Online, Victim-driven Generation of DDoS-Filtering Rules

Christopher E. Early
cearly@cs.uoregon.edu

***Abstract*—Distributed Denial-of-Service (DDoS) attacks continue to pose a significant threat to the availability of Internet services, which are increasingly poorly equipped to face the growing scale and frequency of such attacks. Moreover, since attackers continue to discover and quickly exploit new attack vectors, the variety of DDoS attack types continues to grow, posing yet another obstacle to those seeking to defend against these attacks. On the other hand, in the midst of an ongoing DDoS attack, the victim of the attack has the unique advantage of having the most knowledge about the specific traffic patterns present, which the victim can leverage to generate highly effective traffic filtering rules. Herein we first identify and describe a fundamental trade-off that exists between a set of rules' coverage of attack traffic, potential collateral damage, and resource consumption. We then describe a systematic method by which a DDoS victim may generate traffic-filtering rules while adhering to the victim's constraints, thereby allowing a highly individualized defense to be deployed. Our proposed method relies on a unique tree-based data structure along with a set of heuristic algorithms for efficiently generating rules in real time. We evaluate our rule generation procedure via simulated replay of three real-world DDoS attack traces, and show that we can generate rules with high efficacy towards filtering DDoS traffic, while satisfying the victim's constraints.**

I. Introduction

Distributed Denial of Service—or DDoS—attacks continue to pose a major threat to the reliability and availability of critical Internet infrastructure. Especially due to the expanding availability of high-speed Internet access, attackers are increasingly able to carry out attacks that consume extremely large amounts of bandwidth, in turn making defense a progressively more difficult prospect. Another element that contributes to the difficulty of DDoS defense is that many different forms of this attack exist, all of which achieve a similar result, though by employing slightly different methods. This wide variety of different DDoS attack methods serves as an additional barrier to effective defense methods, and all but precludes the use of static signature-based DDoS defense mechanisms.

Since DDoS attacks typically display highly dynamic behavior, equipping a DDoS victim with the ability to quickly and dynamically react to such attacks by generating accurate and effective traffic filtering rules can be advantageously leveraged to enable a novel DDoS defense system, capable of dynamically adapting to changes in attack patterns. Here, the high-level goal is to filter out or drop any DDoS traffic that traverses a particular network. The filtering strategy used to achieve this goal may be simple or complex, and may be enforced in many different ways, but regardless of the specific mechanism used to implement DDoS filtering, a common feature is the use of **rules**, where in this context, a rule is simply an action, e.g., "drop", that is associated with a set of values that describe the traffic to which the rule should be applied. A simple example is a rule to drop all traffic destined to a particular port number.

In order to have the desired effects, these rules must be deployed on the actual network hardware in locations through which the DDoS traffic travels. Ultimately, in order to process traffic and apply the appropriate rules to the matching traffic at line-speed, these network devices are necessarily constrained in the number of rules that they can accommodate. Thus, available **rule space**, which we define as the constraint on the number of rules that can be accomodated, can quickly become a precious resource. In this project, we examine the resource allocation problem that arises from this situation.

As an example, consider a network administrator who wants to drop all traffic originating from addresses that appear on a blacklist. The simplest solution is to deploy a set of rules containing a "drop" rule for each address on the blacklist, but the size of this list, along with constraints on available rule space, may make it infeasible to deploy this set of rules effectively. In this case, the administrator may wish to replace the set of rules with a *smaller* set of rules that will still have the same effect of dropping all traffic from the blacklisted addresses.

This scenario becomes more complex if the network administrator decides that dropping traffic from *all* of the blacklisted addresses is not necessary if, for example, traffic from 90% of the blacklisted addresses can be dropped with a drastically smaller number of rules. Alternatively, the administrator may allow some small amount traffic that does *not* appear on the blacklist to also be dropped if it allows this policy to be enforced using a smaller number of rules. Many similar situations exist,

from which multiple questions arise. For instance, what is the minimum number of rules required to describe a specifically defined class of traffic? Given a blacklist of attacker addresses, what is the maximum number of blacklist addresses that can be dropped given specific resource constraints (i.e., limited rule space)?

Contributing to the difficulty of this problem is the observation that common formats for describing flow-level network traffic, e.g. OpenFlow, offer a rich format by which Internet traffic can be described using nearly any packet header value, across nearly all Internet protocol layers. Thus, in this context the victim of a DDoS attack has a surfeit of options for how to best describe the malicious attack traffic. Therefore, after first deciding *what* traffic it would like to have filtered, the victim must also decide *how* to describe this traffic.

In this work, we focus on how to generate a set of **traffic engineering rules** optimally in one of four relevant dimensions, and according to certain user-defined constraints. We achieve this using a systematic rule generation procedure which we have developed, the description of which makes up the majority of this paper. At the core of our generation procedure is a unique tree-based data structure, along with three polynomial-time heuristic-based algorithms (Section IV) that produce workable solutions to the NP-complete optimization problems described in Section III.

II. Overview

The goal of this project is to generate a set of **traffic engineering rules** for the purpose of DDoS filtering, according to certain user-defined parameters. This rule generation problem is essentially a question of resource allocation, where the resource in question is rule space, e.g., flow table entries, or space in traditional forwarding tables. More specifically, given descriptions of the DDoS traffic, as well as descriptions of any traffic that is known or suspected to be benign, how may we generate a set of rules that result in dropping the DDoS traffic and not the benign, while requiring no more than a minimal amount of rule space?

We begin by requiring two mutually exclusive classes of traffic, which we call **targeted**, and **protected**. We refer to this traffic collectively as **classified** or **labeled** traffic. Here, we consider the targeted traffic to be the DDoS traffic, which the user desires to drop. In contrast, the protected traffic consists of any traffic that the user wants to preserve, i.e., which the user does not want to drop. In addition, we consider a third class of **unknown** traffic, defined as any traffic that does not fall within the previous two classes. This unknown traffic may be covered by the generated rules if necessary. Note that the union of these three sets of traffic comprises *all possible* Internet traffic.

Thus, generally the goals of the rule generation procedure are to generate a set of rules of a specified size that will effectively drop some portion of the targeted traffic, while only affecting a limited portion of the protected traffic. To quantify how well a set of rules meets these goals, we define **coverage**, **collateral damage**, and **potential collateral damage** to be the amount of targeted traffic, protected traffic, and unknown traffic covered by the generated rules, respectively. Put in these terms, the high-level objectives of rule generation are thus to maximize coverage and minimize collateral damage, potential collateral damage, and the size of the ruleset.

(As a brief digression, note that we may also begin with a set of existing rules, and merely extract the traffic described by the rules to serve as the targeted traffic, and consider all other traffic to be protected. In this case, or any case where only the targeted traffic is provided, there is no distinction between the classes of unknown and protected traffic, and thus no difference between collateral and potential collateral damage.)

Ideally, we would like to generate the smallest set of rules that provides 100% coverage, no collateral damage, and no potential collateral damage. These objectives are not independent, however, and often we must compromise one objective in order to optimize for the other. For example, in order to keep the size of the ruleset within some constraint, we may allow the potential collateral damage to become higher if that in turn lowers the collateral damage, since we may care less about unknown traffic than protected traffic; or, we may allow less than complete coverage if necessary to ensure that no collateral damage occurs. Of course, in the trivial case, if no constraint is imposed on the number of rules, or if the constraint is relatively high, then a set of rules may be generated that covers all targeted traffic, i.e., 100% coverage, and causes **no** collateral damage or potential collateral damage. On the other hand, when a sufficiently stringent constraint is imposed on the number of rules, it remains possible to generate a smaller number of broader rules that still cover all of the targeted traffic, but at the expense of also covering protected or unknown traffic, thus incurring collateral or potential collateral damage, respectively.

Note that the strategies we have described above represent only a few among many viable options for many varied scenarios. One could also, for example, choose to optimize toward a minimal number of rules, while keeping coverage, collateral damage, and potential collateral damage within certain bounds instead. We do not claim any specific strategy to be superior in all scenarios, and in fact, generation of traffic engineering rules with regard to specific objectives merely reflects a different set of priorities on the part of the user, and thus

diverse strategies may become equally valid, depending on the circumstances.

We conclude this section with an example, wherein the DDoS attack victim first decides what traffic to filter by classifying received flows as either "attack" or "benign". These classified flows then become the input to the rule generation procedure, and based on these flows the victim may then construct a set of rules, each of which describes traffic in terms of one or more packet header field values, and further defines the action that should be taken when a match occurs. Clearly, the effectiveness of this type of defense hinges on the victim's ability to generate rules that actually match the attack traffic (i.e., high coverage). Moreover the victim should also desire to prevent collateral damage, which is caused in this context by any benign traffic that is covered by a set of filtering rules. Finally, the victim must also be aware of the cost of deploying these rules, which we measure by the size of the set of rules. Therefore, here the objective is to generate rules with maximal coverage and minimal collateral damage with a minimal number of rules. In a context such as this, and especially if the DDoS victim has only limited rule space with which to defend, the optimization of the defense rules towards the victim's specific defense requirements may enable a more effective defense to be implemented with lower rule space requirements.

The algorithms for rule generation that we describe in Section IV are responsible for generating a set of rules that are optimized towards maximum coverage, minimum collateral damage, or a minimum number of rules, while meeting whatever constraints are imposed, whether they be constraints on the maximum size of the ruleset, the maximum amount of acceptable collateral or potential collateral damage, or the minimum amount of coverage.

III. Problem Formulation

Now that we have discussed the context and motivation for this work, and provided a high-level overview of the problem, we continue with a formal definition.

A. Assumptions

As described in Section II, we assume the user posesses a set of traffic flows classified into two mutually exclusive classes to serve as the input to the rule generation procedure. Here a traffic flow is a sequence of packets belonging to a specific TCP/UDP connection or simply all packets sharing certain common properties (e.g., all packets from a specific source to the IP of the user's machine, or more generally all packets that share a common sequence of bits in the TCP/IP header or payload).

B. Rule Generation Procedure Overview

The basic goal of the rule generation procedure is to transform an input of classified traffic flows into set of traffic engineering rules that meet user-defined constraints. Here, we now formally define this goal, as well as the form that these constraints may take.

The rules in question describe traffic based on TCP/IP header fields of the traffic or byte patterns in their payloads. Some of these fields are by nature **aggregatable** in that a single value in a rule may match multiple patterns. Examples are the source IP address, the destination IP address, or the payload. For example, assuming m rules can match traffic based on m different source IP addresses a_i or m different payload patterns p_i (i=1,...,m), if we can aggregate these m addresses or payload patterns into a single source IP address prefix A or a single payload pattern P, i.e., $A = \cup_{i=1}^{m} a_i$ and $P = \cup_{i=1}^{m} p_i$, we can use one rule based on A or P to replace all m rules, thus making it possible to obtain a smaller number of rules. Other fields are **non-aggregatable**, such as the protocol field (TCP or UDP), source port, destination port, time-to-live (TTL), and Type of Service (TOS). For instance, given m rules that apply to m different source port numbers, there is no single value—and thus no single rule—that can match all m port numbers, thus preventing any reduction in the size of the rule set.

We thus focus on the aggregatable fields of labeled traffic flows to generate rules, since the non-aggregatable fields provide no room for optimization. For example, a rule to block all UDP traffic to DNS port 53 is already optimal in terms of number of rules (one), as is a rule to drop any traffic whose TOS field matches some value. Clearly this variety of rules is already easy to produce and efficient to enforce. Thus, we proceed to the aggregatable fields and perform our optimization over these fields.

C. Problem Formulation

We formulate the problem of rule generation as follows. For a given rule r, we define $b(r, T)$, $g(r, T)$, and $u(r, T)$ to be the coverage, collateral damage, and potential collateral damage of rule r, respectively. They represent the amounts of targeted traffic, protected traffic, and unknown traffic that r matches from traffic set T, respectively. As such, if we have a set of rules R=$\{r_i | i$=1, ..., $n\}$, where r_i is a rule, we have $b(R,T)$=$\sum_{i=1}^{n} b(r_i, T)$, $g(R,T)$=$\sum_{i=1}^{n} g(r_i, T)$, and $u(R,T)$=$\sum_{i=1}^{n} u(r_i, T)$ to represent the coverage, collateral damage, and potential collateral damage of the rule set R, respectively, when applied to the set of traffic T. We may now define the rule generation problem as a following multi-objective optimization problem:

How may we generate a set of rules R=$\{r_i | i = 1, ..., n\}$ **such that among all possible sets of rules,** R

has the minimal $g(R,T)$, minimal $|R|$, and maximal $b(R,T)$ for a given set of traffic T?

We tackle this multi-objective problem using the ϵ-constraint method [9]. We develop an *a priori* method which allows the user to first express her preferences on the lower bound of coverage and the upper bound of collateral damage, potential collateral damage, and the number of rules to generate. We then scalarize this multi-objective optimization problem by formulating multiple single-objective optimization problems, which become the basis for our rule generation procedure. Finally we use heuristic-based algorithms to produce workable solutions to each of these problems in polynomial time.

Assuming the user's preference for the lower bound on coverage of targeted traffic is B, the upper bounds on collateral damage is G, and the upper bound on the number of rules that can be generated is M, we define four single-objective optimization problems as follows:

- **Problem 1:** For a given set of traffic T, output a set of rules R=$\{r_i|i=1,...,n\}$ that minimizes $g(R,T)$, whereas $b(R,T)\geq B$ and $|R|\leq M$;
- **Problem 2:** For a given set of traffic T, output a set of rules R=$\{r_i|i=1,...,n\}$ that maximizes $b(R,T)$, whereas $g(R,T)\leq G$ and $|R|\leq M$; and
- **Problem 3:** For a given set of traffic T, output a set of rules R=$\{r_i|i=1,...,n\}$ that minimizes $|R|$, whereas $g(R,T)\leq G$ and $b(R,T)\geq B$.

IV. Algorithms

In this section, we first describe the **F-tree**, which is the fundamental data structure used for rule generation. We then describe our rule generation algorithms, which operate on the F-tree to generate rule sets that meet the criteria defined in Section III. Finally, we describe how we repeatedly employ these algorithms in real-time to dynamically generate rules over the course of a DDoS attack.

A. F-tree

We now introduce a data structure called the F-tree, and describe how a user can initialize its F-tree based on labeled traffic flows. We use an F-tree to discover how the values in an aggregatable field, which we denote as F or F-field, may aggregate from the bottom up in a way that satisfies the objective and meets the constraints of the rule generation problems. Although our design can be easily extended to non-binary trees, for simplicity below we assume F-tree is a binary tree. Figure 1 shows an example F-tree.

Every node on the F-tree has five associated values:

- p: A prefix of an F-field value, which we also call an F-prefix. Every node is associated with a unique p. Nodes with F-prefix of the same length are on the same level of the tree.
- S: a set of traffic flows that share a common F-prefix p;
- b: the amount of targeted traffic from S, where the amount can be in terms of bytes, packets, or TCP or UDP connections;
- g: the amount of protected traffic from S; and
- u: the amount of unknown traffic from S.

Every node N with a set of child nodes $c_1,...,c_n$ (in a binary tree n is 1 or 2), derives five values from those of its children:

- $N.p$ = `prefix`$(\{c_1.p,\ldots,c_n.p\})$, where `prefix()` is a function to extract the longest (i.e., the most specific) prefix from a set of prefixes;
- $N.S$=$\cup_{i=1}^{n}(c_i.S)$
- $N.b$=$\sum_{i=1}^{n}(c_i.b)$
- $N.g$=$\sum_{i=1}^{n}(c_i.g)$
- $N.u$=$\sum_{i=1}^{n}(c_i.u)$

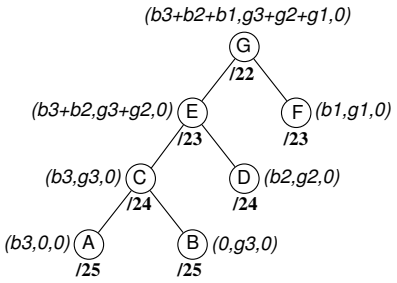


Fig. 1. **An example F-tree where every node represents a source address prefix.** Below every node is the length of its address prefix and next to every node is its (b,g,u) values.

The F-tree has several interesting properties pertaining to rule generation:

- We map every node N to a rule dictating to apply the appropriate action to the traffic whose F-field matches $N.p$. Applying this rule to $N.S$ results in coverage of $N.b$, collateral damage of $N.g$, and potential collateral damage of $N.u$.
- If two sibling nodes are both chosen, representing two rules, their parent node should be used instead to derive simply one rule, without affecting the coverage, collateral damage, and potential collateral damage.
- If a node is chosen to derive a rule, none of its descendants should be used to derive a rule.

B. Rule Generation Algorithms

In order to generate rules, the user will first use labeled traffic flows to populate an F-tree and then use the F-tree to derive rules that solve the rule generation problems listed in Section III-C.

We employ a simple top-down procedure to populate the F-tree based on labeled traffic flows, whereby the

value in each flow's F-field (e.g., the address for an F-tree of source addresses), describes the path from the tree's root to a leaf node, say L, on the F-tree. $L.p$ is then the value in the F-field (i.e., $L.p$ is the longest prefix of the value), $L.S$ is the set of all flows that share this prefix, and $L.b$, $L.g$, and $L.u$ are the sum of the amount of unwanted (bad), protected (good), and unknown traffic in $L.S$, respectively.

After the F-tree is initialized, the fundamental step is called the aggregation step. In this step we replace leaf nodes with their first common ancestor node (which has the longest shared prefix among all ancestors), where the leaf nodes can be two sibling leaf nodes, or a single leaf node without a sibling, or two adjacent leaf nodes that are not siblings—sometimes not even at the same level. (For the first two cases the ancestor node is simply the parent node of the leaf node(s).) With an aggregation step, all descendants of the ancestor will be pruned from the F-tree, and the ancestor node becomes a new leaf node. An aggregation step can be of two types: a clean aggregation step where the g-value and u-value of the ancestor node are *not* greater than those values of nodes being replaced, and a dirty aggregation step where they *are*. Clearly, a clean aggregation step will not introduce extra collateral or potential collateral damage, but a dirty aggregation step will.

We apply the aggregation step on the F-tree repeatedly to determine which nodes to use to produce rules. First, we apply a set of clean aggregation steps over all leaf nodes recursively until no clean aggregation can be done. We also call this process a **clean pruning** process.

We then run a loop process. First, we determine if a subset of leaf nodes of the current F-tree can be used to derive rules that satisfy the constraints; if so, we have met the **stopping condition** and we then use the leaf nodes to derive rules; if not, we select and conduct a dirty aggregation of two adjacent leaf nodes, followed by the clean pruning process to evolve the F-tree, and then go back to the beginning of the loop.

The stopping condition and the heuristic by which we select which dirty aggregation to perform are both dependent on which single-objective optimization problem we seek to solve (Section III-C). Specifically, for problem 1, the stopping condition is to check if in the current F-tree there are M nodes or fewer whose sum of b values are no less than B, and we will apply the dirty aggregation that results in introducing the least extra collateral damage. For problem 2, the stopping condition is to check in the current F-tree there are M nodes or fewer whose sum of collateral continues to stay within the constraints; if not, we must perform a dirty aggregation and we choose the dirty aggregation that brings the smallest decrease of the coverage without violating the constraints. Finally, for problem 3, the stopping condition is to check if we can conduct a dirty aggregation to further decrease the number of rules while ensuring the coverage and collateral damage remain within the constraints; if so, we choose the dirty aggregation that results in the largest number of leaf nodes pruned (i.e., decreases the number of rules the most) without violating the constraints.

Algorithm 1 Minimize Collateral

procedure MIN-COLLATERAL(T, b, m)
  $R \leftarrow$ leaf nodes l of T with $l.b > 0$
  while True **do**
    if $|R| \leq m$ **then**
      break
    end if
    if $b_{sum} \geq b$ **then**
      $R \leftarrow$ top m nodes, sorted by b
      break
    end if
    $A \leftarrow \{lca(r_i, r_{i+1})$ for $r_i \in R\}$
    $A_{min} \leftarrow$ ancestor with minimum g
    $R \leftarrow R \cup \{A_{min}\}$
    $R \leftarrow R - \{$descendants of $A_{min}\}$
  end while
  return R
end procedure

Algorithm 2 Maximize Coverage

procedure MAX-COVERAGE(T, g, m)
  $R \leftarrow$ leaf nodes l of T with $l.b > 0$
  while True **do**
    $g_{total} \leftarrow \sum r_i.g$ for $r_i \in R$
    $A \leftarrow \{lca(r_i, r_{i+1})$ for $r_i \in R\}$
    $G\Delta \leftarrow \{g_{total}$ from a_i for $a_i \in A\}$
    $A_{valid} \leftarrow \{a_i \in A \mid g_i.g \in G_\Delta \leq g_{total}\}$
    if $length(A_{valid}) = 0$ **then**
      break
    end if
    $A_{final} \leftarrow$ ancestor with largest b value
    $R \leftarrow R \cup \{A_{final}\}$
    $R \leftarrow R - \{$descendants of $A_{final}\}$
  end while
  $R \leftarrow$ top m rules by b value
  return R
end procedure

C. Dynamic Rule Generation

In most cases, after a set of rules has been generated and deployed, the victim may continue to receive additional unwanted traffic. Thus, the victim continues to label traffic, insert newly labeled traffic flows to its F-tree, and run algorithms above to derive new rules, all repeatedly at a fixed interval. At each time interval,

Algorithm 3 Minimize Number of Rules

procedure MIN-RULES(T, g, b)
  $R \leftarrow$ leaf nodes l of T with $l.b > 0$
  while True **do**
    $R \leftarrow R$ sorted by numeric value
    $g_{total} \leftarrow \sum r_i.g$ for $r_i \in R$
    $b_{total} \leftarrow \sum r_i.b$ for $r_i \in R$
    $A \leftarrow \{lca(r_i, r_{i+1})$ for $r_i \in R\}$
    $G\Delta \leftarrow \{g_{total}$ from a_i for $a_i \in A\}$
    $B\Delta \leftarrow \{b_{total}$ from a_i for $a_i \in A\}$
    $R\Delta \leftarrow \{|R|$ from a_i for $a_i \in A\}$
    if $length(A) = 0$ **then**
      break
    end if
    $A \leftarrow A$ sorted by corresponding R_Δ value
    for $a_i \in A$ **do**
      if $(g_i \leq g_{total})$ **and** $(b_i \geq b_{total})$ **then**
        $R \leftarrow R \cup \{A_i\}$
        $R \leftarrow R - \{$descendants of $A_i\}$
        break
      end if
    end for
    if no aggregations are made **then**
      break
    end if
  end while
  $R \leftarrow R$ sorted by b value
  $b_{total} \leftarrow \sum r_i.b$ for $r_i \in R$
  for $r_i \in R$ **do**
    if $(b_{total} - r_i.b) \geq b$ **then**
      $R \leftarrow R - \{a_i\}$
      $b_{total} \leftarrow b_{total} - a_i.b$
    end if
  end for
  return R
end procedure

a new set of rules R' is generated, and by comparing R' to the previous interval's rules R, we arrive at the following sets: $R - (R \cap R')$ are rules that must be revoked, $R' - (R \cap R')$ are new rules to be deployed, and $(R \cap R')$ are rules that will remain deployed.

The values of b, g, and u are updated at each time interval according to any newly received traffic flows, and we use an exponentially-weighted moving average to determine the final values of b, g, and u. In this way, these three values decay over time, effectively allowing the rules to time out. For example, if no new traffic is received from a particular unwanted source s, either because a generated rule is completely effective in blocking traffic from s, or because the source s ceases sending attack traffic, the b associated with the corresponding node on the F-tree will decay over time, and the rule will eventually be revoked. On the other hand, if s continues to send attack traffic after the corresponding rule has been revoked, the victim will receive it and will once again generate a rule to block traffic from s. The exponentially-weighted moving average thus provides the best *prediction* of what future traffic will arrive, based on the traffic that has already been received. Ultimately, this process results in a set of rules that reflects the most up-to-date information available about what traffic to filter, thereby more effectively utilizing the victim's rule budget.

V. Evaluation

A. Methodology

To evaluate the rule generation algorithms described in Section IV, we first examine a single round of rule generation, where we apply each algorithm in turn to a set of 1000 randomly generated bit strings of length 32. Note that for evaluation purposes, these bit strings may be thought of as either source IP addresses, or the first four bytes of IP packet payloads. For this static evaluation, and for each algorithm, we vary the two constraints across all possible values, e.g., we vary the constraint on minimum coverage b from 0, representing no coverage of the attack traffic, all the way to 1, which similarly represents complete coverage. Then, we study the value of the objective, e.g., for problem 1, the objective of which is to minimize collateral damage, we vary the constraints b and m, and then measure the collateral damage for each set of rules resulting from each combination of b and m.

Next, we examine our rule generation algorithms in a dynamic setting, by performing three case studies, wherein we apply the procedure described in section IV-C to real-world DDoS attack traces, and measure all relevant metrics over the course of the attack. We show that our rule generation procedure accurately generates rules that meet the specified constraints, and we further demonstrate that for three distinct attacks, our rule generation procedure generates rules that remain effective in the face of quickly shifting attack dynamics.

B. Static Evaluation

We first examine the algorithm for problem 1 described in Section IV. The goal of this algorithm is to minimize the collateral damage, subject to constraints on coverage and number of rules. Figure 2 shows how the coverage, collateral damage, and number of rules vary according to the values of the minimal coverage B and the maximum number of rules M. For these figures we have normalized all axes except M to range between 0 and 1. These figures show that we achieve 100% coverage of the targeted traffic with a rule budget M that achieves a *significantly smaller* rule set, as long

as the minimum coverage (B) is not too high. Collateral damage is indeed minimized, and is zero in most cases. When B is high and M is low, some collateral damage is incurred, since the only way to cover a large percentage of targeted flows with a relatively small number of rules is to allow some collateral damage to occur. This exemplifies the basic tradeoff inherent in this problem.

Next, we continue our evaluation by examining our algorithm for problem 2 described Section IV. Here, the goal is to maximize coverage, while satisfying constraints on the maximum number of rules M and the maximum amount of acceptable collateral damage G. We vary the values for G and M and examine the coverage, collateral damage, and size of the resulting ruleset. As shown in Figure 3, we see that 100% coverage is achieved easily, except when G and M are both low. In these cases, however, the coverage is still maximized, but is subject to the stringent constraints on G and M.

We now examine our algorithm for problem 3, wherein the objective is to minimize the size of the ruleset R, while keeping the collateral damage and coverage within the constraints defined by G and B, respectively. Similarly to the previous case, in figure 4 we vary the values for G and B, and record the size of the resulting ruleset. As expected, the smallest resulting ruleset results from the cases where B is relatively low, or where G is relatively high, or both. When B approaches a normalized value of 1.0, this represents the requirement that nearly all of the targeted traffic must be covered by the set of rules. If this is the case, and the value of G is high, then a small number of rules can still be obtained by introducing a large amount of collateral damage via dirty aggregation. On the other hand, when B remains high, but the value of G approaches 0, very little reduction is possible in the size of the ruleset, since very few (if any) dirty aggregations are possible due to the tight constraint on collateral damage.

C. Dynamic Evaluation

1) Case Study 1: CAIDA2007: The first attack trace against which we evaluate our rule generation procedure was captured by CAIDA in 2007. The attack in question reaches a peak bandwidth of around 80 Mbits/s, and includes over 5000 unique attack sources. In this case study, we apply all three algorithms in turn, and for each algorithm we vary the values of the two relevant constraints. Figure 5 shows the overall dynamics of the attack, in terms of the number of both unwanted and protected flows at a one-second time granularity. Here we can see that the number of unwanted flows per second reaches a peak of roughly 5000 (when this happens, nearly all attack sources are attacking simultaneously).

We begin with the objective of maximizing coverage, subject to constraints on the maximum number of rules M, and the maximum collateral damage G. Here, we vary the value of M, while holding the value of G constant at 0 in all cases. Our goal in defending against this attack is thus to maximize the coverage of the unwanted flows, while keeping the number of rules below M, and without covering any protected flows (which average around 100 per second).

Figure 6 shows the results of our max-coverage algorithm in terms of percentage of unwanted flows filtered at each second, as we vary the value of M. Here, the y-axis shows for each second-length interval, how many unwanted flows from the current interval is filtered by the rules generated in the *previous* interval. Generally, as we increase the value of M, we see a higher percentage of unwanted flows filtered by the generated rules, where a value of 500 is sufficient to filter nearly 100% of the unwanted flows. Similarly, figure 7 shows the cumulative percentage of unwanted flows filtered over the course of the attack. Even with a maximum rule limitation of merely 100, over 60% of all unwanted flows are filtered by the generated rules.

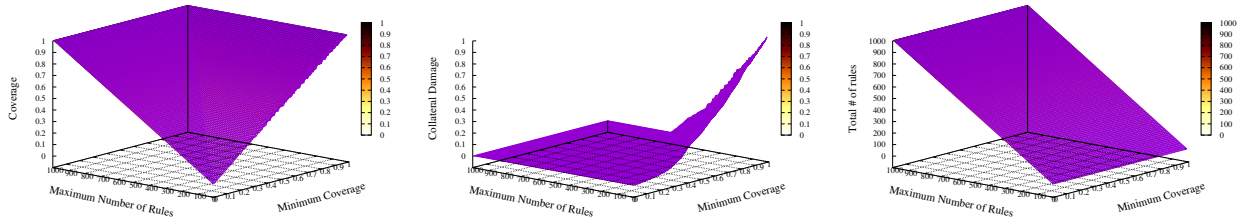
Next, we select the objective of minimizing collateral damage, subject to the constraints B and M, which represent the minimum coverage and maximum number of rules, respectively. Again, we vary the value of M, while B remains constant at 1, representing full coverage of unwanted flows. Our goal is thus to minimize the number of protected flows covered by our rules, while simultaneously covering all unwanted flows and keeping the number of rules less than or equal to M.

Figure 8 shows, at each second over the course of the attack, the percentage of protected traffic flows that are filtered, i.e., collateral damage. Since we keep the constraint B at its maximum value of 1, we see that as we reduce the value of M, the overall collateral damage increases, as shown by figure 9, which shows the cumulative collateral damage as a percentage of the total number of protected traffic flows.

The differences between these results and the results of the max-coverage algorithm merely reflect the consequences of selecting a different objective towards which to optimize. We already know from figure 7 that if we desire no more than 100 rules, and no collateral damage, then we can only achieve roughly 65% coverage. In this case, we desire no more than 100 rules and complete coverage, and we see that we can only reduce the collateral damage to approximately 80%.

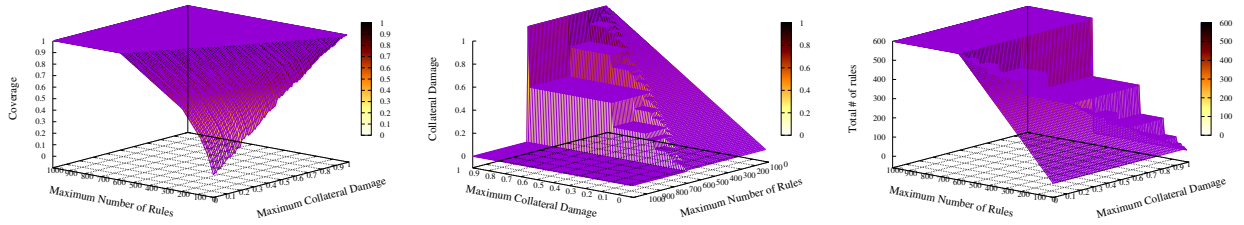
Finally, we choose to minimize the number of rules generated, while keeping coverage greater than or equal to B, and while keeping collateral damage less than or equal to G. Here, we keep the value of G constant at 0, while we vary the value of B from 0.5 to 1.0, representing the range from 50% to complete coverage.

Figure 10 shows the percent coverage achieved by



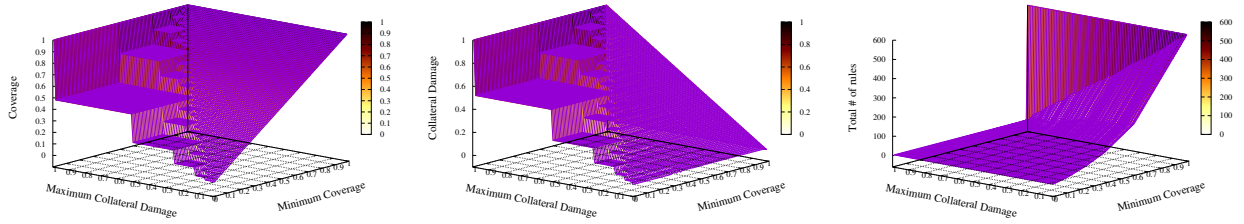
(a) Coverage vs. B and M (b) Collateral damage vs. B and M (c) Number of rules vs. B and M

Fig. 2. **The normalized coverage, normalized collateral damage, and number of generated rules. Rules are optimized toward minimal collateral damage subject to the rule budget M and minimum coverage B.**



(a) Coverage vs. G and M (b) Collateral damage vs. G and M (c) Number of rules vs. G and M

Fig. 3. **The coverage, collateral damage, and number of generated rules, all normalized. Rules are optimized toward maximal coverage subject to the rule budget M and maximum collateral damage G.**



(a) Coverage vs. G and B (b) Collateral damage vs. G and B (c) Number of rules vs. G and B

Fig. 4. **The coverage, collateral damage, and number of generated rules, all normalized. Rules are optimized toward a minimum number of rules, subject to the minimum coverage B and maximum collateral damage G.**

this algorithm as the value of B varies. Here we show that in all cases, the coverage exceeds the constraint, while in all cases the collateral damage remains at 0 (not shown). Here we see similar results to the results of the max-coverage algorithm, where in the most extreme case we achieve on average 65% coverage and no collateral damage with approximately 100 rules. (Figures 11 and 12).

2) Case Study 2: RADB2012: Next, we apply our min-rules algorithm to the RADB2012 attack trace. This DDoS attack was captured by Merit Network, Inc. in 2016, was based on the DNS protocol, and targeted Merit's RADb service, which provides a public registry of network routing information.

For this case study, we employ the min-rules algorithm, meaning that our goal is to defend against this attack using as few rules as necessary to meet the two constraints: B, the minimum coverage, and G, the maximum collateral damage. Here, we execute the rule generation procedure multiple times while varying the value of B from 0.5 to 1, and we keep constant the value of $G = 0$, i.e., no collateral damage.

Again, we first examine the overall dynamics of the attack in question. Figure 13 shows the number of both unwanted and protected flows per second over the course of the attack. While the peak number of flows per second

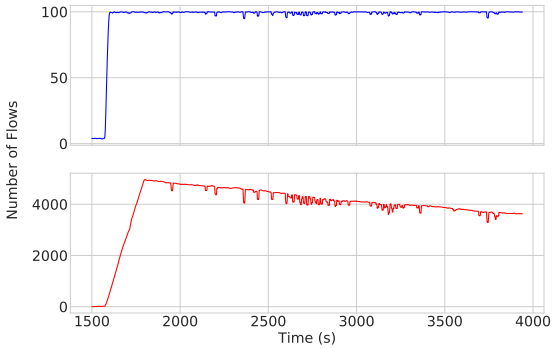


Fig. 5. **Dynamics of unwanted and protected traffic for the CAIDA2007 attack.** Here we show the number of both unwanted and protected traffic flows that would arrive at the victim, if no defense is present.

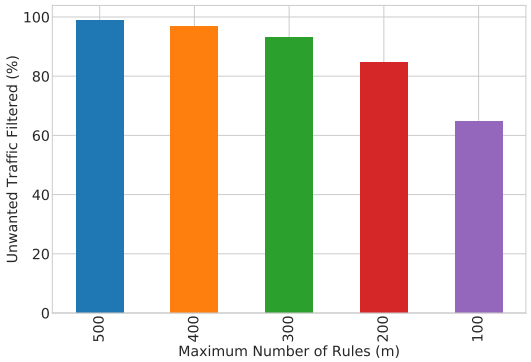


Fig. 7. **Overall percentage of unwanted traffic filtered by the generated rules for the CAIDA2007 attack.** Here we show cumulatively the percentage of unwanted traffic filtered by our rule generation procedure using the max-coverage algorithm, as we vary the limitation on the maximum number of rules M.

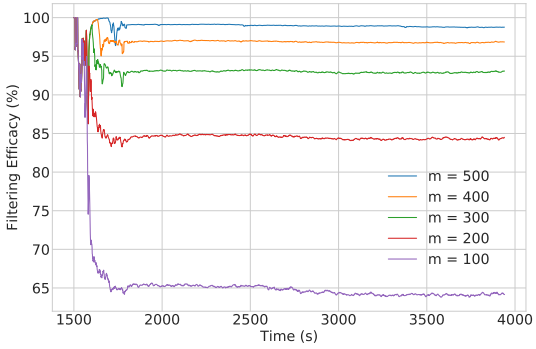


Fig. 6. **Percentage of unwanted traffic filtered by the generated rules for the CAIDA2007 attack.** Here we show the percentage of unwanted traffic filtered by our rule generation procedure using the max-coverage algorithm, as we vary the limitation on the maximum number of rules M.

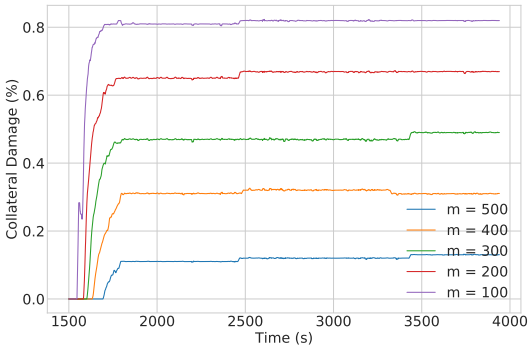


Fig. 8. **Collateral damage caused during CAIDA2007 attack.** Here we show the percentage of protected traffic flows that are filtered at each second by the min-collateral algorithm, over the course of the attack.

is considerably smaller than the CAIDA2007 attack, the attack sources show much more dynamic behavior, with over 10,000 unique attack sources participating in this attack.

In figure 14 we show the percentage of unwanted flows filtered at each second, as we vary the minimum coverage from 0 to 1. Again, this figure shows how effective each interval's rules are at filtering traffic from the next interval. As we expect, when $B = 1$, the percentage of of unwanted flows filtered remains at nearly 100% for the entire attack. Interestingly, even when the minimum coverage is set to be 0.5, on average over 60% of unwanted flows are filtered at each second. This shows that the exponentially-weighted average provides a good prediction of upcoming traffic.

Figure 15 shows the number of rules generated at each second during the attack. Generally, we see a linear trend whereby a higher value for B results in a larger set of rules. Similarly, from figure 16 we can see the mean number of rules required in order to meet the various constraints on minimum coverage.

3) Case Study 3: BOOTER2014: The final attack trace to which we apply our rule generation procedure was captured in 2014 by Santanna et al. [11] as part of a study on so-called booter services, which offer UDP-based DDoS attacks for a price, directed at the purchaser's target of choice. This attack is much shorter than the previous two, lasting only 5 minutes.

Figure 17 shows the overall dynamics of the BOOTER2014 attack. To defend against this attack, we apply the min-collateral algorithm, the goal of which is to generate a set of rules at each interval that covers as few protected flows as possible while meeting constraints on the number of rules M and the minimum coverage B.

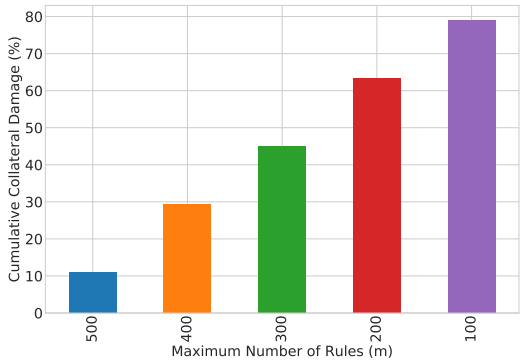


Fig. 9. **Cumulative collateral damage caused during BOOTER2014 attack.** Here we show the cumulative percentage of protected traffic flows that are filtered by the min-collateral algorithm, over the course of the attack.

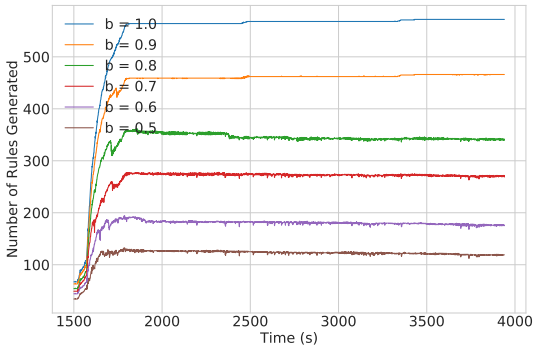


Fig. 11. **Number of rules generated over time for the CAIDA2007 attack.** Here we show the number of rules generated by the min-rules algorithm, as we vary the constraint on minimum coverage B from 50% to 100%.

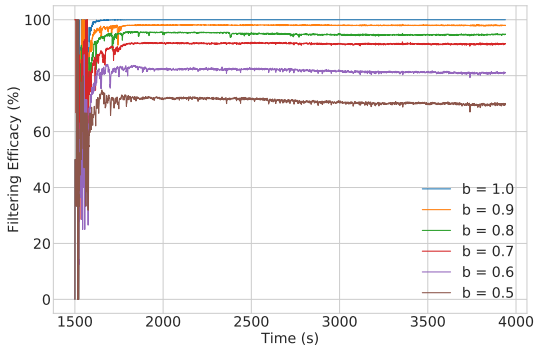


Fig. 10. **Percentage of unwanted traffic filtered by the generated rules for the CAIDA2007 attack.** Here we show the percentage of unwanted traffic that is filtered by our rule generation procedure using the min-rules algorithm, as we vary the limitation on the minimum coverage B.

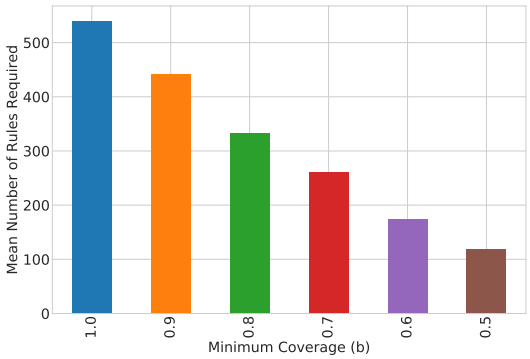


Fig. 12. **Mean number of rules required at any given time for the CAIDA2007 attack.** Here we show the mean number of rules that are required at any given time during the attack, in order min-rules algorithm to meet the constraints of B and G. Here $G = 0$ and B varies from 50% to 100%.

Here, we keep constant the constraint $B = 1$, meaning that we always require complete coverage of unwanted traffic. Meanwhile, we vary the value of M, and measure the relevant metrics of the resulting rules.

Keep in mind that although the goal of this algorithm is to minimize collateral damage, by setting $B = 1$ and M to a relatively low value, which represent quite stringent values for the constraints, we force the rule generation procedure to generate rules that cause a significant amount of collateral damage. As seen in figures 18 and 19, when $M = 100$, over 80% collateral damage is caused at times, and even when $M = 500$, the collateral damage remains around 20%.

VI. Related Work

Previous work related to this project lies across multiple topics. First of all, the following works attempt to gain a better understanding of the behavior of malicious traffic sources, including source distribution, structure, and dynamics. In [4], Kohler et al. examine the structure and distribution of IP addresses in Internet traffic. Work done by Chen et al. [1] describes the observed spatial and temporal characteristics of malicious traffic sources. In [2], Collins et al. attempt to predict the addresses of future bots, for the purpose of predictive filtering, and Ramachandran et al. [10] present a study on the network-level behavior of Internet spammers.

The second body of related work is that which addresses various strategies for filtering malicious traffic. All of these solutions, however, focus on *source-based* filtering, rather than *flow-based* filtering, which allows for more expressive traffic engineering rules. Work by Liu et al. [7] addresses the problem of network-layer filtering of traffic originating from large-scale botnets.

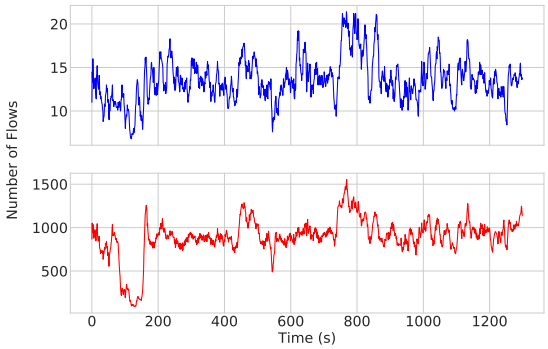


Fig. 13. **Dynamics of unwanted and protected traffic for the RADB2016 attack.** Here we show the number of both unwanted and protected traffic flows that would arrive at the victim, if no defense is present.

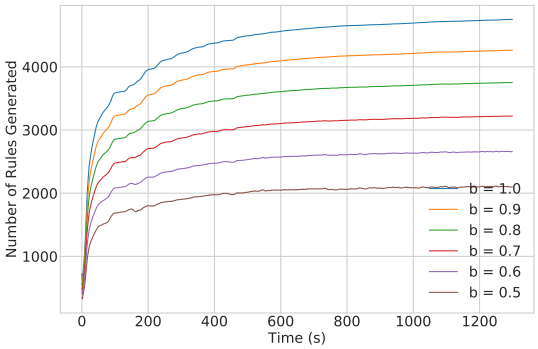


Fig. 15. **Number of rules generated over time for the RADB2016 attack.** Here we show the number of rules generated for the RADB2012 attack, as we vary the constraint on minimum coverage B from 50% to 100%.

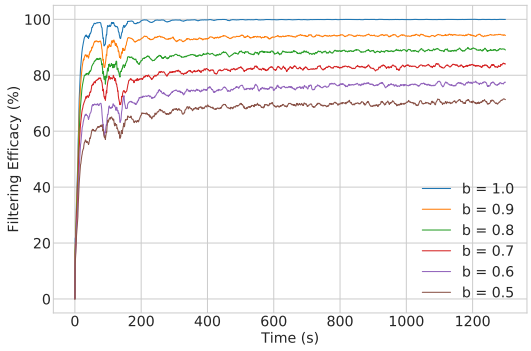


Fig. 14. **Percentage of unwanted traffic filtered by the generated rules for the RADB2016 attack.** Here we show the percentage of unwanted traffic for the RADB2012 attack that is filtered by our rule generation procedure using the min-rules algorithm, as we vary the limitation on the minimum coverage B.

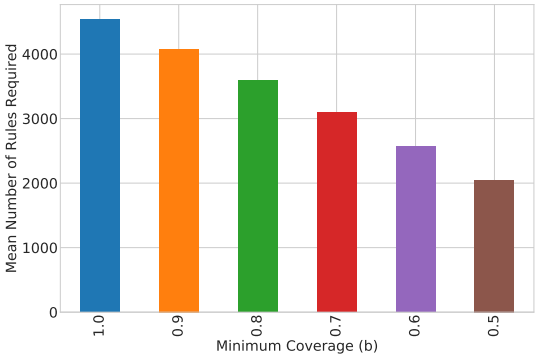


Fig. 16. **Mean number of rules required at any given time for the RADB2016 attack.** Here we show the mean number of rules that are required at any given time during the attack, in order to meet the constraints of B and G. Here $G = 0$ and B varies from 50% to 100%.

Soldo et al. offer models and algorithms for source-based filtering of malicious traffic in [13], and later expand on their work to provide an optimal source-based filtering solution in [12].

Finally, mathematically-grounded methods of representing and reasoning about IP packet headers are proposed in [6] and [3].

VII. Discussion

A. Connection with Multiply Constrained Knapsack Problem

An astute reader may recognize that the we present in section III-C four optimization problems are quite similar in nature to the 0-1 variant of the multi-dimensional knapsack problem, which is also sometimes called the **multiply constrained** knapsack problem. This is a variation on the traditional 0-1 knapsack problem where the selection of items is constrained in multiple dimensions. The 0-1 variant of this problem was shown to be NP-complete [8], and further, it has been shown that no fully polynomial-time approximation scheme exists, unless $P = NP$ [5].

The salient difference between the problems we define above and the traditional multi-dimensional knapsack problem is that in this case, there exists a heirarchical relationship between the items that may be selected. In effect, this means that the selection of one item has an impact on which other items may be selected. For example, if we consider the set from which we must choose to be the set of all possible rules (a finite set), then after choosing a rule that covers some portion $A \subset T$ of the targeted traffic, it no longer makes sense to choose any other rule that covers A, since this will not meaningfully increase coverage, and instead merely redundantly covers traffic that is already covered.

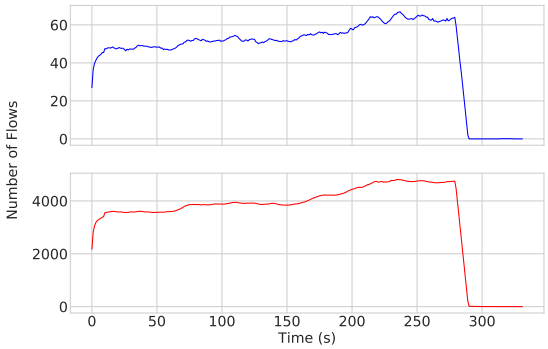


Fig. 17. **Dynamics of unwanted and protected traffic for the BOOTER2014 attack.** Here we show the number of both unwanted and protected traffic flows that would arrive at the victim, if no defense is present.

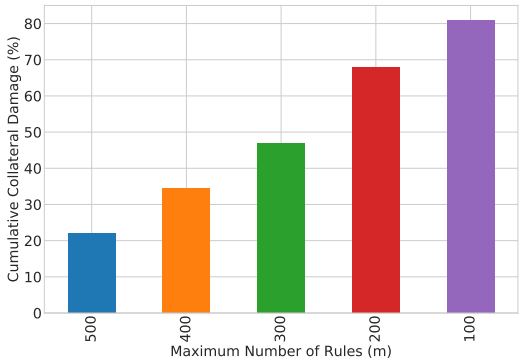


Fig. 19. **Cumulative collateral damage caused during BOOTER2014 attack.** Here we show the cumulative percentage of protected traffic flows that are filtered by the generated rules, over the course of the attack.

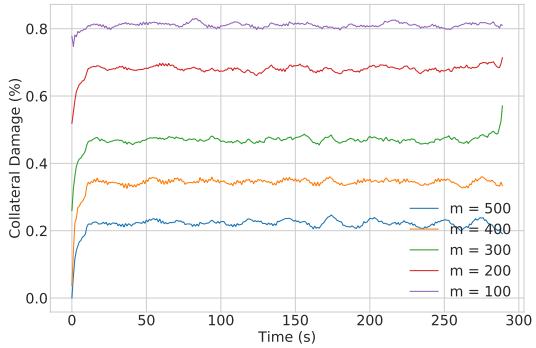


Fig. 18. **Collateral damage caused during BOOTER2014 attack.** Here we show the percentage of protected traffic flows that are filtered by the generated rules, over the course of the attack.

This difference reduces the number of valid items, thus simplifying the problem to some degree, but prevents the use of existing pseudopolynomial-time approximations.

B. Algorithmic Complexity

The three algorithms for which pseudocode is provided in figures 1, 2, and 3 share a similar structure. The main loop required for each algorithm begins with an initial set of selected nodes, and performs an aggregation at each iteration until the stopping condition is satisfied.

Assuming that the number of selected nodes is initially n, we first note that n must be less than or equal to the number of input flows, i.e., the number of elements in T that are classified as **targeted**. The maximum number of aggregations that can be performed on n nodes is then $n-1$, and thus the main loop may run n times in the worst case.

Within this main loop, it is necessary to sort the list of selected nodes. Assuming a sorting complexity of $O(nlog(n))$, this sort then dominates other terms within the loop, such as iterating through ancestors for all selected nodes (an operation that costs $O(n)$). We then have a **worst-case** complexity of $O(n^2log(n))$, where n is less than or equal to the size of the input of unwanted traffic.

Note however that the number of items to be sorted decreases by one with each iteration of the main loop. This results in an **average-case** complexity of $O(n^2)$, where n is less than or equal to the size of the input of unwanted traffic. Further, we can see trivially that the **best-case** complexity for these algorithms is merely $O(nlog(n))$, which represents the time required for a single sort of n items.

VIII. Conclusion

In this work we have devised a systematic method for the constrained generation of traffic-filtering rules for the purpose of DDoS defense. More specifically, given descriptions of the DDoS traffic, as well as descriptions of any traffic that is known or suspected to be benign, we generate a set of rules that result in dropping the DDoS traffic and not the benign, while requiring no more than a minimal amount of rule space. After formulating the problem in section III as a multi-objective optimization problem and scalarizing the problem as a set of single-objective constrained optimizations, we described in section IV a novel tree-based data structure, which we call the F-tree, as well as a set of algorithms that operate upon the F-tree to generate sets of rules that satisfy the DDoS victim's objectives, while remaining within particular constraints.

We thoroughly evaluated these algorithms, both in a static context by applying each algorithm to randomly-generated simulated DDoS attack sources, and in a dynamic context, by repeatedly applying each algorithm to

defend against simulated playbacks of real-world DDoS attack traces. Our evaluation shows that these algorithms do in fact generate rules that maximize coverage and minimize collateral damage and the size of the ruleset. Moreover, our evaluation further shows that the rules generated by our algorithms prove effective over time, even as the DDoS attack in question may display dynamic behavior.

REFERENCES

[1] Zesheng Chen, Chuanyi Ji, and Paul Barford. Spatial-temporal characteristics of internet malicious sources. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 2306–2314. IEEE, 2008.

[2] M Patrick Collins, Timothy J Shimeall, Sidney Faber, Jeff Janies, Rhiannon Weaver, Markus De Shon, and Joseph Kadane. Using uncleanliness to predict future botnet addresses. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 93–104. ACM, 2007.

[3] Peyman Kazemian, George Varghese, and Nick McKeown. Header space analysis: Static checking for networks. In *NSDI*, volume 12, pages 113–126, 2012.

[4] Eddie Kohler, Jinyang Li, Vern Paxson, and Scott Shenker. Observed structure of addresses in ip traffic. *IEEE/ACM Transactions on Networking*, 14(6):1207–1218, 2006.

[5] Bernhard Korte and Rainer Schrader. On the existence of fast approximation schemes. In *Nonlinear Programming 4*, pages 415–437. Elsevier, 1981.

[6] TV Lakshman and Dimitrios Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *ACM SIGCOMM Computer Communication Review*, volume 28, pages 203–214. ACM, 1998.

[7] Xin Liu, Xiaowei Yang, and Yanbin Lu. To filter or to authorize: Network-layer dos defense against multimillion-node botnets. In *ACM SIGCOMM Computer Communication Review*, volume 38, pages 195–206. ACM, 2008.

[8] Michael J Magazine and Maw-Sheng Chern. A note on approximation schemes for multidimensional knapsack problems. *Mathematics of Operations Research*, 9(2):244–247, 1984.

[9] Kaisa Miettinen. *Nonlinear Multiobjective Optimization*. volume 12 of International Series in Operations Research and Management Science. Springer US, 1 edition, 1999.

[10] Anirudh Ramachandran and Nick Feamster. Understanding the network-level behavior of spammers. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 291–302. ACM, 2006.

[11] J. J. Santanna, R. van Rijswijk-Deij, R. Hofstede, A. Sperotto, M. Wierbosch, L. Z. Granville, and A. Pras. Booters - an analysis of ddos-as-a-service attacks. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 243–251, May 2015.

[12] Fabio Soldo, Katerina Argyraki, and Athina Markopoulou. Optimal source-based filtering of malicious traffic. *IEEE/ACM Transactions on Networking*, 20(2):381–395, 2012.

[13] Fabio Soldo, Athina Markopoulou, and Katerina Argyraki. Optimal filtering of source address prefixes: Models and algorithms. In *INFOCOM 2009, IEEE*, pages 2446–2454. IEEE, 2009.