Optimal Source-Based Filtering of Malicious Traffic

Fabio Soldo, IEEE Student Member, Katerina Argyraki, IEEE Member, and Athina Markopoulou, IEEE Member

Abstract—In this paper, we consider the problem of blocking malicious traffic on the Internet, via source-based filtering. In particular, we consider filtering via access control lists (ACLs): these are already available at the routers today but are a scarce resource because they are stored in the expensive ternary content addressable memory (TCAM). Aggregation (by filtering source prefixes instead of individual IP addresses) helps reduce the number of filters, but comes also at the cost of blocking legitimate traffic originating from the filtered prefixes. We show how to optimally choose which source prefixes to filter, for a variety of realistic attack scenarios and operators’ policies. In each scenario, we design optimal, yet computationally efficient, algorithms. Using logs from Dshield.org, we evaluate the algorithms and demonstrate that they bring significant benefit in practice.

I. INTRODUCTION

How can we protect our network infrastructure from malicious traffic, such as scanning, malicious code propagation, spam, and distributed denial-of-service (DDoS) attacks? These activities cause problems on a regular basis, ranging from simple annoyance to severe financial, operational and political damage to companies, organizations and critical infrastructure. In recent years, they have increased in volume, sophistication, and automation, largely enabled by botnets, which are used as a fundamental building block in this effort. For example, an Internet service provider (ISP) may use filtering in response to an ongoing DDoS attack, to block the DDoS traffic before it reaches its clients. Another ISP may want to proactively identify and block traffic carrying malicious code before it reaches and compromises vulnerable hosts in the first place. In either case, filtering is a necessary operation that must be performed within the network.

Filtering capabilities are already available at routers today via access control lists (ACLs). ACLs enable a router to match a packet header against pre-defined rules and take pre-defined actions on the matching packets. They are currently used for enforcing a variety of policies, including infrastructure protection. For the purpose of blocking malicious traffic, a filter is a simple ACL rule that denies access to a source IP address or prefix. To keep up with the high forwarding rates of modern routers, filtering is implemented in hardware: ACLs are typically stored in ternary content addressable memory (TCAM), which allows for parallel access and reduces the number of lookups per forwarded packet. However, TCAM is more expensive and consumes more space and power than conventional memory. The size and cost of TCAM puts a limit on the number of filters, and this is not expected to change in the near future. With thousands or tens of thousands of filters per path, an ISP alone cannot hope to block the currently witnessed attacks, not to mention attacks from multimillion-node botnets expected in the near future.

Consider the example shown in Fig.1(a): an attacker commands a large number of compromised hosts to send traffic to a victim $V$ (say a webserver), thus exhausting the resources of $V$ and preventing it from serving its legitimate clients. The ISP of $V$ tries to protect its client by blocking the attack at the gateway router $G$. Ideally, $G$ should install one separate filter to block traffic from each attack source. However, there are typically fewer filters than attack sources, hence aggregation is used, i.e., a single filter (ACL) is used to block an entire source address prefix. This has the desired effect of reducing the number of filters necessary to block all attack traffic, but also the undesired effect of blocking legitimate traffic originating from the blocked prefixes (we will call the damage that results from blocking legitimate traffic “collateral damage”). Therefore, filter selection can be viewed as an optimization problem that tries to block as many attack sources with as little collateral damage as possible, given a limited number of filters. Furthermore, several measurement studies have demonstrated that malicious sources exhibit temporal and spatial clustering, a feature that can be exploited by prefix-based filtering.

In this paper, we formulate a general framework for studying source prefix filtering as a resource allocation problem. To the best of our knowledge, optimal filter selection has not been explored so far, as most related work on filtering has focused on protocol and architectural aspects. Within this framework, we formulate and solve five practical source-address filtering problems, depending on the attack scenario and the operator’s policy and constraints. Our contributions are twofold. On the theoretical side, filter selection optimization leads to novel variations of the multidimensional knapsack problem. We exploit the special structure of each problem, and design optimal

1 A router linecard or supervisor-engine card typically supports a single TCAM chip with tens of thousands of entries. For example, the Cisco Catalyst 4500, a mid-range switch, provides a 64,000-entry TCAM to be shared among all its interfaces (48–384). Cisco 12000, a high-end router used at the Internet core, provides 20,000 entries that operate at line-speed per linecard (up to 4 Gigabit Ethernet interfaces). The Catalyst 6500 switch can fit 16K-32K patterns and 2K-4K masks in the TCAM. Depending on how an ISP connects to its clients, each individual client can typically use only part of these ACLs, i.e., a few hundreds to a few thousands filters.
and computationally efficient algorithms. On the practical side, we provide a set of cost-efficient algorithms that can be used both by operators to block undesired traffic and by router manufacturers to optimize the use of TCAM and eventually the cost of routers. We used logs from Dshield.org to demonstrate that optimally selecting which source prefixes to filter brings significant benefits compared to non-optimized filtering or to generic clustering algorithms.

The outline of the rest of the paper is as follows. In Section II, we formulate the general framework for optimal source prefix filtering. In Section III, we study five specific problems that correspond to different attack scenarios and operator policies: blocking all addresses in a blacklist (BLOCK-ALL); blocking some addresses in a blacklist (BLOCK-SOME); blocking all/some addresses in a time-varying blacklist (TIME-VARYING BLOCK-ALL/SOME); blocking flows during a DDoS flooding attack to meet bandwidth constraints (FLOODING); and distributed filtering across several routers during flooding (DIST-FLOODING). For each problem, we design an optimal, yet computationally efficient, algorithm to solve it. In Section IV, we use data from Dshield.org to evaluate the performance of our algorithms in realistic attack scenarios and we demonstrate that they bring significant benefit in practice. Section V discusses related work and puts our work in perspective. Section VI concludes the paper.

II. PROBLEM FORMULATION AND FRAMEWORK

A. Terminology and Notation

Table I summarizes our terminology and notation.

**Source IP Addresses and Prefixes:** Every IPv4 address $ip$ is a 32-bit sequence. We use standard IP/mask notation, i.e., we write $p/l$ to indicate a prefix $p$ of length $l$ bits, where $p$ and $l$ can take values $l = 0, 1, ..., 32$ and $p = 0, 1, ..., 2^l - 1$, respectively. For brevity, when the meaning is obvious from the context, we simply write $p$ to indicate prefix $p/l$. We write $ip \in p/l$ to indicate that address $ip$ is within the $2^{32-l}$ addresses covered by prefix $p/l$.

**Blacklists and Whitelists:** A blacklist ($BL$) is a set of unique source IP addresses that send bad (undesired) traffic to the victim. Similarly, a whitelist ($WL$) is a set of unique source IP addresses that send good (legitimate) traffic to the victim.

An address may belong either to a blacklist (in which case we call it a “bad” address) or to a whitelist (in which case we call it a “good” address), but not to both. We use $|BL|$ and $|WL|$ to indicate the number of addresses in $BL$ and $WL$, respectively. For brevity, we also use $N = |BL|$ for the number of addresses in the blacklist, which is the size of the most important input to our problem.

Each address $ip$ in a blacklist or a whitelist is assigned a weight $w_{ip}$, indicating the importance of that address. If $ip$ is a bad address, we assign it a negative weight $w_{ip} \leq 0$, which indicates the benefit from blocking $ip$; if $ip$ is a good address, we assign it a positive weight $w_{ip} \geq 0$, which indicates the damage from blocking $ip$. The higher the absolute value of the weight, the higher the benefit or damage and thus the preference to block the address or not.

The weight $w_{ip}$ can have a different interpretation depending on the filtering problem. For instance, it can represent the amount of bad/good traffic originating from the corresponding source address, or it can express policy: depending on the amount of money gained/lost by the ISP when blocking source address $ip$, an ISP operator can assign large positive weights to its important customers that should never be blocked, or large negative weights to the worst attack sources that must definitely be blocked.

Creating blacklists and whitelists (i.e., identifying bad and good addresses and assigning appropriate weights to them) is a difficult problem on its own right, but orthogonal to this work. We assume that the blacklist $BL$ is provided by another module (e.g., an intrusion detection system or historical data) as input to our problem. The sources of legitimate traffic are also assumed known: e.g., web servers or ISPs typically keep historical data and know their customers. If it is not explicitly given, we take a conservative approach and define the whitelist $WL$ to include all addresses that are not in $BL$.

**Filters:** We focus on filtering of source address prefixes. In our context, a filter is an ACL rule that specifies that all packets with a source IP address in prefix $p/l$ should be blocked. $F_{\text{max}}$ is the maximum number of available filters, and it is given as input to our problem. Filter optimization is meaningful only when $F_{\text{max}}$ is much smaller than the size of the blacklist $N = |BL|$; otherwise the optimal would be to block every single
 bad address. $F_{\text{max}} << N$ is indeed the case in practice due to the size and cost of the TCAM, as mentioned in Section II.

The decision variable $x_{p/l} \in \{0,1\}$ is 1 if a filter is assigned to block prefix $p/l$; 0 otherwise. A filter $p/l$ blocks all $2^{32-l}$ addresses in that range. Hence, $b_{p/l} = |\sum_{ip \in p/l} w_{ip}|$ expresses the benefit from filter $p/l$, whereas $g_{p/l} = |\sum_{ip \in p/l} w_{ip}|$ expresses the collateral damage it causes. An effective filter should have a large benefit $b_{p/l}$ and low collateral damage $g_{p/l}$.

**Collateral Damage and Benefit:** We define the collateral damage of a filtering solution as $\sum_{p/l} \sum_{ip \in p/l} w_{ip} \cdot x_{p/l}$, i.e., the sum of the weights of the good addresses whose traffic is blocked. We define the filtering benefit as $\sum_{p/l} \sum_{ip \in p/l} w_{ip} \cdot x_{p/l}$, i.e., the sum of the weights of the bad addresses whose traffic is blocked.

### B. Rationale and Overview of Filtering Problems

Given a set of bad and a set of good source addresses ($BL$ and $WL$), a measure of their importance (the address weights $w$), and a resource budget ($F_{\text{max}}$ plus, possibly other resources, depending on the particular problem), the goal is to select which source prefixes to filter so as to minimize the impact of bad traffic and can be accommodated with the given resource budget. Different variations of the problem can be formulated, depending on the attack scenario and the victim network’s policies and constraints: the network operator may want to block all bad addresses or tolerate to leave some unblocked; the attack may be of low rate or a flooding attack; filters may be installed at one or several routers.

At the core of each filtering problem lies the following optimization problem:

$$\min_{p/l} \sum_{ip \in p/l} w_{ip} x_{p/l}$$

s.t. $\sum_{p/l} x_{p/l} \leq F_{\text{max}}$

$$\sum_{p/l} x_{p/l} \leq 1 \quad \forall p/l \in BL$$

$$x_{p/l} \in \{0,1\} \quad \forall l = 0, \ldots, 32, p = 0, \ldots, 2^l - 1$$

Eq. (1) expresses the objective to minimize the total cost of bad traffic, which consists of two parts: the collateral damage (the terms with $w_{ip} > 0$) and the cost of leaving bad traffic unblocked (the terms with $w_{ip} < 0$). We use notation $\sum_{p/l}$ to denote summation over all possible prefixes $p/l$: $l = 0, \ldots, 32, p = 0, \ldots, 2^l - 1$. Eq. (2) expresses the constraint on the number of filters. Eq. (3) states that overlapping filters are mutually exclusive, i.e., each bad address can be blocked at most once, otherwise filtering resources are wasted. Eq. (4) lists the decision variables $x_{p/l}$ corresponding to all possible prefixes, and will be omitted from now on for brevity.

Eq. (1)–(4) provide the general framework for filter-selection optimization. Different filtering problems can be written as special cases, possibly with additional constraints. As we discuss in Section III, these are all multi-dimensional knapsack problems [11], which are, in general, NP-hard. The specifics of each problem affect dramatically the complexity, which can vary from linear to NP-hard.

In this paper, we formulate five practical filtering problems and develop optimal, yet computationally efficient algorithms to solve them. Here, we summarize the rationale behind each problem and we outline our main results; the exact formulation and detailed solution is provided in Section III.

**BLOCK-ALL:** Suppose a network operator has a blacklist $BL$ of size $N$, a whitelist $WL$, and a weight assigned to each address that indicates the amount of traffic originating from that address. The total number of available filters is $F_{\text{max}}$. The first practical goal the operator may have is to install a set of filters that block all bad traffic so as to minimize the amount of good traffic that is blocked. We design an optimal algorithm that solves this problem at the lowest achievable complexity (linearly increasing with $N$).

**BLOCK-SOME:** A blacklist and a whitelist are given, as before, but the operator is now willing to block only some, instead of all, bad traffic, so as to decrease the amount of good traffic blocked at the expense of leaving some bad traffic unblocked. The goal now is to block only those prefixes that have the highest impact and do not contain sources that generate a lot of traffic, so as to minimize the total cost in Eq. (1). We design an optimal, lowest-complexity (linearly increasing with $N$) algorithm for this problem, as well.

**TIME-VARYING BLOCK-ALL (SOME):** Bad addresses may change over time [4]: new sources may send malicious traffic and conversely, previously active sources may disappear (e.g., when their vulnerabilities are patched). One way to solve the dynamic versions of BLOCK-ALL (SOME) is to run the algorithms we propose for the static versions for the blacklist/whitelist pair at each time slot. However, given that subsequent blacklists typically exhibit significant overlap [4], it may be more efficient to exploit this temporal correlation and incrementally update the filtering rules. We show that it is possible to update the optimal solution, as new IPs are inserted in or removed from the blacklist, in $\log N$ time.

**FLOODING:** In a flooding attack, such as the one shown in Fig. 1, a large number of compromised hosts send traffic to the victim and exhaust the victim’s access bandwidth. In this case, our framework can be used to select the filtering rules that minimize the amount of good traffic that is blocked while meeting the access bandwidth constraint – in particular, the total bandwidth consumed by the unblocked traffic should not
exceed the bandwidth of the flooded link, e.g., link G-V in Fig. 1. We prove that this problem is NP-hard and we design a pseudo-polynomial algorithm that solves it optimally, with complexity that grows linearly with the blacklist and whitelist size, i.e., $|\mathcal{B}| + |\mathcal{W}|$.

**DIST-FLOODING**: All the above problems aim at installing filters at a single router. However, a network operator may use the filtering resources collaboratively across several routers to better defend against an attack. Distributed filtering may also be enabled by the cooperation among several ISPs against a common enemy. The question in both cases is not only which prefixes to block, but also at which router to install the filters. We study the practical problem of distributed filtering against a flooding attack. We prove that the problem can be decomposed into several FLOODING problems, which can be solved in a distributed way.

III. FILTERING PROBLEMS AND ALGORITHMS

In this section, we provide the detailed formulation of each problem and we present the algorithm that solves it. We start by defining the data structure that we use to represent the problem and to develop our algorithms.

A. A Data Structure for Representing Filtering Solutions

**Definition 1 (LCP Tree)**: Given a set of addresses $\mathcal{A}$, we define the *Longest Common Prefix* tree of $\mathcal{A}$, denoted by LCP-tree($\mathcal{A}$), as the binary tree with the following properties: (i) each leaf represents a different address in $\mathcal{A}$ and there is a leaf for each address in $\mathcal{A}$; (ii) each intermediate (non-leaf) node represents the longest common prefix between the prefixes represented by its two children.

The LCP-tree($\mathcal{A}$) can be constructed from the complete binary tree (with root leaves at level 32 corresponding to all addresses $[0, ..., 2^{32} - 1]$, and intermediate nodes at level $i = 1, ..., 32$ corresponding to all prefixes of length $i$) by removing the branches that do not have addresses in $\mathcal{A}$, and then by removing nodes with a single child. It is a variation of the binary (or unibit) trie [12], but does not have nodes with a single child. The LCP-tree($\mathcal{A}$) offers an intuitive way to represent sets of prefixes that can block the addresses in set $\mathcal{A}$: each node in the LCP tree represents a prefix that can be blocked, hence we can represent a filtering solution as the pruned version of the LCP tree, whose leaves are all and only the blocked prefixes.

**Example 1**: For instance, consider the LCP tree depicted in Figure 2 whose leaves correspond to bad addresses that we want to block. One (expensive) solution is to use one filter to block each bad address; thus the LCP tree is not pruned and its leaves correspond to the filters. Another feasible solution is to use three filters and block traffic from prefixes 0/1, 8/2, and 12/4; this can be represented by the pruned version of the LCP tree that includes the aforementioned prefixes as leaves. Yet another (rather radical) solution is to filter a single prefix (0/0) to block all traffic; this can be represented by the pruned version of the LCP tree that includes only its root.

![Figure 2. Example of LCP-tree(\mathcal{B}). Consider a blacklist consisting of the following 9 bad addresses: \mathcal{B} = \{10.0.0.1, 10.0.0.3, 10.0.0.4, 10.0.0.5, 10.0.0.7, 10.0.0.8, 10.0.0.10, 10.0.0.11, 10.0.0.12\}. All remaining addresses are considered good. Each leaf represents one address in the \mathcal{B}. Each intermediate node represents the longest common prefix representing all bad addresses in that subtree. At each intermediate node $p$, we also show the branches that do not have addresses in $\mathcal{B}$; do not have nodes with a single child. The LCP-tree(\mathcal{A}) offers an intuitive way.

Complexity: Given a list of addresses $\mathcal{A}$, we can build the LCP-tree($\mathcal{A}$) by performing $|\mathcal{A}|$ insertions in a Patricia trie [12]. To insert a string of $m$ bits, we need at most $m$ comparisons. Thus, the worst-case complexity is $O(m|\mathcal{A}|)$, where $m = 32$ (bits) is the length of a 32-bit IPv4 address.

B. BLOCK-ALL

**Problem Statement**: Given a blacklist $\mathcal{B}$, a whitelist $\mathcal{W}$, and the number of available filters $F_{\text{max}}$, select filters that block all bad traffic and minimize collateral damage.

**Formulation**: We formulate this problem by making two adjustments to the general framework of Eq. (1)–(4). First, Eq. (1) becomes Eq. (5) below, which expresses the goal to minimize the collateral damage. Second, Eq. (5) becomes Eq. (7) below, which enforces the constraint that every bad address should be blocked by exactly one filter, as opposed to at most one filter in Eq. (5).

$$\min \sum_{p/l} g_{p/l} x_{p/l}$$

subject to

$$\sum_{p/l} x_{p/l} \leq F_{\text{max}}$$

$$\sum_{p/l} x_{p/l} = 1 \quad \forall ip \in \mathcal{B}$$

**Characterizing an Optimal Solution**: Our algorithm starts from LCP-tree($\mathcal{B}$) and outputs a pruned version of that LCP tree. Hence, we start by proving that an optimal solution to BLOCK-ALL can indeed be represented as a pruned version of that LCP tree.

**Proposition 3.1**: An optimal solution to BLOCK-ALL can be represented as a pruned subtree of LCP-tree($\mathcal{B}$) with the same root as LCP-tree($\mathcal{B}$), up to $F_{\text{max}}$ leaves, and each non-leaf node having exactly two children.
Proof: We prove that, for each feasible solution to BLOCK-ALL \( S \), there exists another feasible solution \( S' \) that (i) can be represented as a pruned subtree of LCP-tree(\( BL \)) as described in the proposition and (ii) whose collateral damage is smaller or equal to \( S \)’s. This is sufficient to prove the proposition, since an optimal solution is also a feasible one.

Any filtering solution can be represented as a pruned subtree of the full binary tree of all IP addresses (LCP-tree(\( \{0, 1, ..., 2^{32} - 1\} \)) with the same root and leaves corresponding to the filtered prefixes. \( S \) is a feasible solution to BLOCK-ALL, therefore \( S \) uses up to \( F_{\text{max}} \) filters, i.e., its tree has up to \( F_{\text{max}} \) leaves. Indeed, if this was not the case, Eq. (4) would be violated and \( S \) would not be a feasible solution.

Let us assume that the tree representing \( S \) includes a prefix \( \tilde{p} \) that is not in LCP-tree(\( BL \)). There are three possible cases:

1) \( \tilde{p} \) includes no bad addresses. In this case, we can simply remove \( \tilde{p} \) from \( S \)’s tree (i.e., unblock \( \tilde{p} \)).
2) Only one of \( \tilde{p} \)’s children includes bad addresses. In this case, we can replace \( \tilde{p} \) with the child node.
3) Both of \( \tilde{p} \)’s children contain bad addresses. In this case, \( \tilde{p} \) is already the longest common prefix of all bad addresses in \( BL \cap \tilde{p} \), thus already on the LCP-tree(\( BL \)).

Clearly, each of these operations transforms feasible solution \( S \), which is assumed not to be on LCP-tree(\( BL \)), into another feasible solution \( S' \) with smaller or equal collateral damage but on the LCP-Tree(\( BL \)). We can repeat this process for all prefixes that are in \( S \)’s tree but not in LCP-tree(\( BL \)), until we create a feasible solution \( S' \) that includes only prefixes from LCP-tree(\( BL \)) and has smaller or equal collateral damage.

The only element missing to prove the proposition is to show that, in the pruned LCP subtree that represents \( S' \), each non-leaf node has exactly two children. We show this by contradiction: Suppose there exists a non-leaf node in our pruned LCP subtree that has exactly one child. This can only result from pruning out one child of a node in the LCP tree. This means that all the bad addresses (leaves) in the subtree of this child node remain unfiltered, which violates Eq. (7); but this is a contradiction because \( S' \) is a feasible solution.

Algorithm 1 Algorithm for solving BLOCK-ALL

1: build LCP-tree(\( BL \))
2: for all leaf nodes \( \text{leaf} \) do
3: \( z_{\text{leaf}}(F) = 0 \forall F \in [1, F_{\text{max}}] \)
4: \( X_{\text{leaf}}(F) = \{\text{leaf}\} \forall F \in [1, F_{\text{max}}] \)
5: end for
6: level = level(leaf)-1
7: while level \( \geq \) level(root) do
8: \( z_{\text{leaf}}(p) \) such that level(p) \( = \) level do
9: \( z_p(1) = g_p \)
10: \( X_p(1) = \{p\} \)
11: \( z_p(F) = \min_{n=1,...,F-1} \{z_{s_l}(F-n) + z_{s_r}(n)\} \forall F \in [2, F_{\text{max}}] \)
12: \( X_p(F) = X_{s_l}(F-n) \cup X_{s_r}(n) \forall F \in [2, F_{\text{max}}] \)
13: end for
14: level = level - 1
15: end while
16: return \( z_{\text{root}}(F_{\text{max}}) \), \( X_{\text{root}}(F_{\text{max}}) \)

with boundary conditions for leaf and intermediate nodes:

\[
\begin{align*}
z_{\text{leaf}}(F) &= 0 \quad \forall F \geq 1 \quad (9) \\
z_p(1) &= g_p \quad \forall p \quad (10)
\end{align*}
\]

Once we compute \( z_p(F) \) for all prefixes in the LCP tree, we simply read the value of the optimal solution, \( z_{\text{root}}(F_{\text{max}}) \).

We also use auxiliary variables \( X_p(F) \) to keep track of the set of prefixes used in the optimal solution. In lines 4 and 10 of Algorithm 1, \( X_p(F) \) is initialized to the single prefix used. In line 12, after computing the new cost, the corresponding set of prefixes is updated: \( X_p(F) = X_{s_l}(F-n) \cup X_{s_r}(n) \).

Theorem 3.2: Algorithm 1 computes the optimal solution of problem BLOCK-ALL: the prefixes that are contained in set \( X_p(F) \) are the optimal \( z_p(F) = 1 \) for Eq. (5)-(7).

Proof: Recall that, \( z_{\text{root}}(F_{\text{max}}) \) denotes the value of the optimal solution of BLOCK-ALL with \( F_{\text{max}} \) filters (i.e., the minimum collateral damage), while \( X_{\text{root}}(F_{\text{max}}) \) denotes the set of filters selected in the optimal solution. Let \( s_l \) and \( s_r \) denote the two children nodes (prefixes) of \( \text{root} \) in the LCP-tree(\( BL \)). Finding the optimal allocation of \( F_{\text{max}} > 1 \) filters to block all addresses contained in \( \text{root} \) (possibly all IP space), is equivalent to finding the optimal allocation of \( x \geq 1 \) filters to block all addresses in \( s_l \) and \( y \geq 1 \) prefixes for bad addresses in \( s_r \), such that \( x + y = F_{\text{max}} \). This is because prefixes \( s_l \) and \( s_r \) jointly contain all bad addresses. Moreover, each of \( s_l \) and \( s_r \) contains at least one bad address. Thus, at least one filter must be assigned to each of them. If \( F_{\text{max}} = 1 \), i.e., there is only one filter available, the only feasible solution is to select \( \text{root} \) as the prefix to filter out. The same argument recursively applies to descendant nodes, until either we reach a leaf node, or we have only one filter available. In these cases, the problem is trivially solved by Eq. (9).

Complexity: The LCP-tree is a binary tree with \( |BL| \) leaves; therefore, it has \( O(|BL|) \) intermediate nodes (prefixes). Computing Eq. (8) for every node \( p \) and for every value \( F \in [1, F_{\text{max}} - 1] \) involves solving \( O(|BL|F_{\text{max}}) \) subproblems, one for every pair \((p, F)\) with complexity \( O(F_{\text{max}}) \). \( z_p(F) \) in Eq. (8) requires only the optimal solution at the sibling nodes, \( z(s_l, F-n), z(s_r, n) \). Thus, proceeding from the leaves to the root, we can compute the optimal solution in \( O(|BL|F_{ \text{max}}^2) \). In practice, the complexity is even lower, since
we do not need to compute \( z_p(F) \) for all values \( F \leq F_{\max} \), but only for \( F \leq \min\{|leaves(p)|, F_{\max}\} \), where \(|leaves(p)|\) is the number of the leaves in prefix \( p \) in the LCP tree. Moreover, we only need to compute entries \( z_p(F) \) for every prefix \( p \), s.t. we cover all addresses in \( BL \cap p \), which may require \( F \leq F_{\max} \) for long prefixes in the LCP-tree.

Finally, we observe that the asymptotic complexity is \( O(|BL|) \), since \( F_{\max} \ll N = |BL| \) and \( F_{\max} \) does not depend on \( |BL| \) but only on the TCAM size. Thus, the time complexity increases linearly with the number of bad addresses \( |BL| \). This is within a constant factor of the lowest achievable complexity, since we need to read all \( |BL| \) bad addresses at least once.

C. BLOCK-SOME

Problem Statement: Given a blacklist \( BL \), a whitelist \( WL \), and the number of available filters \( F_{\max} \), the goal is to select filters so as to minimize the total cost of the attack.

Formulation: This is precisely the problem described by Eq. 1–4, but put slightly rephrased to better compare it with BLOCK-ALL. There are two differences from BLOCK-ALL. First, the goal is to minimize the total cost of the attack, which involves both collateral damage \( g_p/F \) and the filtering benefit \( b_{p/l} \), which is expressed by Eq. (11). Second, Eq. (12) states that every bad address must be filtered by at most one prefix, which means that it may or may not be filtered.

\[
\begin{align*}
\min \sum_{p/l} \left( g_{p/l} - b_{p/l} \right) x_{p/l} \\
\text{s.t.} \sum_{p/l} x_{p/l} \leq F_{\max} \\
\sum_{p/l:ip\in p/l} x_{p/l} \leq 1 \quad \forall ip \in BL
\end{align*}
\] (11)

Characterizing an Optimal Solution: As with BLOCK-ALL, our algorithm starts from LCP-tree(\( BL \)) and outputs a pruned version of that LCP tree. The only difference is that some bad addresses may now remain unfiltered. In the pruned LCP subtree that represents our solution, this means that there may exist intermediate (non-leaf) nodes with a single child.

Proposition 3.3: An optimal solution to BLOCK-SOME can be represented as a pruned subtree of LCP-tree(\( BL \)) with: the same root as LCP-tree(\( BL \)) and up to \( F_{\max} \) leaves.

Proof: In Proposition 3.1 we proved that any solution of Eq. 5–6 can be reduced to a (pruned) subtree of the LCP tree with at most \( F_{\max} \) leaves. Moreover, the constraint expressed by Eq. 13, which imposes the use of non-overlapping prefixes, is automatically imposed considering the leaves of the pruned subtree as the selected filter. This proves that any feasible solution of BLOCK-SOME can be represented as a pruned subtree of the LCP tree with at most \( F_{\max} \) leaves. And thus, so can an optimal solution.

Algorithm: The algorithm that solves BLOCK-SOME is similar to Algorithm 1 in that it relies on the LCP tree and a dynamic programming (DP) approach. The main difference is that not all bad addresses need to be filtered, hence, at each step, we can assign \( n = 0 \) filters to the left and/or right subtree. More specifically, whereas in line (11) of Algorithm 1 we had \( n = 1, \ldots, F - 1 \), now we have \( n = 0, 1, \ldots, F \). We can recursively compute the optimal solution as before:

\[
z_p(F) = \min_{n=0,\ldots,F} \left\{ z_{n}(F - n) + z_{s_r}(n) \right\}
\] (14)

with boundary conditions

\[
\begin{align*}
z_p(0) &= 0 \quad \forall p \\
z_p(1) &= \min \left\{ g_p - b_p, \min_{n=0,1} \left\{ z_{n}(1 - n) + z_{s_r}(n) \right\} \right\}
\] (15)

\[
z_{\text{leaf}}(F) = -b_{\text{leaf}} \quad \forall F \geq 1
\] (17)

where \( p \) is an intermediate node (prefix) and \( \text{leaf} \) is a leaf node in the LCP-tree.

Complexity: The analysis of Algorithm 1 applies to this algorithm as well. The complexity is the same, i.e., linearly increasing with \( |BL| \).

BLOCK-ALL vs. BLOCK-SOME: There is an interesting connection between the two problems. The latter can be regarded as an automatic way to select the best subset from \( BL \) and run BLOCK-ALL only on that subset. If the absolute value of weights of bad addresses are significantly larger than the weights of the good addresses, then BLOCK-SOME degenerates to BLOCK-ALL.

D. TIME-VARYING BLOCK-ALL(SOME)

We now consider that the blacklist and whitelist change over time and we seek to incrementally update the filtered prefixes that are affected by the change. More precisely, consider a sequence of blacklists, \( \{BL_{\tau_0}, BL_{\tau_1}, \ldots\} \) and of whitelists, \( \{WL_{\tau_0}, WL_{\tau_1}, \ldots\} \) at times \( \tau_0, \tau_1, \ldots \), respectively.
**Problem Statement:** Given: (i) a blacklist and whitelist, $BL_{\tau_{i-1}}$ and $WL_{\tau_{i-1}}$, (ii) the number of available filters $F_{\text{max}}$, (iii) the corresponding solution to BLOCK-ALL(SOME), denoted by $S_{\tau_{i-1}}$, and (iv) another blacklist and whitelist, $BL_{\tau_{i}}$ and $WL_{\tau_{i}}$; obtain the solution to BLOCK-ALL(SOME) for the second blacklist/whitelist, denoted by $S_{\tau_{i}}$.

**Algorithm:** Consider, for the moment, that the whitelist remains the same and focus on the changes in the blacklist.

(i) **Addition.** First, consider that the two blacklists differ only in a single new bad address, which does not appear in $BL_{\tau_{i-1}}$, but appears in $BL_{\tau_{i}}$. There are two cases, depending on whether the new bad address belongs to a prefix that is already filtered in $S_{\tau_{i-1}}$. If it is, no further action is needed, and $S_{\tau_{i}} = S_{\tau_{i-1}}$. Otherwise, we modify the LCP tree that represents $S_{\tau_{i-1}}$ to also include the new bad address, as illustrated in Fig. 2. The key point is that we only need to add one new intermediate node to the LCP tree (the gray node in Fig. 2), corresponding to the longest common prefix between the new bad address and its closest bad address that is already in the LCP tree. The optimal allocation of $F$ filters to the subtree rooted at prefix $p$ depends only on how these $F$ filters are allocated to the children of $p$. Hence, when we add a new node to the LCP tree, we need to recompute the optimal filter allocation (i.e., recompute $z_p(F)$ and $X_g(F)$ for $F$, according to Eq. (3)) for all and only the predecessors of the new node, all the way up to the root node.

(ii) **Deletion.** Then assume that two blacklists differ in one deleted bad address, which appears in $BL_{\tau_{i-1}}$ but not in $BL_{\tau_{i}}$. In this case, we modify the LCP tree that represents $S_{\tau_{i-1}}$ to remove the leaf node that corresponds to that address as well as its parent node (since that node does not have two children any more), and we recompute the optimal filter allocation for all and only the node's predecessors.

(iii) **Adjustment.** Finally, suppose that the two blacklists differ in one address, which appears in both blacklists but with different weights; or, that the two blacklists are the same, while the two whitelists differ in one address (it either appears in one of the two whitelists or it appears in both whitelists but with different weights). In all of these cases, we do not need to add or remove any nodes from the LCP tree, but we do need to adjust the collateral damage or filtering benefit associated with one node, hence recompute the optimal filter allocation for all and only that node’s predecessors.

(iv) **Multiple addresses.** If the two successive time instances differ in multiple addresses, we repeat the procedures described above as needed, i.e., we perform one node addition for each new bad address, one deletion for each removed bad address, and up to one adjustment for each other difference.

**Complexity:** Since the LCP tree is a complete binary tree, any leaf node has at most $\log(|BL|)$ predecessors, so, inserting a new bad address (or removing one) requires $O(\log(|BL|)F_{\text{max}}^2)$ operations. Hence, deriving $S_{\tau_{i}}$ from $S_{\tau_{i-1}}$ as described above is asymptotically better than computing it from scratch using Algorithm 1 if and only if the number of different addresses between the two time instances is less than $\frac{|BL|}{\log(|BL|)}$.

### E. FLOODING

**Problem Statement:** Given: (i) a blacklist $BL$ and a whitelist $WL$, where the absolute weight of each bad and good address is equal to the amount of traffic it generates, (ii) the number of available filters $F_{\text{max}}$, and (iii) a constraint on the victim’s link capacity (bandwidth) $C$; select filters so as to minimize collateral damage and make the total traffic fit within the victim’s link capacity.

**Formulation:** To formulate this problem, we need to make two adjustments to the general framework of Eq. (1–4). First, Eq. (1) becomes Eq. (13), which expresses the goal to minimize collateral damage. Second, we add a new constraint Eq. (20), which specifies that the total traffic that remains unblocked after filtering (which is the total traffic, $T_0 = \sum_{p/l} w_{ip}$, minus the traffic that gets blocked, $\sum_{p/l} (g_{p/l} + b_{p/l}) x_{p/l}$) should fit within the link capacity $C$, so as to avoid congestion and packet loss.

$$\min \sum_{p/l} g_{p/l} x_{p/l} \quad \text{s.t.} \quad \sum_{p/l} x_{p/l} \leq F_{\text{max}}$$

$$T_0 - \sum_{p/l} (g_{p/l} + b_{p/l}) x_{p/l} \leq C$$

$$\sum_{p/l: ip \in BL} x_{p/l} \leq 1 \quad \forall ip \in BL$$

**Characterizing an Optimal Solution:** We represent the optimal solution as a pruned subtree of an LCP-tree. However, we start with the full binary tree of all bad and good addresses LCP-tree($BL \cup WL$). Moreover, to handle the constraint in Eq. (20), each node corresponding to prefix $p$ is assigned an additional cost, $T_p$, indicating the total amount of traffic sent by $p$, $T_p = g_p + b_p$.

**Proposition 3.4:** An optimal solution of FLOODING can be represented as the leaves of a pruned subtree of LCP-tree($BL \cup WL$), with the same root, up to $F_{\text{max}}$ leaves, and total cost of the leaves $\geq T_0 - C$.

**Proof:** Similarly to Proposition 3.1 we prove that for every feasible solution to FLOODING $S$, there exists another feasible solution $S'$, which (i) can be represented as a pruned subtree of LCP-tree($BL \cup WL$) as described in the proposition and (ii) whose collateral damage is smaller or equal to $S$‘s. This is sufficient to prove the proposition, since an optimal solution is also a feasible one.

Any filtering solution can be represented as a pruned subtree of LCP-tree($BL \cup WL$) with the same root and leaves corresponding to the filtered prefixes. $S$ is a feasible solution to FLOODING, therefore: $S$‘s tree has up to $F_{\text{max}}$ leaves, otherwise Eq. (12) would be violated; and the total cost of $S$‘s leaves is $\geq T_0 - C$, otherwise Eq. (20) would be violated.

Suppose that $S$ includes a prefix $\hat{p}$ that is not in LCP-tree($BL \cup WL$). We can construct a better feasible solution $S'$, which can be represented as a pruned subtree of LCP-tree($BL \cup WL$): $S'$ has the same root, up to $F_{\text{max}}$ leaves and total cost of the leaves $\geq T_0 - C$. There are three possibilities: 1) $\hat{p}$ includes neither bad nor good addresses. In this case, we can simply remove $\hat{p}$ from $S$, i.e., unblock $\hat{p}$. 
2) Only one of \( \tilde{p} \)'s children includes bad or good addresses. In this case, we can replace \( \tilde{p} \) with the child that contains the bad addresses.

3) Both of \( \tilde{p} \)'s children include bad or good addresses. In this case, \( \tilde{p} \) is already a longest common prefix and we do not need to do anything.

Clearly, each of these operations transforms feasible solution \( S \) into another feasible solution with smaller or equal collateral damage while still preserving the capacity constraint. This is because the transformations filter the same amount of traffic, just using the longest prefix possible to do so. We can repeat this process for all prefixes that are in \( S \) but not in LCP-tree(\( BL \cup WL \)), until we create a feasible solution \( S' \) that includes only prefixes from LCP-tree(\( BL \cup WL \)) and has smaller or equal collateral damage.

**Theorem 3.5:** FLOODING (i.e., Eq. (18)–(21)) is NP-Hard.

**Proof:** To prove that FLOODING is \( \mathcal{NP} \)-hard, we consider the knapsack problem with a cardinality constraint:

\[
\max_{i \in N} \sum_{i \in N} p_i x_i \quad (22)
\]

\[
\sum_{i \in N} x_i = k \quad (23)
\]

s.t. \[
\sum_{i \in N} w_i x_i \leq C_1 \quad (24)
\]

which is known to be \( \mathcal{NP} \)-hard [11] and we show that it reduces to FLOODING. To do this, we put FLOODING in a slightly different form, by making two changes.

First, we change the inequality in Eq. (19) to an equality. Any feasible solution to FLOODING that uses \( F < F_{max} \) filters can be transformed to another feasible solution with exactly \( F_{max} \) filters, without increasing collateral damage. In fact, given a feasible solution \( S \) that uses \( F < F_{max} \) filters, as long as \( F < |BL| \), it is always possible to remove a filter from a prefix \( p \) and add two filters to the two prefixes corresponding to \( p \)'s children in LCP-tree(\( BL \cup WL \)). The solution constructed this way uses \( F + 1 \) filters, blocks all addresses blocked in \( S \), and has a cost less or equal to \( S \)'s.

Second, we define variables \( \bar{x}_{p/l} = -x_{p/l} \), \( \bar{F}_{max} = -F_{max} \) and \( C = T_0 - C \) and use them to rewrite FLOODING:

\[
\max_{\lambda \geq 0} \sum_{p/l} \left( g_{p/l} x_{p/l} \right) \quad (25)
\]

s.t. \[
\sum_{p/l} \bar{x}_{p/l} = \bar{F}_{max}, \quad (26)
\]

\[
\sum_{p/l} \left( g_{p/l} + b_{p/l} \right) \bar{x}_{p/l} \leq \bar{C} \quad (27)
\]

\[
\sum_{p/l : \tilde{p} \in p/l} -\bar{x}_{p/l} \leq -1 \quad \forall \tilde{p} \in BL \quad (28)
\]

For a given instance of the problem defined by Eq. (22)–(23), we construct an equivalent instance of the problem defined by Eq. (25)–(28) by introducing the following mapping. For \( ip \in BL \cup WL \): \( g_{ip} = p_i, (g_{ip} + b_{ip}) = w_i \). For all other prefixes \( p/l \) that are not addresses in the blacklist or whitelist: \( (g_{p/l} + b_{p/l}) = \bar{C} + 1 \). Moreover, we assign \( \bar{F}_{max} = k \) and \( \bar{C} = C_1 \). With this assignment, a solution to the problem defined by Eq. (22) can be obtained by solving FLOODING, then taking the values of variables \( x_{p/l} \) that are blocked.

**Algorithm:** Given the hardness of the problem, we do not look for a polynomial-time algorithm. We design a pseudo-polynomial-time algorithm that optimally solves FLOODING. Its complexity is linearly with the number of good and bad addresses and with the magnitude of \( C_{max} \).

Our algorithm is similar to the one that solves BLOCK-SOME, i.e., it relies on an LCP tree and a DP approach. However, we now use the LCP tree of all the bad and good addresses. Moreover, when we compute the optimal filter allocation for each subtree, we now need to consider not only the number of filters allocated to that subtree, but also the corresponding amount of capacity (i.e., the amount of the victim's capacity consumed by the unfiltered traffic coming from the corresponding prefix). We can recursively compute the optimal solution bottom-up as before:

\[
z_{p}(F,c) = \min_{n=0 \ldots, c} \left\{ z_{s_i}(F - n, c - m) + z_{s_i}(n, m) \right\} \quad (29)
\]

where \( z_{p}(F,c) \) is the minimum collateral of prefix \( p \) when allocating \( F \) filters and capacity \( c \) to that prefix.

**Complexity:** Our DP approach computes \( O(CF_{max}) \) entries for every node in LCP-tree(\( BL \cup WL \)). Moreover, the computation of a single entry, given the entries of descendant nodes, requires \( O(CF_{max}) \) operations, Eq. (22). We can leverage again the observation that we do not need to compute \( CF_{max} \) entries for all nodes in the LCP tree: At a node \( p, \) it is sufficient to compute Eq. (29) only for \( c = 0, \ldots, C = \min\{C, \sum_{ip \in p/l} w_{ip} \} \leq C \) and \( f = 0, \ldots, F = \max\{F_{max}, |leaves(p)|\} \). Therefore, the optimal solution to FLOODING, \( z_{root}(F_{max}, C) \), can be computed in \( O((|BL| + |WL|)^2) \) time. This is increasing linearly with the number of addresses in \( BL \cup WL \) and is polynomial in \( C \). The overall complexity is pseudo-polynomial because \( C \) cannot be polynomially bounded in the input size. In the evaluation section, we present a heuristic algorithm that operates in increments \( \Delta C \) of \( C \). Finally, we note that \( F_{max} << C \) and thus \( F_{max} \) does not appear in the asymptotic complexity.

**BLOCK-SOME vs. FLOODING:** There is an interesting connection between the two problems. To see that, consider the partial Lagrangian relaxation of Eq. (18)–(21):

\[
\max_{\lambda \geq 0} \left\{ \min_{p/l} \left[ (1 - \lambda)g_{p/l} - \lambda b_{p/l} \right] x_{p/l} + \sum_{p/l} \lambda T_{0} - \lambda C \right\} \quad (30)
\]

\[
s.t. \sum_{p/l} x_{p/l} \leq F_{max} \quad (31)
\]

\[
s.t. \sum_{p/l} x_{p/l} \leq 1 \quad \forall ip \in BL \quad (32)
\]

For every fixed \( \lambda \geq 0 \), Eq. (30)–(32) are equivalent to Eq. (11)–(13) for a specific assignments of weights \( w_{ip} \). This shows that dual feasible solutions of FLOODING are instances of BLOCK-SOME for a particular assignment of weights. The dual problem, in the variable \( \lambda \), aims exactly at tuning the Lagrangian multiplier to find the best assignment of weights.
F. DISTRIBUTED-FLOODING

Problem Statement: Consider a victim V that connects to the Internet through its ISP and is flooded by a set of attackers listed in a blacklist BL, as in Fig.1(a). To reach the victim, attack traffic passes through one or more ISP routers. Let R be the set of unique such routers. Let each router \( u \in R \) have capacity \( C(u) \) on the downstream link (towards V) and a limited number of filters \( F_{\text{max}}(u) \). The volume of good/bad traffic through every router is assumed known. Our goal is to allocate filters on some or all routers, in a distributed way, so as to minimize the total collateral damage and avoid congestion on all links of the ISP network.

Formulation. Let the variables \( x_{p/l}^{(u)} \in \{0, 1\} \) indicate whether or not filter \( p/l \) is used at router \( u \). Then the distributed filtering problem can be stated as:

\[
\min \sum_{u \in R} \sum_{p/l} \left( g_{p/l}^{(u)} + \lambda_p^{(u)} \right) x_{p/l}^{(u)} \quad \forall u \in R \quad (33)
\]

s.t. \( \sum_{p/l} x_{p/l}^{(u)} \leq F_{\text{max}}^{(u)} \quad \forall u \in R \quad (34) \)

\[
T_0^{(u)} - \sum_{p/l} \left( g_{p/l}^{(u)} + h_{p/l}^{(u)} \right) x_{p/l}^{(u)} \leq C^{(u)} \quad \forall u \in R \quad (35)
\]

\[
\sum_{u \in R} \sum_{p/l \in BL} x_{p/l}^{(u)} \leq 1 \quad \forall ip \in BL \quad (36)
\]

Characterizing an Optimal Solution. Given the sets BL, WL, R, and \( F_{\text{max}}^{(u)}, C^{(u)} \) at each router, we have:

Proposition 3.6: There exists an optimal solution of DIST-FLOODING that can be represented as a set of \(|R|\) different pruned subtrees of the LCP-tree \((BL \cup WL)\), each corresponding to a feasible solution of FLOODING for the same input, and s.t. every subtree leaf is not a node of another subtree.

Proof. Feasible solutions of DIST-FLOODING allocate filters on different routers s.t. Eq.\((34) \) and \((35) \) are satisfied independently at every router. In the LCP tree, this means having \(|R|\) subtrees, one for every router, each having at most \( F_{\text{max}}^{(u)} \) leaves and associated blocked traffic \( \geq T_0^{(u)} - C^{(u)} \), where \( T_0^{(u)} \) is the total incoming traffic at router \( u \). Each subtree can be thought as a feasible solution of a FLOODING problem. Eq.\((36) \) ensures that the same address is not filtered multiple times at different routers, to avoid waste of filters. In the LCP-tree, this translates into every leaf appearing at most in one subtree.

Algorithm. Constraint \((36) \), which imposes that different routers do not block the same prefixes, prevents us from a direct decomposition of the problem. To decouple the problem, consider the following partial Lagrangian relaxation:

\[
L(x, \lambda) = \sum_{u \in R} \sum_{p/l} g_{p/l}^{(u)} x_{p/l}^{(u)} + \sum_{ip \in BL} \lambda_{ip} \left( \sum_{u \in R} \sum_{p/l \in ip} x_{p/l}^{(u)} - 1 \right)
\]

\[
= \sum_{u \in R} \left( \sum_{p/l} \left( g_{p/l}^{(u)} + \lambda_p^{(u)} \right) x_{p/l}^{(u)} - \sum_{ip \in BL} \lambda_{ip} \right) \quad (37)
\]

where \( \lambda_{ip} \) is the Lagrangian multiplier (price) for the constraint in Eq.\((36) \), and \( \lambda_p^{(u)} = \sum_{ip \in ip/p} \lambda_{ip} \) is the price associated with prefix \( p/l \). With this relaxation, both the objective function and the other constraints immediately decompose in \(|R|\) independent sub-problems, one per router \( u \):

\[
\min \sum_{p/l} \left( g_{p/l}^{(u)} + \lambda_p^{(u)} \right) x_{p/l}^{(u)} \quad (38)
\]

s.t. \( \sum_{p/l} x_{p/l}^{(u)} \leq F_{\text{max}}^{(u)} \quad (39) \)

\[
T_0^{(u)} - \sum_{p/l} \left( g_{p/l}^{(u)} + h_{p/l}^{(u)} \right) x_{p/l}^{(u)} \leq C^{(u)} \quad (40)
\]

The dual problem is:

\[
\max \sum_{u \in R} \sum_{ip \in BL} h_{ip}(\lambda) - \sum_{ip \in BL} \lambda_{ip} \quad (41)
\]

where \( h_{ip}(\lambda) \) is the optimal solution of \((38)-(40) \) for a given \( \lambda \). Given the prices \( \lambda_{ip} \), every sub-problem \((38)-(40) \) can be solved independently and optimally by router \( u \) using e.g., Eq.\((29) \). Problem \((41) \) can be solved using a projected sub-gradient method \((11) \). In particular, we use the following update rule to compute shadow prices at each iteration:

\[
\lambda_{ip}^{(k+1)} = \lambda_{ip}^{(k)} + \alpha \left( \sum_{u \in R} \sum_{p/l \in ip} x_{p/l}^{(u)} - 1 \right)
\]

where \( \alpha \) is the step size. The interpretation of the update rule is quite intuitive: for every \( ip \) that is filtered with multiple filters the corresponding shadow price, \( \lambda_{ip} \), is augmented proportionally to the number of times it is blocked. Increasing the prices has in turn the effect of forcing the router to try to unblock the corresponding \( ip \). The price is increased until a single filter is used to block that \( ip \).

Note, however, that since \( x \) is an integer variable, \( x \in \{0, 1\} \), the dual problem is not always guaranteed to converge to a primal feasible solution \((13) \).

Distributed vs. Centralized Solution. The above formulation lends itself naturally to a distributed implementation. Each router needs to solve only its own sub-problem \((38)-(40) \) independently from others. A single machine (e.g., the victim’s gateway or a dedicated node) should solve the master problem \((41) \) to iteratively find the prices that coordinate all subproblems. At every iteration of the sub-gradient, the new \( \lambda_{ip} \)’s need to be broadcasted to all routers. Given the \( \lambda_{ip} \)’s, the routes solve independently a sub-problem each and return the computed \( x_{p/l}^{(u)} \) to the node in charge of the master problem. Even in a centralized setting, our scheme lends itself to parallel computation of Eq.\((33)-(56) \).

IV. PRACTICAL EVALUATION

In this section, we evaluate our algorithms using real logs from malicious traffic. We demonstrate that our algorithms bring significant benefit compared to non-optimized filter selection or to generic clustering algorithms, in a wide range of scenarios. The reason behind this benefit is the well-known fact that sources of malicious traffic exhibit spatial and temporal clustering \((3-9) \), which is exploited by our algorithms. Indeed, clustering in a blacklist allows to use a small number of filters to block prefixes with high density of malicious IPs at low collateral damage. Furthermore, it
has also been observed that the good and bad addresses are typically not co-located, which allows for distinguishing between good and bad traffic [6], [7], [14], and in our case for efficient filtering of the most “contaminated” prefixes.

A. Simulation Setup

We used 61-day logs from Dshield.org [10] - a repository of firewall and intrusion detection logs collected. The dataset consists of 758,698,491 attack reports, from 32,950,391 different IP sources belonging to about 600 contributing organizations. Each report includes a timestamp, the contributor ID, and the information for the flow that raised the alarm, including the (malicious) source IP and the (victim) destination IP. Looking at the attack sources in the logs, we verified that malicious sources are clustered in a few prefixes, rather than uniformly distributed over the IP space, consistently with what was observed before e.g., in [3]–[7].

In our simulations, we considered a blacklist to be the set of sources attacking a particular organization (victim) during a day-period. The degree of clustering varied significantly in the blacklists of different victims and across different days. The higher the clustering, the more the benefit we expect from our approach. We also simulated the whitelist, by generating good IP addresses according to the multifractal distribution in [15] on routable prefixes. We performed the simulations on a linux-machine with 2.4 GHz processor with 2 GB RAM.

B. Simulation of BLOCK-ALL and BLOCK-SOME

Simulation Scenarios I & II. In Fig. 4 we considered two example blacklists corresponding to two different victims, each attacked by a large number (up to 100,000) of malicious IPs in a single day. We picked to present the blacklists with the highest and the lowest degree of source clustering observed in the entire dataset these two in particular, referred to as “High Clustering” (I) and “Low Clustering” (II), respectively; these demonstrate the range of benefit from our approach.

BLOCK-ALL. We ran Algorithm 1 in these two scenarios and we show the results in Fig. 4. We made the following observations. First, the optimal algorithm performs significantly better than a generic clustering algorithm that does not exploit the structure of IP prefixes. In particular, it reduces the collateral damage (CD) by up to 85% compared to K-means, when run on the same (high-clustering) blacklist. Second, the degree of clustering in a blacklist matters: the CD is lowest (highest) in the blacklist with highest (lowest) degree of clustering, respectively. Results obtained for different victims and days were similar and lied in between the two extremes. A few thousands of filters were sufficient to significantly reduce collateral damage in all cases.

BLOCK-SOME. In Fig. 5, we focus on Scenario II, i.e., the Low Clustering blacklist and thus the highest CD (dashed line in Fig. 4), which is the least favorable input for our algorithm. Unlike BLOCK-ALL, BLOCK-SOME allows the operator to trade-off lower CD for some unfiltered bad IPs by appropriately tuning the weights. For simplicity, in Fig. 5 we assigned the same weights by appropriately tuning the weights. For simplicity, in Fig. 5 we assigned the same weights to all good and bad sources; however, the framework has the flexibility to assign different weights to different IPs. In Fig. 5(a), the CD is always smaller than the corresponding CD in Fig. 4; they become equal only when we block all bad IPs. In Fig. 5(b), we observe that BLOCK-SOME reduces the CD by 60% compared to BLOCK-ALL while leaving unfiltered only 10% of bad IPs and using only a few hundreds a filters.

In Fig. 5(c), the total cost of the attack (i.e., the weighted sum of bad and good traffic blocked) decreases as $F_{max}$ increases. The interaction between these two competing factors is complex and strongly depends on the input blacklist and whitelist. In the data we analyzed, we observed that CD tends to first increase and then decrease with $F_{max}$, while the number of unfiltered bad IPs tends to decrease. The ratio $w_b/w_g$ captures the effort made by BLOCK-SOME to block all bad IPs and become similar to BLOCK-ALL.

C. Simulation of FLOODING and DIST-FLOODING

Simulation Scenario III. We consider a web server under

2Since we picked a ratio $w_b/w_g > 1$, bad IPs are more important. When $F_{max}$ is high, the algorithm first tries to cover small clusters or single bad IPs. In the case of high $W$, this happens around 10,000 filters. CD remains almost constant in this phase, at the end of which all bad IPs are filtered (as in Fig. 5b). In the final phase, the algorithm releases single good IPs, which are less important and all bad IPs are blocked similarly to BLOCK-ALL.
a DDOS attack. We assume that the server has a typical access bandwidth of \( C = 100 \) Mbps and can handle 10,000 connections per second (a typical capability of a web server that handles light content). We assume that each good (bad) source generates the same amount of good (bad) traffic. We also assume that \( F_{\text{max}} = 12,000 \) filters are available (consistent with the discussion in footnote 1) and we vary \( F = 1, \ldots, F_{\text{max}} \). Before the attack, 5,000 good sources are picked from 15 and utilize 10\% of the capacity. During the attack, the total bad traffic is \( 100C = 1 \) Gbps and is generated by a typical blacklist (141,763 bad source IPs), based on Dshield logs of a randomly chosen victim for a randomly chosen day 3.

**FLOODING - Optimal.** Fig. 6(a) and Fig. 6(b) show the collateral damage of the optimal solution of FLOODING, for Scenario III, as a function of the number of available filters \( F_{\text{max}} \), and as a function of the bottleneck capacity \( C \), respectively. As baseline for comparison, we simulate uniform rate-limiting, which drops the same fraction of all incoming connections and is a common practice in DDOS attacks. Since bad sources outnumber the good sources in a typical DDOS connections and is a common practice in DDOS attacks. Since bad sources outnumber the good sources in a typical DDOS attack, uniform rate-limiting penalizes disproportionally the bad sources.

We also observe that varying the number of filters or the available capacity has a different impact on the collateral damage. While the collateral damage decreases exponentially as the number of filters increases, when we increase the available capacity we observe two trends. First, as capacity increases, the optimal solution allows traffic from good sources that do not belong in prefixes with many malicious sources. This causes a linear decrease with slope equal to the amount of traffic generated by good sources. For even larger \( C \), good IPs located in the same prefix as malicious sources are released. This trend depends on the specific clustering of good and bad IPs considered as well as on the amount of traffic generated by both good and bad sources. In Fig. 6(c) we plot the collateral damage as a function of both the number of available filters and the available capacity. When the value of \( F \) (\( C \)) is too low, increasing \( C \) (\( F \)) does not yield any benefit. Most of the improvement is obtained when both resources increase.

**FLOODING - Heuristic.** The benefit of the optimal solution of FLOODING comes at high computational cost, due to the intrinsic hardness of the FLOODING problem. To address this issue, we design a heuristic for solving FLOODING, which can be tuned to achieve the desired tradeoff between collateral damage and computational time. In particular, instead of solving all subproblems, \( z_p(f, n \Delta C) \) for \( n \in \mathbb{N} \) and \( C = n \Delta C \leq C_{\text{max}}, n = 1, 2, \ldots \). The coarser the capacity increments \( \Delta C \), the fewer subproblems we need to solve, but at the cost of higher collateral damage. In this scenario, increasing \( \Delta C \) significantly reduces the computational time by 3 orders of magnitude, while the percentage of good traffic that is blocked (CD \%) is only increased by a factor 3.

---

3However, because this problem is NP-hard we do not simulate the entire IP space, but the range \([60.0.0.0, 90.0.0.0]\), which is known to account for the largest amount of malicious traffic, e.g., see 6. We also scale all parameters by a factor of 8, \( F_{\text{max}}, C_{\text{max}}, w_i \) to maintain a constant ratio between the number of IPs and \( F_{\text{max}} \), and the total flow generated and \( C_{\text{max}} \).
scenario III, because there is less unused capacity before the attack, and more malicious traffic during the flooding attack.

In Fig. 7 we show the percentage of good traffic that is blocked by the heuristic vs. the time required to obtain a solution, for scenario IV. As we can see in Fig. 7 the optimal solution of FLOODING ($\Delta C = 1$) requires about 1 day of computation and has CD that is only 6% of the total good traffic. Larger values of $\Delta C$ allows to dramatically reduce the computational time by about 3 orders of magnitude while the CD is only increased by a factor 2-3. This asymmetry was also the case in other DShield logs we simulated. This can be very useful in practice: an operator may decide to use an approximation of the optimal filtering policy to immediately cope with incoming bad traffic, and then successively refine the allocation of filters to further reduce the collateral damage if the attack persists.

DISTRIBUTED-FLOODING. We simulated the scenario where an ISP utilizes multiple routers to collaboratively block malicious traffic. We consider the same scenario (III) as for the optimal flooding for a single router, but now we assume that the traffic reaches the victim routed over the example topology illustrated in Fig.8.

We use a sub-gradient descent method to solve the dual problem in Eq. (41). In Fig. 9 we show the convergence of the method for two different step sizes: 0.05 and 0.01. We also compare against the “no coordination” case, when routers do not coordinate but act independently to block malicious traffic; this corresponds to the first iteration of the sub-gradient method. In the next iterations, routers coordinate, through the exchange of shadow prices $\lambda$, and avoid the redundant overlap of prefixes at multiple locations. This reduces the collateral damage significantly, i.e., by $\sim$ 50%.

V. OUR WORK IN PERSPECTIVE

Bigger Picture and Assumptions. Dealing with malicious traffic is a hard problem that requires the cooperation of several components, including detection and mitigation techniques, as well as architectural aspects. In this paper, we do not propose a novel solution. Instead, we optimize the use of filtering - a mechanism that already exists on the Internet today and is a necessary building block of any bigger solution. We focus on the optimal construction of filtering rules, which can be then installed and propagated by filtering protocols [16], [17].

We rely on an intrusion detection system or on historical data, to distinguish good from bad traffic and to provide us with a blacklist. Detection of malicious traffic is an important problem but out of the scope of this paper. The sources of legitimate traffic are also assumed known and used for assessing the collateral damage; e.g., web servers or ISPs typically keep historical data and know their important customers.

We also consider addresses in the blacklist to not be spoofed. This is reasonable today that attackers use botnets, and control a huge number of infected hosts for a short period of time, so that they do not even need to use spoofing. On 2005, less than 20% of addresses were spoofable [18], while in 2008, only 7% of addresses in DShield logs were found likely spoofed [4]. Even if there is some amount of spoofed traffic, our algorithms treat it as the rest of malicious traffic and weight the cost vs. the benefit of blocking a source prefix (which may include both malicious spoofed and legitimate traffic). Looking into the future, there is also a number of proposals promising to enforce source accountability, including ingress filtering [19], self-certifying addresses [20], packet passports [21]. To the extent that spoofing interferes with the ability to define blacklists, our algorithms work best together with an anti-spoofing mechanism, but also do the best that can be done today without it.

Deployment scenario. A practical deployment scenario is that of a single network under the same administrative authority, such as an ISP or a campus network. The operator can use our algorithms to install filters at a single edge router or at several routers, in order to optimize the use of its resources and to defend against an attack in a cost-efficient way. Our distributed algorithm may also be useful, not only for a routers within the same ISP, but also, in the future, when different ISPs start cooperating against common enemies.

ACLs vs. firewall rules. Our algorithms may also be applicable in a different context: to configure firewall rules to protect public-access networks, such as university campus networks or web-hosting networks. Unlike routers where TCAM puts a hard limit on the number of ACLs, there is no hard limit on the number of firewall rules, in software; however, there is still an incentive to minimize their number and thus
any associated performance penalty [22].

There is a body of work on firewall rule management and (mis)configuration [23], which aims at detecting anomalies such as the existence of multiple firewall rules that match the same packet, or the existence of a rule that will never match packets flowing through a specific firewall. In contrast, we focus on resource allocation: given a blacklist and a whitelist as input to the problem, our goal is to optimally select which prefixes to filter so as to optimize an appropriate objective subject to the constraints. Furthermore, the work in [23] considers firewalls for enterprises, which are not supposed to be accessed from outside and thus can be protected without filtering rules.

Measurement studies. Several measurement studies have demonstrated that malicious sources exhibit spatial and temporal clustering [3–7, 9]. In order to deal with dynamic malicious IP addresses [8], IP prefixes rather than individual IP addresses are typically considered. The clustering, in combination with the fact that the distribution of addresses as well as other statistical characteristics differ for good and bad traffic, have been exploited in the past for detection and mitigation of malicious traffic, such as e.g., spam [6, 7] or DDoS [14]. In this work, we exploit these characteristics for efficient prefix-based filtering of malicious traffic.

Prefix Selection. The work in [14] studied source prefix filtering for classification and blocking of DDoS traffic, which is closely related to our FLOODING problem. The selection of prefixes in [14] was done heuristically, thus leading to large collateral damage was incurred. In contrast, we tackle analytically the optimal source prefix selection so as to minimize collateral damage. Furthermore, we provide a more general framework for formulating and optimally solving a family of related problems, including but not limited to FLOODING.

The work in [24] is related to our TIME-VARYING problem: it designed and analyzed an online learning algorithm for tracking malicious IP prefixes based on a stream of labeled data. The goal was detection, i.e., classifying a prefix as malicious, depending on the ratio of malicious and legitimate traffic it generates, and subject to a constraint on the number of prefixes. In contrast: (i) we identify precisely (not approximately) the IP prefixes with the highest concentration of malicious traffic; (ii) we follow a different formulation (dynamic programming inspired by knapsack problems); (iii) we use the results of detection as input to our filtering problem.

An earlier body of literature focused on identifying IP prefixes with significant amount of network traffic, typically referred to as hierarchical heavy hitters: [25–27]. However, it did not consider the interaction between legitimate and malicious traffic within the same prefix, which is the core tradeoff studied in this paper.

Relation to Knapsack Problems. Filter selection belongs to the family of multidimensional knapsack problems (dKP) [11]. The general dKP problem is well-known to be NP-hard. The most relevant variation is the knapsack with cardinality constraint (1.5KP) [28, 29], which has \(d = 2\) constraints, one of them being a limit on the number of items: \(\sum_{i \in N} w_i x_i \leq C\) and \(\sum_{j \in N} x_j \leq k\). The 1.5KP problem is also NP-hard.

These classic problems do not consider correlation between items. However, in filtering, the selection of an item (prefix) voids the possibility to select other items (all overlapping prefixes). dKP problems with correlation between items have been studied in [30, 31], where the items were partitioned into classes and up to one item per class was picked. In our case, a class is the set of all prefixes covering a certain address. Each item (prefix) can belong simultaneously to any number of classes, from one class (/32 address) to all classes (/0 prefix).

To the best of our knowledge, we are the first to tackle a case where the classes are not a partition of the set of items.

A continuous relaxation does not help either. Allowing \(x_{p/l}\) to be fractional corresponds to rate-limiting of prefix \(p/l\). This has no advantage neither from a practical (rate limiters are more expensive than ACLs, because in addition to looking up packets in TCAM, they also require rate and computation on the fast path) nor from a theoretical point of view (the continuous 1.5KP is still NP-hard [32]).

In summary, the special structure of the prefix filtering problem, i.e., the hierarchy and overlap of candidate prefixes, leads to novel variations of dKP that could not be solved by directly applying existing methods in the KP literature.

Our prior work. This journal paper builds on our conference paper in [33]. Compared to [33], new contributions in this paper include: the formulation and optimal solution of the time-varying version of the filtering problem; an extended evaluation section, which simulates all filtering problems over Dshield.org logs, including FLOODING and DIST-FLOODING which were not evaluated in [33]; and additional proofs, complexity analysis and comments that were not in [33].

Earlier on, in a related workshop paper [34], we also studied optimal range-based filtering, where malicious source addresses were aggregated into continuous ranges (of numbers in the IP address space \([0, 2^{32} − 1]\), instead of prefixes. This was an easier problem that allowed for greedy solutions. Unfortunately, ranges are not implementable in ACLs; furthermore, it is well-known that ranges cannot be efficiently approximated by a combination of prefixes [12]. Therefore, despite the intuition we gained in [34], we had to solve the problem of prefix-based filtering from scratch in this paper.

VI. Conclusion

In this paper, we introduce a formal framework for optimal source prefix-based filtering. The framework is rooted at the theory of the knapsack problem and provides a novel extension to it. Within it, we formulate five practical problems, presented in increasing order of complexity. For each problem, we designed optimal algorithms that are also low-complexity (linear or pseudo-polynomial in the input size). We simulate our algorithms over Dshield.org logs and demonstrate that they bring significant benefit compared to non-optimized filter selection or to generic clustering algorithms. A key insight behind that benefit is that our algorithms exploit the spatial and temporal clustering exhibited by sources of malicious traffic.
Dshield.org logs are available to us. We would also like to thank M. Faloutsos for insightful discussions. This work has been supported by the NSF CyberTrust grant 0831530.

REFERENCES

[1] W. P. ——, “Understanding acl on catalyst 6500 series switches,” http://www.cisco.com/en/US/products/hw/solutions/ps708/products_ white_paper09186a00800c9470.shtml.
[2] ——, “Protecting your core: Infrastructure protection access control lists,” http://www.cisco.com/en/US/tech/tk648/tk361/technologies_white_paper09186a0801a1a55.shtml.
[3] M. Collins, T. Shimneall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane, “Using uncleanness to predict future botnet addresses,” in Proc. of the ACM Internet Measurement Conference, San Diego, CA, USA, Oct. 2007.
[4] Z. Chen, C. Ji, and P. Barford, “Spatial-temporal characteristics of internet malicious sources,” in Proc. of IEEE INFOCOM Mini-Conference, Phoenix, AZ, USA, May 2008.
[5] Z. Mao, V. Sekar, O. Spatschek, J. Van Der Merwe, and R. Vasudevan, “Analyzing large ddos attacks using multiple address sources,” in Proc. of ACG SIGCOMM Workshop on Large-Scale Attack Defense, Pisa, Italy, Sept. 2006.
[6] A. Ramachandran and N. Feamster, “Understanding the network-level behavior of spammers,” in Proc. of ACM SIGCOMM, Pisa, Italy, Sept. 2006.
[7] Z. Chen, C. Ji, P. Barford, and A. Ramachandran, “Understanding the network-level behavior of spammers,” in Proc. of the USENIX Security Symposium, San Jose, CA, USA, July 2008.
[8] Dshield.org, “Dshield: Cooperative network security community - internet security,” http://www.dshield.org.
[9] Fabio Soldo is currently a Ph.D. candidate in the Networked Systems Program at UC Irvine. He received the B.S. degree in Mathematics from Politecnico di Torino, Italy, in 2004, and M.S. degree in Mathematical Engineering from Politecnico di Torino and Politecnico di Milano, Italy, in 2006. He had internships with Telefonica Research, Docomo Labs and Google. His research interests are in the areas of design and analysis of network algorithms and network protocols, and defense mechanisms against malicious traffic.
[10] Katerina Argyraki is a researcher with the Operating Systems group in the School of Computer and Communication Sciences, EPFL, Switzerland. She works on network architectures and protocols with a focus on denial-of-service defenses and accountability. She received her undergraduate degree in Electrical and Computer Engineering from the Aristotle University, Thessaloniki, Greece, in 1999, and her Ph.D. in Electrical Engineering from Stanford University, in 2007.
[11] Athina Markopoulou (SM ’98, M’02) is an assistant professor in the EECS Dept. at the University of California, Irvine. She received the Diploma degree in Electrical and Computer Engineering from the National Technical University of Athens, Greece, in 1996, and the M.S. and Ph.D. degrees, both in Electrical Engineering, from Stanford University in 1998 and 2003, respectively. She has been a postdoctoral fellow at Sprint Labs (2003) and at Stanford University (2004-2005), and a member of the technical staff at Aratra Inc. (2005). Her research interests include network coding, network measurements and security, media streaming and online social networks. She received the NSF CAREER award in 2008.