy}^{new}$;
20     $c = c + 1$;
21 end
22 return $s_{sol} \in s_{glo}$
```

5.3.4 Traversing the Graph. At the beginning of each cycle c (initially, $c = 1$), the algorithm will instantiate an empty list, s_{loc}, to store each solution obtained in that cycle **(lines 1 and 2)**. Also, each edge in G begins with an equal amount of pheromone **(line 3)**, which will be explained in more detail later. Each ant a will build up a solution, s_a, or a set of offers, by traveling through the map and adding each offer it visits to its solution **(lines 4 and 5)**. If any individual offer in the graph surpasses the budget, W_b, or collateral damage threshold, W_c, it is removed from the graph before the algorithm begins. Note, the algorithm lends itself well to parallelization because all A_{max} number of ants can traverse the graph *in parallel*. An ant will begin by choosing an offer x in the graph, uniformly and at random **(line 6)**. It will then add the offer to s_a and the algorithm will update l_{avl}, or the list of all available offers **(lines 7 and 8)**. In doing so, the algorithm is not only removing x from l_{avl}, but is also updating the offers that overlap with x so that attack and legitimate traffic filtered by x does not impact the overlapping offers' values and weights, respectively. Additionally, updating l_{avl} removes all offers that if chosen next would surpass either the budget or collateral damage threshold.

5.3.5 Probability of Choosing the Next Offer. After the first offer is chosen, the ant will continue traveling through the graph, adding to s_a **(lines 9 through 11)**. So while l_{avl} is not empty, the ant will select a next offer $y \in l_{avl}$ based on the probability p_{xy}, given by Equation 5:

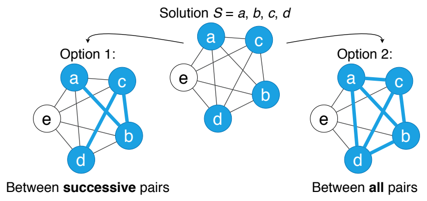


Figure 6: An ant choosing nodes a, b, c, and d as its solution can drop pheromone in two different ways.

$$p_{xy} = \begin{cases} \frac{\tau_{xy}^{\alpha} \times u_y^{\beta}}{\sum_{z \in l_{avl}} \tau_{xz}^{\alpha} \times u_z^{\beta}}, & \text{if } y \in l_{avl} \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

The probability p_{xy} is the amount of pheromone between x and y, or τ_{xy}, multiplied by the *attractiveness* of y all divided by the amount of pheromone between x and every other available offer multiplied by the attractiveness of every other available offer. The tuning parameter α helps to control the importance of the pheromone trail between two offers, while the tuning parameter β helps to control the importance of the attractiveness of an offer. The attractiveness of offer y, or u_y, is essentially the value-to-weight ratio of offer y, given by Equation 6:

$$u_y = \frac{value(y)}{weight_c(y)^m + weight_b(y)^n} \quad (6)$$

The parameters m and n are set by the victim, depending on whether collateral damage or price is of higher (or equal) concern. The amount of pheromone τ_{xy} is initially set to some minimum value τ_{min} **(line 3)**. However, τ_{xy} is updated after each cycle, as described later in this section.

5.3.6 Dropping Pheromone. Multiple updates need to be made after the ant selects the next offer y based on p_{xy}. It adds offer y to s_a and l_{avl} is updated **(line 12)**. But most importantly, the ant will update the graph by dropping a constant amount of pheromone, τ_c, on the edge between x and y **(line 13)**. This process is repeated until l_{avl} is empty, thereby completing the ants journey.

An interesting aspect of the ACO framework is the manner in which pheromone is laid on the graph. As shown in Fig. 6, there are two ways pheromone can be laid: between *each couple* of successively selected offers in a solution, or between *all pairs* of different offers selected in a solution. Given a solution $S = a, b, c, d$, the first option of laying pheromone emphasizes the order of choosing offers by increasing the desirability of choosing a node b over nodes c and d, if the ant is currently on node a. On the other hand, the second option does not emphasize the order of choosing offers, but rather increases the desirability of choosing any node (b, c, and d) that is a part of the same solution S that node a belongs to. From our evaluation of the ACO-based algorithm, we obtained slightly better results when laying pheromone in the second way. This may be because the desirability of choosing the next offer should not be based solely on the previous offer chosen, but on *all* previous offers chosen. This is unlike the TSP problem where the next city on the route should be solely based on the previous city chosen.

After an ant completes its journey, its final solution s_a is added to s_{loc}, and the next ant begins its journey **(lines 14 through 16)**.

Once all of the ants complete their journeys, the best solution s_{bst}, or solution with the highest value, from s_{loc} is saved to s_{glo}, or a list of best solutions at each cycle **(lines 17 and 18)**. The next step of the algorithm is the most important.

5.3.7 *Evaporating Pheromone.* Based on the latest best solution found, each pheromone trail on the graph is updated **(line 19 and 20)**. Pheromone will first be evaporated from each edge in the graph, but after evaporation, the ant that produced s_{bst} will be able to add more pheromone to the edges that it traveled to create the solution so as to negate the effects of evaporation. So given an edge (x, y), the new amount of pheromone between offer x and offer y, or τ_{xy}^{new}, in the graph is equal to (Equation 7):

$$\tau_{xy}^{new} = \left[(1-\rho) \times \tau_{xy}^{old} + \rho \times \Delta\tau_{xy}^{bst}\right]_{\tau_{min}}^{\tau_{max}} \tag{7}$$

where $\rho, 0 \leq \rho \leq 1$, represents the evaporation rate, τ_{xy}^{old} represents the amount of pheromone currently between x and y, and $\Delta\tau_{xy}^{bst}$, which represents the addition of pheromone to the edges of the best solution, is equal to (Equation 8):

$$\Delta\tau_{xy}^{bst} = \begin{cases} value(s_{bst}), & \text{if } (x, y) \in s_{bst} \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Therefore, the pheromone trails of the best solution will remain the same, while trails from the other solutions will be evaporated. This biases ants from the next cycles to choose solutions that are similar to the best solutions found in previous cycles. However, in order to increase the probability that ants do not get stuck at a local maxima, we set a maximum and minimum amount of pheromone for each edge, τ_{max} and τ_{min}, respectively. This entire process is repeated until the number of cycles equals C_{max} **(line 21)**. After the last cycle is completed, the algorithm concludes by selecting and returning the best solution, s_{sol}, among s_{glo} which is the final solution **(line 22)**.

5.3.8 *Approaching Optimality.* The ACO-based rule selection algorithm has a clear advantage over the greedy, naive, and dynamic programming-based algorithms. Unlike the greedy, naive, and dynamic programming-based algorithms, the ACO-based algorithm does not need to eliminate offers in order to handle the overlapping nature of offers. As C_{max} approaches infinity, s_{sol} approaches the optimal solution. In other words, with *enough* cycles, and given the fact that each edge in G will *always* have a non-zero probability of being traversed, the ACO-based algorithm is *guaranteed* to consider *all possible* combinations of offers and thereby *eventually* find the optimal solution (albeit not in polynomial time).

5.4 Complexity Analysis

Performing a comprehensive complexity analysis of the rule selection algorithm is a difficult task due to the stochastic nature of the algorithm. However, the rule selection algorithm presented in this paper is a specialized version of the MAX-MIN Ant System (MMAS) for the all-pairs shortest path problem (APSP) [45]. Sudholt et al. [45] evaluated the running time bounds of the general $MMAS_{APSP}$ algorithm for each iteration to be $O(\Delta\ell\ell^* + \frac{\ell}{\rho})$, given a graph of n vertices, maximum degree Δ, maximum number of edges ℓ on any shortest path, ℓ^* which is equivalent to $\max\{\ell, \ln n\}$, and the pheromone evaporation rate ρ. Therefore, for the rule selection algorithm, under a large-scale DDoS attack where attack sources are highly distributed, the running time bounds becomes $O(C_{max} * (\Delta\ell^2 + \frac{\ell}{\rho}))$. Note, ℓ^* can be replaced by ℓ because under a large-scale DDoS attack, the number of generated rules will most likely be relatively large, causing the number of offers to be relatively large ($\geq$ 100) and thereby increasing the probability that $\max\{\ell, \ln n\} = \ell$. In conclusion, the variable with the largest impact on the rule selection algorithm's running time is the total number of offers the defense agent has to select from.

5.5 Trust & Security Considerations

The rule selection algorithm relies on the trust between defense agents and filtering networks (i.e., that the rules belonging to the offers selected by the defense agent will be deployed by the filtering networks). From the in-network defense systems we studied, some either assume that the system is made up of trusted or semi-trusted networks [9, 11, 15, 21, 27, 33], or rely on a trusted certificate authority (CA) to establish trust among networks [31, 34, 38, 41]. Also, recent work conducted by Gong et al. [20] attempt to tackle the problem of lack of verifiable filtering (i.e., the defense agent has no straightforward way of verifying if a filtering network has correctly executed its filtering request) by utilizing hardware-based trusted execution environments (TEEs) to create verifiable in-network filters. Such work can be leveraged to ensure trust between defense agents and filtering networks. Because our main focus is not on the deployment or implementation of a DDoS defense system, but on a general rule selection algorithm that can benefit most in-network approaches, we leave the more granular details on security and trust to the aforementioned in-network defense systems and solutions.

Let's consider how an adversary may attempt to thwart an in-network DDoS defense solution that utilizes the rule selection algorithm. In the case when a DDoS attack targets transit links instead of a single victim network, as with the Crossfire [26] and Coremelt [44] attacks, rule generation and selection must be done in a collaborative manner as to minimize collateral damage. For simplicity, let us assume that a DDoS attack is targeting one bottleneck link. The network that is directly upstream to the bottleneck link (the network unaffected by the bottleneck) will be responsible for generating and selecting rules because, at its vantage point, it can observe all of the DDoS traffic that needs to be filtered. However, by not knowing what traffic is legitimate (traffic that is desired by networks downstream to the bottleneck link), it may cause collateral damage to traffic that is destined to other downstream networks, which is obviously unacceptable. Therefore, the defense agent (or network generating the rules), must collaborate with the downstream networks, who are the true victims of the attack, to help differentiate attack and legitimate traffic, to generate and select appropriate rules. Unfortunately, persistent link-flooding attacks may be made up of attack traffic that is indistinguishable from legitimate traffic, making it impossible to avoid collateral damage. Lee et al. [30] present a collaborate defense mechanism for mitigating persistent link-flooding attacks, which sends rerouting and rate-control requests to upstream ASes. This in-network solution can utilize the offer-based operational model to help incentivize and guarantee successful collaboration between ASes.

An adversary may also change its behavior to render the rules selected by the rule selection algorithm useless. For example, an adversary could launch an attack that comprises of a series of short-lived bursts, that may only last for a few seconds, and utilize distinct bots for each burst, thereby making the source IP-based rules deployed for previous bursts irrelevant and ineffective in mitigating the current or subsequent bursts. The same applies for attacks that leverage IP spoofing. In such a cases, the defense agent can generate rules that filter on the basis of other TCP/IP fields (e.g., protocol, payload, time-to-live (TTL), Type of Service (ToS), etc.) or a combination of fields. It is important to reiterate that the rule selection algorithm is not dependent on the type of rule that needs to be selected.

6 EVALUATION

6.1 Evaluation Methodology

The main goal of our evaluation is to show the efficacy of our ACO-based rule selection algorithm at filtering attack traffic as compared to other rule selection strategies, including the optimal solution. We also study the runtimes of the strategies to determine how quickly each would respond to a DDoS attack. To make our evaluation as realistic as possible, we test the algorithms on two real-world attack traces and one synthetic trace mimicking a real DDoS attack.

Note that the tuning parameters of any ACO-based algorithm affects its performance. There is a plethora of research on approaches to choosing the best values for the tuning parameters. However, since this is not a focus for our work, we leverage existing research on ACO applied to the multidimensional knapsack problem to choose the values of our parameters [16]. It is very likely that the values we chose for our parameters are not optimal and can be improved. This is an aspect of our work that we will look into in the future.

6.2 Experimental Setup

We built a simulation framework consisting of: 1) the ACO-based algorithm along with the greedy, naive, dynamic programming-based, and branch-and-bound-based rule selection algorithms for comparison, 2) an AS-level Internet topology derived from real-world data, and 3) three attack traces (two of which are real-world traces, while one is a synthetic trace based on a real-world attack). We ran the simulation framework (coded in Java) on a personal computer and high-performance server.

We construct an AS-level Internet topology by using the complete routing table dump obtained from all RouteViews [7] collectors on July 16, 2019. Specifically, for each collector, RouteViews provides routing tables containing AS-level paths from the collector's peers towards all reachable IP prefixes. The AS-level paths from all collectors altogether form our AS-level topology. We treat collectors as the DDoS victims and certain reachable prefixes as attack sources. In order to replay attack traces on this AS-level topology, we assume that the AS-level paths are symmetric; in other words, we assume that the path of any DDoS traffic from a prefix (i.e., DDoS sources) to a collector (i.e., victim) is exactly the reverse of the path from the collector to the prefix. Note this assumption does not always hold in the real world.

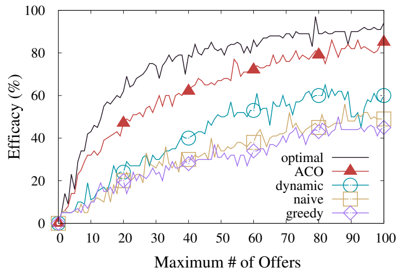
We use three different attack traces. The first is the CAIDA 2007 DDoS attack trace [6], which includes ~4,700 attack sources and ~1,400 source ASes. The second is Merit's RADb DDoS attack trace [3], which includes ~2,300 attack sources and ~1,300 source ASes. We also create a synthetic trace that follows the attack distribution of the September 2016 DDoS attack launched by the Mirai botnet on Krebs on Security, as detailed by Antonakakis et al. [10]. For each of the three traces, we run the simulation 100 times for each collector as the victim, and average all of the runs together. Before the start of a new run, we randomly choose 100 ASes as filtering ASes (or DDoS-filtering networks). In future work, we plan on analyzing how choosing filtering networks based on their tier affects the evaluation results.

6.3 Efficacy

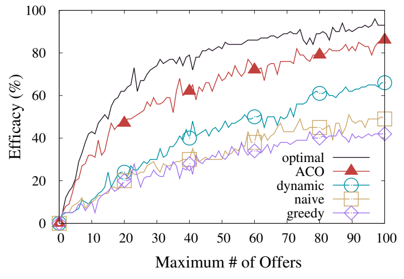
We evaluate the performance of the five selection algorithms in terms of efficacy, or the percentage of total DDoS traffic filtered, with respect to the victim's budget which is represented by the maximum number of offers that can be selected. Note that in most cases, the victim's collateral damage threshold will be met before the maximum number of offers can be selected.

Fig. 7 shows the efficacy achieved by each of the five defense strategies during replays of the CAIDA 2007 (Fig. 7a), RADb 2016 (Fig. 7b), and Mirai 2016 (Fig. 7c) traffic traces, respectively. The first and most obvious trend observed from these graphs is that as the maximum number of rules increases, the efficacy increases. However, this trend seems to be logarithmic and tends to increase only very slightly after about a maximum of 90 offers. Second, the lines are not smooth, which is due to the randomness in selecting the defending ASes after each run of the simulation. Lastly, and most importantly, we can see that our ACO-based algorithm performs only slightly worse than the optimal solution (on average, the ACO-based algorithm ~10% less effective than the optimal), and consistently outperforms the dynamic programming, naive, and greedy algorithms.

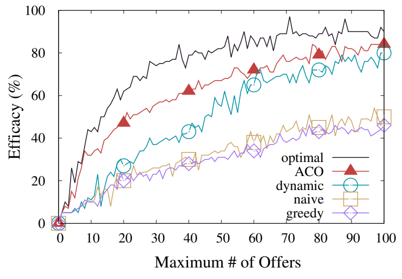
Both the greedy and naive algorithms perform underwhelmingly in all three attacks because of how unevenly the attack sources are distributed and how the two algorithms prioritize the offers. If the victim prioritizes offers based on the amount of attack traffic that can be filtered (i.e., the greedy algorithm), then offers that filter the most attack traffic will be selected, ignoring the collateral damage and monetary cost incurred by those offers. Only a few number of offers may be deployed before the collateral damage threshold is surpassed and the victim's budget is met. Similarly, if the victim prioritizes offers based on their value-to-weight ratio (i.e., the naive algorithm), the same problem may occur. Let's take a look at a simple example. For simplicity, let's combine collateral damage and price as one weight so we are dealing with a one-dimensional problem. Let's say our victim's weight threshold is $W = x$ and the rule selection algorithm has two offers to deploy, where offer a has a weight of $w_a = x$ and a value of $v_a = x - 1$, and offer b has a weight of $w_b = 1$ and a value of $v_b = 1$. In this simple example, the rule selection algorithm will only select offer b (it will select offer b first because it is of higher priority and not be able to select offer a due to the weight threshold). If $x > 2$, then the optimal solution would be to choose offer a. This situation occurs more frequently



(a) CAIDA 2007



(b) RADb 2016



(c) Mirai 2016

Figure 7: Efficacy of selection algorithms under different traffic traces.

Table 2: Runtimes for the rule selection algorithms.

6-Core Intel i7 Processor			
Algorithms	**CAIDA 2007**	**RADb 2016**	**Mirai 2016**
optimal	~9 hrs	~9 hrs	~9 hrs
ACO	11.33 s	10.72 s	12.20 s
dynamic	5.79 s	5.75 s	4.32 s
naive	0.37 s	0.71 s	0.60 s
greedy	0.42 s	0.55 s	0.43 s
24-Core Intel Xeon Processor			
ACO	0.83 s	0.74 s	0.91 s

when the distribution of traffic is uneven, which is the case in most DDoS attacks.

While the dynamic programming algorithm achieves significantly better results than the greedy and naive algorithms, it performs worse than the ACO-based algorithm. This is due to the fact that the dynamic programming algorithm cannot handle overlapping offers as effectively as the ACO-based algorithm. In order for the dynamic programming algorithm to run in pseudo-polynomial time, it must deal with overlapping offers in the pre-processing step. This step potentially prevents the dynamic programming algorithm from obtaining an optimal solution. However, in the case of the Mirai 2016 attack (Fig. 7c), the dynamic programming algorithm achieves efficacy close to (and at one point, better than) the ACO-based algorithm, essentially as the maximum number of offers increase. This is most likely because the distribution of attack sources in the Mirai 2016 attack leads to a small number of overlapping candidate offers to select from, and as a result, the pre-processing step does not significantly degrade the quality of the solution.

In conclusion, the ACO-based algorithm achieves the best results among the sub-optimal algorithms, and is relatively close to the optimal solution, regardless of the attack. On average, there is ~10% difference between the efficacy of the ACO-based algorithm and the optimal solution. At its best, the ACO-based algorithm can improve efficacy by more than 20% and 30% over the dynamic programming and greedy/naive algorithms, respectively.

6.4 Runtime

The runtimes for all rule selection algorithms compared in this paper, as shown in Table 2, are dependent on the number of offers to select from. To put this in perspective, in order to find the optimal solution for a maximum of 100 offers on a personal computer (6-core Intel i7 processor), the optimal branch-and-bound-based algorithm takes around 9 hours to find the optimal solution, while the ACO-based algorithm takes around 11 seconds to find a near optimal solution, making it almost 3,000 times as fast as the optimal algorithm. The dynamic programming algorithm takes about half the time of the ACO-based algorithm. The greedy and naive algorithms take less than one second to finish executing. Note that we have not looked into further optimizing the ACO-based algorithm and it may be very possible to reduce the runtimes seen in the table. In fact, by utilizing a total of 24 cores on a server with dual Intel Xeon E5-2690 processors, we were able to reduce the ACO-based algorithm's runtime to *0.83 s*, *0.74 s*, and *0.91 s* for the CAIDA 2007, RADb 2016, and Mirai 2016 traces, respectively. The reduction in runtime is due to the fact that the ACO-based algorithm can leverage parallelization, as explained in Section 5.3.4. Nonetheless, the runtime table shows that the ACO-algorithm, while only being slightly less effective, is significantly faster than the optimal branch-and-bound-based algorithm, and while being relatively slower, is significantly more effective than the dynamic programming, naive, and greedy algorithms.

7 CONCLUSION

Due to the ever-growing size and frequency of DDoS attacks, effective in-network DDoS defense is increasingly necessary. However, the fundamental dilemma for most in-network DDoS defense solutions is generating, selecting, and placing rules in a network so that an attack can be effectively mitigated. In this paper, we tackle the problem of rule selection for in-network DDoS defense.

We make several contributions in this paper. We formalize the NP-hard rule selection problem for offer-based in-network filtering, and transform the ACO framework to create an algorithm that attempts to find a near-optimal solution to the problem. From our understanding, this is the first work to apply the ACO framework to optimize the selection of filtering rules for the purposes of DDoS defense. We evaluate the ACO-based algorithm on a large scale by comparing it to different rule selection algorithms, using real-world and synthetic DDoS traces over an Internet topology derived from real-world data. Our results show that the ACO-based algorithm outperforms the other rule selection algorithms under real-world attacks and performs only slightly worse than the optimal solution even at a large scale.

ACKNOWLEDGMENTS

This project is in part the result of funding provided by the Science and Technology Directorate of the United States Department of Homeland Security under contract number D15PC00204. The views and conclusions contained herein are those of the authors and should not be interpreted necessarily representing the official policies or endorsements, either expressed or implied, of the Department of Homeland Security or the US Government. We further thank Ann Cox and anonymous reviewers of this paper for their comments.

REFERENCES

[1] 2005. Implementing BGP Flowspec. https://www.cisco.com/c/en/us/td/docs/routers/asr9000/software/asr9k_r5-2/routing/configuration/guide/b_routing_cg52xasr9k/b_routing_cg52xasr9k_chapter_011.html.

[2] 2014. CAT 6500 and 7600 Series Routers and Switches TCAM Allocation Adjustment Procedures. http://www.cisco.com/c/en/us/support/docs/switches/catalyst-6500-series-switches/117712-problemsolution-cat6500-00.html.

[3] 2016. A DDoS Event Against the RADb Service. https://www.impactcybertrust.org/dataset_view?idDataset=576.

[4] 2018. Configure Commonly Used IP ACLs. https://www.cisco.com/c/en/us/support/docs/ip/access-lists/26448-ACLsamples.html.

[5] 2019. Cisco Systems and Arbor: Effective DDoS Mitigation in Distributed Peering Environments. https://www.cisco.com/c/m/en_us/network-intelligence/service-provider/digital-transformation/distributed-peering-architecture.html.

[6] 2019. The CAIDA UCSD DDoS Attack 2007 Dataset. https://www.caida.org/data/passive/ddos-20070804_dataset.xml.

[7] 2019. University of Oregon Route Views Project. http://archive.routeviews.org.

[8] Inès Alaya, Christine Solnon, and Khaled Ghédira. 2004. Ant algorithm for the multi-dimensional knapsack problem. In *International Conference on Bioinspired Optimization Methods and their Applications (BIOMA)*.

[9] David G Andersen. 2003. Mayday: Distributed Filtering for Internet Services. In *USENIX Symposium on Internet Technologies and Systems (USITS)*.

[10] Manos Antonakakis, Tim April, Michael Bailey, Matt Bernhard, Elie Bursztein, Jaime Cochran, Zakir Durumeric, J Alex Halderman, Luca Invernizzi, Michalis Kallitsis, et al. 2017. Understanding the Mirai Botnet. In *USENIX Security Symposium*.

[11] Katerina J Argyraki and David R Cheriton. 2005. Active Internet Traffic Filtering: Real-Time Response to Denial-of-Service Attacks. In *USENIX Annual Technical Conference (ATC)*.

[12] Benjamin Armbruster, J Cole Smith, and Kihong Park. 2007. A Packet Filter Placement Problem with Application to Defense Against Spoofed Denial of Service Attacks. In *European Journal of Operational Research (EJOR)*.

[13] Tim Battles, Danny McPherson, and Chris Morrow. 2004. Customer-Triggered Real-Time Blackholes. In *North American Network Operators' Group (NANOG)*.

[14] Evan Cooke, Michael Bailey, Z Morley Mao, David Watson, Farnam Jahanian, and Danny McPherson. 2004. Toward Understanding Distributed Blackhole Placement. In *ACM Workshop on Rapid Malcode (WORM)*.

[15] Christoph Dietzel, Georgios Smaragdakis, Matthias Wichtlhuber, and Anja Feldmann. 2018. Stellar: Network Attack Mitigation Using Advanced Blackholing. In *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*.

[16] Marco Dorigo and Christian Blum. 2005. Ant Colony Optimization Theory: A Survey. In *Theoretical Computer Science*.

[17] Karim El Defrawy, Athina Markopoulou, and Katerina Argyraki. 2007. Optimal Allocation of Filters Against DDoS Attacks. In *Information Theory and Applications Workshop (ITA)*.

[18] Stefka Fidanova. 2007. Ant Colony Optimization and Multiple Knapsack Problem. In *Handbook of Research on Nature-Inspired Computing for Economics and Management*.

[19] Franklin Djeumou Fomeni and Adam N Letchford. 2013. A Dynamic Programming Heuristic for the Quadratic Knapsack Problem. In *INFORMS Journal on Computing (JOC)*.

[20] Deli Gong, Muoi Tran, Shweta Shinde, Hao Jin, Vyas Sekar, Prateek Saxena, and Min Suk Kang. 2019. Practical Verifiable In-network Filtering for DDoS Defense. In *IEEE International Conference on Distributed Computing Systems (ICDCS)*.

[21] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. 2014. SDX: A Software Defined Internet Exchange. In *ACM SIGCOMM Computer Communication Review (CCR)*.

[22] Felipe Huici and Mark Handley. 2007. An Edge-to-Edge Filtering Architecture Against DoS. In *ACM SIGCOMM Computer Communication Review (CCR)*.

[23] Shahrear Iqbal, Md Faizul Bari, and M Sohel Rahman. 2010. Solving the Multi-Dimensional Multi-Choice Knapsack Problem with the Help of Ants. In *International Conference on Swarm Intelligence (ICSI)*.

[24] Lei Jiao, Ruiting Zhou, Xiaojun Lin, and Xu Chen. 2019. Online Scheduling of Traffic Diversion and Cloud Scrubbing with Uncertainty in Current Inputs. In *ACM International Symposium on Mobile Ad Hoc Networking and Computing (MobiHoc)*.

[25] Aapo Kalliola, Kiryong Lee, Heejo Lee, and Tuomas Aura. 2015. Flooding DDoS Mitigation and Traffic Management with Software Defined Networking. In *IEEE International Conference on Cloud Networking (CloudNet)*.

[26] Min Suk Kang, Soo Bum Lee, and Virgil D Gligor. 2013. The Crossfire Attack. In *IEEE Symposium on Security and Privacy (S&P)*.

[27] Angelos D Keromytis, Vishal Misra, and Dan Rubenstein. 2004. SOS: An Architecture for Mitigating DDoS Attacks. In *IEEE Journal on Selected Areas in Communications (J-SAC)*.

[28] Min Kong, Peng Tian, and Yucheng Kao. 2008. A New Ant Colony Optimization Algorithm for the Multidimensional Knapsack Problem. In *Computers & Operations Research*.

[29] Bernhard Korte and Rainer Schrader. [n.d.]. On the Existence of Fast Approximation Schemes. In *Nonlinear Programming*.

[30] Soo Bum Lee, Min Suk Kang, and Virgil D Gligor. 2013. CoDef: Collaborative Defense Against Large-scale Link-flooding Attacks. In *ACM International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*.

[31] Jun Li, Skyler Berg, Mingwei Zhang, Peter Reiher, and Tao Wei. 2014. Drawbridge: Software-Defined DDoS-Resistant Traffic Engineering. In *ACM SIGCOMM Computer Communication Review (CCR)*.

[32] VC Li and Guy L Curry. 2005. Solving Multidimensional Knapsack Problems with Generalized Upper Bound Constraints using Critical Event Tabu Search. In *Computers & Operations Research*.

[33] Xin Liu, Xiaowei Yang, and Yanbin Lu. 2008. To Filter or to Authorize: Network-Layer DoS Defense Against Multimillion-Node Botnets. In *ACM SIGCOMM Computer Communication Review (CCR)*.

[34] Zhuotao Liu, Hao Jin, Yih-Chun Hu, and Michael Bailey. 2016. MiddlePolice: Toward enforcing destination-defined policies in the middle of the Internet. In *ACM Special Interest Group on Security, Audit and Control (SIGSAC)*.

[35] Michael J Magazine and Maw-Sheng Chern. 1984. A Note on Approximation Schemes for Multidimensional Knapsack Problems. In *Mathematics of Operations Research*.

[36] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. 2008. OpenFlow: Enabling Innovation in Campus Networks. In *ACM SIGCOMM Computer Communication Review (CCR)*.

[37] A. Mortensen, T. Reddy, F. Andreasen, N. Teague, and R. Compton. 2019. Distributed-Denial-of-Service Open Threat Signaling (DOTS) Architecture. *Internet Engineering Task Force* (2019).

[38] George Oikonomou, Jelena Mirkovic, Peter Reiher, and Max Robinson. 2006. A Framework for a Collaborative DDoS Defense. In *Annual Computer Security Applications Conference (ACSAC)*.

[39] Kostas Pagiamtzis and Ali Sheikholeslami. 2006. Content-Addressable Memory (CAM) Circuits and Architectures: A Tutorial and Survey. In *IEEE Journal of Solid-State Circuits*.

[40] David Pisinger. 2005. Where are the Hard Knapsack Problems?. In *Computers & Operations Research*.

[41] Sivaramakrishnan Ramanathan, Jelena Mirkovic, Minlan Yu, and Ying Zhang. 2018. SENSS Against Volumetric DDoS Attacks. In *Annual Computer Security Applications Conference (ACSAC)*.

[42] Lumin Shi, Devkishen Sisodia, Mingwei Zhang, Jun Li, Alberto Dainotti, and Peter Reiher. 2019. The Catch-22 Attack. In *Annual Computer Security Applications Conference (ACSAC)*.

[43] Fabio Soldo, Katerina Argyraki, and Athina Markopoulou. 2012. Optimal Source-based Filtering of Malicious Traffic. In *IEEE/ACM Transactions on Networking (TON)*.

[44] Ahren Studer and Adrian Perrig. 2009. The Coremelt Attack. In *European Symposium on Research in Computer Security (ESORICS)*.

[45] Dirk Sudholt and Christian Thyssen. 2012. Running Time Analysis of Ant Colony Optimization for Shortest Path Problems. In *Journal of Discrete Algorithms*.

[46] Yang Xu and Yong Liu. 2016. DDoS Attack Detection under SDN Context. In *IEEE International Conference on Computer Communications (INFOCOM)*.

[47] Wencong You, Lei Jiao, Jun Li, and Ruiting Zhou. 2020. Scheduling DDoS Cloud Scrubbing in ISP Networks via Randomized Online Auctions. In *IEEE International Conference on Computer Communications (INFOCOM)*.

[48] Mingwei Zhang, Lumin Shi, Devkishen Sisodia, Jun Li, and Peter Reiher. 2019. On Multi-Point, In-Network Filtering of Distributed Denial-of-Service Traffic. In *IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*.

[49] Shuyuan Zhang, Franjo Ivancic, Cristian Lumezanu, Yifei Yuan, Aarti Gupta, and Sharad Malik. 2014. An Adaptable Rule Placement for Software-Defined Networks. In *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*.