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.**

Index Terms— **Network Security, Internet, Filtering, Clustering Algorithms.**

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 the platform for launching these attacks.

Protecting a victim (host or network) from malicious traffic is a hard problem that requires the coordination of several complementary components, including non-technical (*e.g.,* business and legal) and technical solutions (at the application and/or network level). Filtering support from the network is 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 [1], and they are currently used for enforcing a variety of policies, including infrastructure protection [2]. 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.[1] 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 [3]–[9], 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

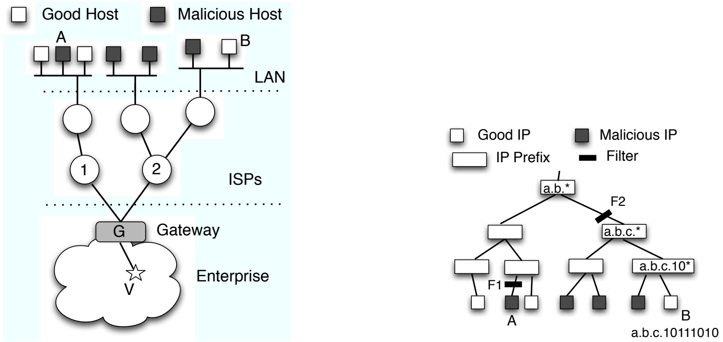
This work was supported by the NSF CyberTrust Award 0831530 and by an Ambizone grant from the Swiss National Science Foundation.

Fabio Soldo and Athina Markopoulou are with the Department of Electrical Engineering and Computer Science at the University of California, Irvine (email: {fsoldo, athina}@uci.edu).

Katerina Argyraki is with the School of Computer Science and Communication Sciences at EPFL, Switzerland (email: katerina.argyraki@epfl.ch).

Manuscript received on June 3rd, 2010; revised on July 1st, 2011.

[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.



(a) Actual network.

(b) Hierarchy of source IP addresses and prefixes

Fig. 1. Example of a distributed attack. Let's assume that the gateway router G has only two filters, $F1$ and $F2$, available to block malicious traffic and protect the victim V. It uses $F1$ to block a single malicious source address (A) and $F2$ to block the entire source prefix $a.b.c.*$, which contains 3 malicious sources but also one legitimate source (B). Therefore, the selection of filter $F2$ trades-off collateral damage (blocking B) for reduction in the number of filters (from 3 to 1). We note that both filters, $F1$ and $F2$, are ACLs installed at the same router G.

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 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 use 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` [10] 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 ($\mathcal{BL}$) is a set of unique source IP addresses that send bad (undesired) traffic to the victim. Similarly, a whitelist ($\mathcal{WL}$) 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 a to whitelist (in which case we call it a "good" address), but not to both. We use $|\mathcal{BL}|$ and $|\mathcal{WL}|$ to indicate the number of addresses in $\mathcal{BL}$ and $\mathcal{WL}$, respectively. For brevity, we also use $N = |\mathcal{BL}|$ 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 its importance. 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 $\mathcal{BL}$ 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 $\mathcal{WL}$ to include all addresses that are not in $\mathcal{BL}$.

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_{max} is the maximum number of available filters, and it is given as input to our problem. Filter optimization is meaningful only when F_{max} is much smaller than the size of the blacklist $N = |\mathcal{BL}|$; otherwise the optimal would be to block every single bad address. $F_{max} << N$ is indeed the case in practice due to the size and cost of the TCAM, as mentioned in Section I.

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

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

B. Rationale and Overview of Filtering Problems

Given a set of bad and a set of good source addresses ($\mathcal{BL}$ and $\mathcal{WL}$), a measure of their importance (the address weights w), and a resource budget (F_{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.

$$\min \sum_{p/l} \sum_{ip \in p/l} w_{ip} x_{p/l} \tag{1}$$

$$\text{s.t.} \sum_{p/l} x_{p/l} \leq F_{max} \tag{2}$$

$$\sum_{p/l: ip \in p/l} x_{p/l} \leq 1 \quad \forall ip \in \mathcal{BL} \tag{3}$$

$$x_{p/l} \in \{0,1\} \quad \forall l = 0,...,32, p = 0,...,2^l \tag{4}$$

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: $l = 0,...,32$, $p = 0,...,2^l - 1$. 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.

ip	Generic IP address
w_{ip}	Weight assigned to address ip
$\mathcal{BL}$	Blacklist: a list of bad addresses
$N = \vert\mathcal{BL}\vert$	Number of unique addresses in $\mathcal{BL}$
$\mathcal{WL}$	Whitelist: a set of "good" addresses
p/l (or "p" for short)	Prefix p of length l bits (IP/mask notation)
$ip \in p/l$	IP address that belongs to prefix p/l
$x_{p/l} \in \{1,0\}$	Indicates if a filter blocks prefix p/l or not
$g_{p/l} = \sum_{ip \in p/l \cap \mathcal{WL}} w_{ip}$	Collateral damage from filtering prefix p/l
$b_{p/l} = \vert\sum_{ip \in p/l \cap \mathcal{BL}} w_{ip}\vert$	Bad traffic blocked by filtering prefix p/l
F_{max} $(<< N)$	Maximum number of available filters
$z_p(F)$	Value of the optimal solution of subproblem that considers only addresses in prefix p and up to F filters
$X_p(F)$	Set of filters used in optimal solution $z_p(F)$

TABLE I
SUMMARY OF NOTATION AND TERMINOLOGY

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 V, 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 $\mathcal{BL}$ of size N, a whitelist $\mathcal{WL}$, 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_{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 good 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 is it 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.,* $|\mathcal{BL}| + |\mathcal{WL}|$.

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 across 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.

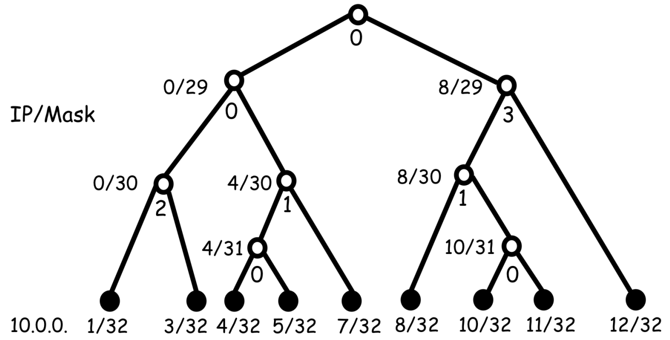


Fig. 2. **Example of LCP-tree($\mathcal{BL}$).** Consider a blacklist consisting of the following 9 bad addresses: $\mathcal{BL} = \{10.0.0.1, 10.0.0.3, 10.0.0.4, 10.0.0.5, 10.0.0.7, 10.0.0.8, 10.0.0.10, 10.0.0.11, 10.0.0.12\}$. All remaining addresses are considered good. Each leaf represents one address in the $\mathcal{BL}$. Each intermediate node represents the longest common prefix p covering all bad addresses in that subtree. At each intermediate node p, we also show the collateral damage (*i.e.,* number of good addresses blocked) when we filter prefix p instead of filtering each of its children. *E.g.,* if we use two filters to block bad addresses 10.0.0.5/32 and 10.0.0.7/32 the collateral damage is 0; if, instead, we use one filter to block prefix 10.0.0.4/30, we also block good address 10.0.0.6/32, *i.e.,* we cause collateral damage 1.

.

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 $\mathcal{BL}$, a whitelist $\mathcal{WL}$, and the number of available filters F_{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. (3) 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.(3).

$$\min \sum_{p/l} g_{p/l} x_{p/l} \tag{5}$$

$$\text{s.t.} \quad \sum_{p/l} x_{p/l} \leq F_{max} \tag{6}$$

$$\sum_{p/l: ip \in p/l} x_{p/l} = 1 \qquad \forall ip \in \mathcal{BL} \tag{7}$$

Characterizing an Optimal Solution: Our algorithm starts from LCP-tree($\mathcal{BL}$) 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($\mathcal{B}L$) with the same root as LCP-tree($\mathcal{BL}$), up to F_{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($\mathcal{BL}$) 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$\big(\{0, 1, ..., 2^{32} - 1\}\big)$) 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_{max} filters, *i.e.,* its tree has up to F_{max} leaves. Indeed, if this was not the case, Eq. (6) 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($\mathcal{BL}$). 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 $\mathcal{BL} \cap \tilde{p}$, thus already on the LCP-tree($\mathcal{BL}$).

Clearly, each of these operations transforms feasible solution S, which is assumed not to be on LCP-tree($\mathcal{BL}$), into another feasible solution S' with smaller or equal collateral damage but on the LCP-Tree($\mathcal{BL}$). We can repeat this process for all prefixes that are in S's tree but not in LCP-tree($\mathcal{BL}$), until we create a feasible solution S' that includes only prefixes from LCP-tree($\mathcal{BL}$) 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: Algorithm 1, which solves BLOCK-ALL, consists of two steps. First, we build the LCP tree from the input blacklist $\mathcal{BL}$. Second, in a bottom-up fashion, we compute $z_p(F)\ \forall p, F$, *i.e.,* the minimum collateral damage needed to block all bad addresses in the subtree of prefix p using at most F filters. Following a dynamic programming (DP) approach, we can find the optimal allocation of filters in the subtree rooted at prefix p, by finding a value n and assigning $F - n$ filters to the left subtree and n to the right subtree, so as to minimize collateral damage. The fact that BLOCK-ALL needs to filter all bad addresses (leaves in the LCP tree) implies that at least one filter must be assigned to the left and right subtree, *i.e.,* $n = 1, 2, ..., F-1$. In other words, for every pair of sibling nodes, s_l (left) and s_r (right), with common parent node p, the following recursive equation holds:

$$z_p(F) = \min_{n=1,\ldots,F-1} \Big\{ z_{s_l}(F-n) + z_{s_r}(n) \Big\}, \ F > 1 \quad (8)$$

with boundary conditions for leaf and intermediate nodes:

$$z_{leaf}(F) = 0 \ \ \forall F \geq 1 \quad (9)$$

$$z_p(1) = g_p \ \ \forall p \quad (10)$$

Once we compute $z_p(F)$ for all prefixes in the LCP tree, we simply read the value of the optimal solution, $z_{root}(F_{max})$. 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: $X_p(F) = X_{s_l}(F-n) \cup X_{s_r}(n)$.

Algorithm 1 Algorithm forsolving BLOCK-ALL

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

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 $x_{p/l} = 1$ for Eq. (5)–(7).

Proof: Recall that, $z_{root}(F_{max})$ denotes the value of the optimal solution of BLOCK-ALL with F_{max} filters (*i.e.,* the minimum collateral damage), while $X_{root}(F_{max})$ denotes the set of filters selected in the optimal solution. Let s_l and s_r denote the two children nodes (prefixes) of $root$ in the LCP-tree($\mathcal{BL}$). Finding the optimal allocation of $F_{max} > 1$ filters to block all addresses contained in $root$ (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 $x + y = F_{max}$. 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 $F_{max} = 1$, *i.e.,* there is only one filter available, the only feasible solution is to select $root$ 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 $|\mathcal{BL}|$ leaves; therefore, it has $O(|\mathcal{BL}|)$ intermediate nodes (prefixes). Computing Eq. (8) for every node p and for every value $F \in [1, F_{max} - 1]$ involves solving $O(|\mathcal{BL}|F_{max})$ sub-problems, one for every pair (p, F) with complexity $O(F_{max})$. $z_p(F)$ 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 $O(|\mathcal{BL}|F_{max}^2)$. 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 $\mathcal{BL} \cap p$, which may require $F \leq F_{max}$ for long prefixes in the LCP-tree.

Finally, we observe that the asymptotic complexity is $O(|\mathcal{BL}|)$, since $F_{max} << N = |\mathcal{BL}|$ and F_{max} does not depend on $|\mathcal{BL}|$ but only on the TCAM size. Thus, the time complexity increases linearly with the number of bad addresses $|\mathcal{BL}|$. This is within a constant factor of the lowest achievable complexity, since we need to read all $|\mathcal{BL}|$ bad addresses at least once. Although the above is a worst-case analysis, we confirmed in simulation that the computation time in practice is very close to that.

C. BLOCK-SOME

Problem Statement: Given a blacklist $\mathcal{BL}$, a whitelist $\mathcal{WL}$, 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 $g_{p/l}$ and the filtering benefit $b_{p/l}$, which is expressed by Eq. (11). Second, Eq. (13) states that every bad address must be filtered by *at most* one prefix, which means that it may or may not be filtered.

$$\min \sum_{p/l} \Big(g_{p/l} - b_{p/l}\Big) x_{p/l} \tag{11}$$

$$\text{s.t.} \sum_{p/l} x_{p/l} \leq F_{max} \tag{12}$$

$$\sum_{p/l: ip \in p/l} x_{p/l} \leq 1 \qquad \forall ip \in \mathcal{BL} \tag{13}$$

Characterizing an Optimal Solution: As with BLOCK-ALL, our algorithm starts from LCP-tree($\mathcal{BL}$) 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($\mathcal{B}L$) with: the same root as LCP-tree($\mathcal{B}L$) 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, whereas in line (11) of Algorithm 1 we had $n = 1, ..., F-1$, now we have $n = 0, 1, ..., F$. We can recursively compute the optimal solution as before:

$$z_p(F) = \min_{n=0,\ldots,F} \Big\{ z_{s_l}(F-n) + z_{s_r}(n) \Big\} \tag{14}$$

with boundary conditions

$$z_p(0) = 0 \quad \forall\ p \tag{15}$$

$$z_p(1) = \min \Big\{ g_p - b_p, \min_{n=0,1} \Big\{ z_{s_l}(1-n) + z_{s_r}(n) \Big\} \Big\} \tag{16}$$

$$z_{leaf}(F) = -b_{leaf} \quad \forall F \geq 1 \tag{17}$$

where p is an intermediate node (prefix) and $leaf$ 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 $|\mathcal{BL}|$.

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 $\mathcal{BL}$ 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 the case when 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, $\{\mathcal{BL}_{\tau_0}, \mathcal{BL}_{\tau_1}, \ldots\}$ and of whitelists, $\{\mathcal{WL}_{\tau_0}, \mathcal{WL}_{\tau_1}, \ldots\}$ at times $\tau_0, \tau_1, \ldots$, respectively.

Problem Statement: Given: (i) a blacklist and whitelist, $\mathcal{BL}_{\tau_{i-1}}$ and $\mathcal{WL}_{\tau_{i-1}}$, (ii) the number of available filters F_{max}, (iii) the corresponding solution to BLOCK-ALL(SOME), denoted $\mathcal{S}_{\tau_{i-1}}$, and (iv) another blacklist and whitelist, $\mathcal{BL}_{\tau_i}$ and $\mathcal{WL}_{\tau_i}$; obtain the solution to BLOCK-ALL(SOME) for the second blacklist/whitelist, denoted $\mathcal{S}_{\tau_i}$.

Algorithm: Consider, for the moment, that the whitelist remains the same and assume the focus is 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 $\mathcal{BL}_{\tau_{i-1}}$, but appears in $\mathcal{BL}_{\tau_i}$. There are two cases, depending on whether the new bad address belongs to a prefix that is already filtered in $\mathcal{S}_{\tau_{i-1}}$. If it is, no further action is needed, and $\mathcal{S}_{\tau_i} = \mathcal{S}_{\tau_{i-1}}$. Otherwise, we modify the LCP tree that represents $\mathcal{S}_{\tau_{i-1}}$ to also include the new bad address, as illustrated in Fig. 3. The key point is that we only need to add one new intermediate node to the LCP tree (the gray node in

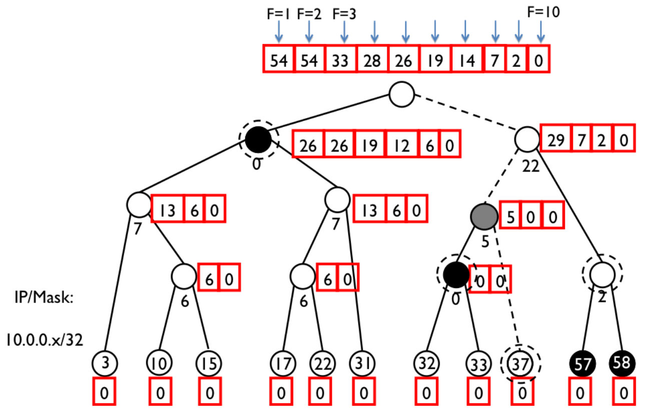


Fig. 3. **Example of (i) BLOCK-ALL and (ii) TIME-VARYING BLOCK-ALL.** Consider a blacklist of 10 bad IP addresses $\mathcal{BL} = \{10.0.0.3, 10.0.0.10, 10.0.0.15, 10.0.0.17, 10.0.0.22, 10.0.0.31, 10.0.0.32, 10.0.0.33, 10.0.0.57, 10.0.0.58\}$ The table next to each node p shows the minimum cost $z_p(F)$ computed by the DP algorithm for BLOCK-ALL for $F = 1,$ *...number of leaves in subtree*. The optimal solution to BLOCK-ALL consists of the 4 prefixes highlighted in black. When a new address, *e.g.,* 10.0.0.37, is added to the blacklist, a leaf node is added to the tree and TIME-VARYING needs to update all and only the ancestor nodes in LCP-tree($\mathcal{BL}$), indicated by the dashed lines, according to Eq. (8). Moreover, a new node is created to denote the longest common prefix between 10.0.0.37 and 10.0.0.32 (or 10.0.0.33). Note that all other nodes corresponding to the longest common prefixes between 10.0.0.37 and other addresses in $\mathcal{BL}$ are already in the LCP tree. The new optimal solution consists of the 4 prefixes indicated by the dashed circles.

Fig. 3), 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 $X_p(F)$ $\forall F$, according to Eq. (8)) for all and only the ancestors 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 $\mathcal{BL}_{\tau_{i-1}}$ but not in $\mathcal{BL}_{\tau_i}$. In this case, we modify the LCP tree that represents $\mathcal{S}_{\tau_{i-1}}$ to we 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 ancestors.

(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 ancestors.

(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 $\log(|\mathcal{BL}|)$ ancestors, so, inserting a new bad address (or removing one) requires $O(\log(|\mathcal{BL}|)F^2_{max})$ operations. Hence, deriving $\mathcal{S}_{\tau_i}$ from $\mathcal{S}_{\tau_{i-1}}$ 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 $\frac{|\mathcal{BL}|}{\log|\mathcal{BL}|}$.

E. FLOODING

Problem Statement: Given: (i) a blacklist $\mathcal{BL}$ and a whitelist $\mathcal{WL}$, 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_{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. (18), 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, $T_0 = \sum_{ip \in \mathcal{BL} \cup \mathcal{WL}} w_{ip}$, minus the traffic that gets blocked, $\sum_{p/l} \Big(g_{p/l} + b_{p/l}\Big) x_{p/l}$ should fit within the link capacity C, so as to avoid congestion and packet loss.

$$\min \sum_{p/l} g_{p/l} x_{p/l} \tag{18}$$

$$\text{s.t.} \sum_{p/l} x_{p/l} \leq F_{max} \tag{19}$$

$$T_0 - \sum_{p/l} \Big(g_{p/l} + b_{p/l}\Big) x_{p/l} \leq C \tag{20}$$

$$\sum_{p/l: ip \in p/l} x_{p/l} \leq 1 \quad \forall ip \in \mathcal{BL} \tag{21}$$

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($\mathcal{BL} \cup \mathcal{WL}$). 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($\mathcal{BL} \cup \mathcal{WL}$), with the same root, up to F_{max} leaves, and the 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($\mathcal{BL} \cup \mathcal{WL}$) 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$\big(\{0, 1, ..., 2^{32} - 1\}\big)$ 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_{max} leaves, otherwise Eq. (19) 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 $\tilde{p}$ that is *not* in LCP-tree($\mathcal{BL} \cup \mathcal{WL}$). We can construct a better feasible solution S', which can be represented as a pruned subtree of LCP-tree($\mathcal{BL} \cup \mathcal{WL}$): S' has the same root, up to F_{max} leaves and total cost of the leaves $\geq T_0 - C$. There are three possibilities:

1) $\tilde{p}$ includes neither bad nor good addresses. In this case, we can simply remove $\tilde{p}$ from S, *i.e.,* unblock $\tilde{p}$.
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($\mathcal{BL} \cup \mathcal{WL}$), until we create a feasible solution S' that includes only prefixes from LCP-tree($\mathcal{BL} \cup \mathcal{WL}$) 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:

$$\max \sum_{i \in N} p_i x_i \tag{22}$$

$$\sum_{i \in N} x_i = k \tag{23}$$

$$\text{s.t.} \sum_{i \in N} w_i x_i \leq C_1 \tag{24}$$

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 $F < |\mathcal{BL}|$, 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($\mathcal{BL} \cup \mathcal{WL}$). The solution constructed this way uses $F + 1$ filters, blocks all addresses blocked in S, and has a cost that is less or equal to S's.

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

$$\max \sum_{p/l} g_{p/l} \bar{x}_{p/l} \tag{25}$$

$$\text{s.t.} \sum_{p/l} \bar{x}_{p/l} = \bar{F}_{max}, \tag{26}$$

$$\sum_{p/l} \Big(g_{p/l} + b_{p/l}\Big) \bar{x}_{p/l} \leq \bar{C} \tag{27}$$

$$\sum_{p/l: ip \in p/l} -\bar{x}_{p/l} \leq -1 \; \forall ip \in \mathcal{BL} \tag{28}$$

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 $ip \in \mathcal{BL} \cup \mathcal{WL}$: $g_{ip} = p_i$, $(g_{ip} + b_{ip}) = w_i$. 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:

$$z_p(F, c) = \min_{\substack{n=0,\ldots,F \\ m=0,\ldots,c}} \{z_{s_l}(F - n, c - m) + z_{s_r}(n, m)\} \tag{29}$$

where $z_p(F, c)$ is the minimum collateral damage 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($\mathcal{BL} \cup \mathcal{WL}$). Moreover, the computation of a single entry, given the entries of descendant nodes, require $O(CF_{max})$ operations, Eq.(29). 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 $c = 0, ..., \tilde{C} = \min\{C, \sum_{ip \in p/l} w_{ip}\} \leq C$ and $f = 0, ..., F = \max\{F_{max}, |leaves(p)|\}$. Therefore, the optimal solution to FLOODING, $z_{root}(F_{max}, C)$, can be computed in $O((|\mathcal{BL}| + |\mathcal{WL}|)C^2)$ time. This is increasing linearly with the number of addresses in $\mathcal{BL} \cup \mathcal{WL}$ 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 ΔC 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):

$$\max_{\lambda\geq 0}\Big\{ \min\sum_{p/l} \Big[(1-\lambda)g_{p/l}-\lambda b_{p/l}\Big]x_{p/l}+ \quad (30)$$
$$+\sum_{p/l}\lambda T_0-\lambda C\Big\}$$

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

$$\sum_{p/l:ip\in p/l} x_{p/l}\leq 1 \quad \forall\ ip\in\mathcal{BL} \quad (32)$$

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 λ, 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 $\mathcal{BL}$, as in Fig.1(a). To reach the victim, attack traffic passed through one or more ISP routers. Let $\mathcal{R}$ be the set of unique such routers. Let each router $u\in\mathcal{R}$ have capacity $C^{(u)}$ on the downstream link (towards V) and a limited number of filters $F^{(u)}_{max}$. 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 $x^{(u)}_{p/l}\in\{0,1\}$ indicate whether or not filter p/l is used at router u. Then the distributed filtering problem can be stated as:

$$\min\sum_{u\in\mathcal{R}}\sum_{p/l} g^{(u)}_{p/l}x^{(u)}_{p/l} \quad (33)$$

$$\text{s.t.} \sum_{p/l} x^{(u)}_{p/l}\leq F^{(u)}_{max} \qquad \forall u\in\mathcal{R} \quad (34)$$

$$T^{(u)}_0-\sum_{p/l}\Big(g^{(u)}_{p/l}+b^{(u)}_{p/l}\Big)x^{(u)}_{p/l}\leq C^{(u)} \qquad \forall u\in\mathcal{R} \quad (35)$$

$$\sum_{u\in\mathcal{R}}\sum_{p/l\ni ip} x^{(u)}_{p/l}\leq 1 \qquad \forall ip\in\mathcal{BL} \quad (36)$$

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

Proposition 3.6: There exists an optimal solution of DIST-FLOODING that can be represented as a set of $|\mathcal{R}|$ different pruned subtrees of the LCP-tree($\mathcal{BL}\cup\mathcal{WL}$), 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 $|\mathcal{R}|$ subtrees, one for every router, each having at most $F^{(u)}_{max}$ leaves and their associated blocked traffic $\geq T^{(u)}_0 - C^{(u)}$, where $T^{(u)}_0$ 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:

$$L(x,\lambda)=\sum_{u\in\mathcal{R}}\sum_{p/l} g^{(u)}_{p/l}x^{(u)}_{p/l}+\sum_{ip\in\mathcal{BL}}\lambda_{ip}\Big(\sum_{u\in\mathcal{R}}\sum_{p/l\ni ip}x^{(u)}_{p/l}-1\Big)$$
$$=\sum_{u\in\mathcal{R}}\Big(\sum_{p/l}\Big(g^{(u)}_{p/l}+\lambda_{p/l}\Big)x^{(u)}_{p/l}\Big)-\sum_{ip\in\mathcal{BL}}\lambda_{ip} \quad (37)$$

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

$$\min\sum_{p/l}\Big(g^{(u)}_{p/l}+\lambda_{p/l}\Big)x^{(u)}_{p/l} \quad (38)$$

$$\text{s.t.}\sum_{p/l}x^{(u)}_{p/l}\leq F^{(u)}_{max} \quad (39)$$

$$T^{(u)}_0-\sum_{p/l}\Big(g^{(u)}_{p/l}+b^{(u)}_{p/l}\Big)x^{(u)}_{p/l}\leq C^{(u)} \quad (40)$$

The dual problem is:

$$\max_{\lambda_{ip}\geq 0}\sum_{u\in\mathcal{R}}h_u(\lambda)-\sum_{ip\in\mathcal{BL}}\lambda_{ip} \quad (41)$$

where $h_u(\lambda)$ is the optimal solution of (38)-(40) for a given λ. Given the prices λ_{ip}, every sub-problem (38)-(40) can be solved independently and optimally by router u using 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:

$$\lambda^{(k+1)}_{ip}=\lambda^{(k)}_{ip}+\alpha(\sum_{u}\sum_{p/l\ni\ ip}x^{(u)}_{p/l}-1)$$

where α 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, λ_{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. When the number of source attackers is too large to be blocked at a single router, we need to combine resources such as the overall network

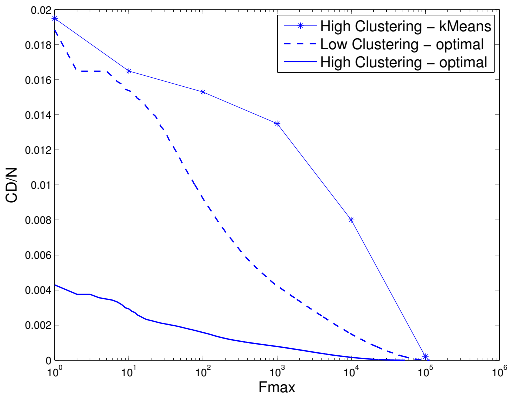


Fig. 4. **Evaluation of BLOCK-ALL in Scenarios I (High Clustering blacklist) and II (Low Clustering blacklist.** We plot the collateral damage (CD) (normalized over the number of malicious sources N) vs. number of filters F_{max}. We compare Algorithm 1 to k-means clustering (we simulated 50 runs of Lloyd's algorithm [14]).

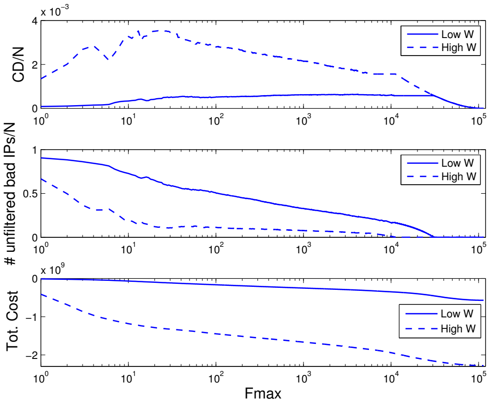


Fig. 5. **Evaluation of BLOCK-SOME for Scenario II (Low Clustering blacklist)**. Three metrics are considered: (a) collateral damage (CD) (b) number of unfiltered bad IPs (UBIP) (c) and total cost $CD + W \cdot UBIP$. The operator expresses preference for UBIP vs. CD by tuning the weights $w_b,\ w_g$. We considered a high (2^{14}) and a low (2^{10}) value for $W = \frac{w_b}{w_g}$.

capacity and the number of filters, available at multiple routers to reduce the overall collateral damage. The optimal allocation of filters, in that case, can be found by solving Eq.(33)-(36) provides the optimal allocation of filters in that case.

The solution can be found either in a centralized or in a distributed way. In a centralized approach, a single node solves Eq.(33)-(36) and distributes the optimal filter allocation to the all routers involved. This reduces the communication overhead between routers, however, the computation burden is put all on the node that solves both the master problem and $|\mathcal{R}|$ different subproblems. This can become infeasible in practice due to the large number of attackers.

An alternative approach is to have each router solve its own sub-problem Eq.(38)-(40) and a single node (*e.g.,* the victim's gateway or a dedicated node) to solve the master problem Eq.(41).The communication overhead of this scheme is limited. At every iteration of the sub-gradient, the shadow prices λ_{ip}'s are broadcast to all routers. For 100,000 of bad IPs, if we encode the value of variables λ_{ip} using 2 Bytes, each router need to send about 200KB of data at each iteration. Given the λ_{ip}'s, the routers solve independently a sub-problem each and return the computed $x_{p/l}^{(u)}$ to the node in charge of the master problem. In general, let N denote the total number of active IP sources, the communication overhead is $O(N + F_{max})$ per router per iteration. Our approach only requires communication between the node solving the master problem and the nodes solving the subproblems. It does not entail communication between subproblems, which would incur a significant communication overhead.

IV. Simulation Results

In this section, we evaluate our algorithms using real logs of malicious traffic from `Dshield.org`. 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 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 good and bad addresses are typically not co-located, which allows for distinguishing between good and bad traffic [6], [7], [15], and in our case for efficient filtering of the prefixes with most malicious sources.

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 single 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 [16] 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

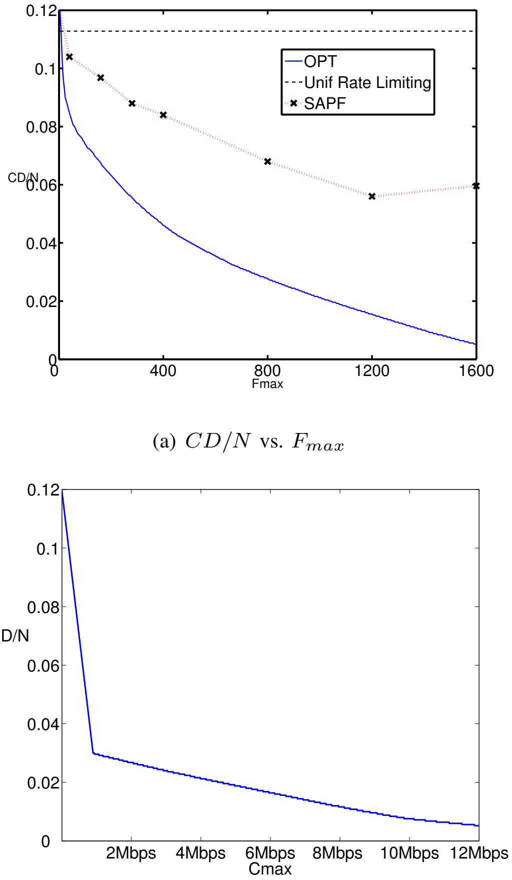
Simulation Scenarios I & II. In Fig. 4, we consider two example blacklists corresponding to two different victims, each attacked by a large number of malicious IPs in a single day. In order to demonstrate the range of benefit of our approach, we chose the blacklists with the highest and the lowest degree of source clustering observed in the entire dataset, referred to as “High Clustering” (Scenario I) and “Low Clustering” (Scenario II), respectively. The degree of clustering of a blacklist is captured by the entropy associated with the distribution of the IP addresses [17]. Intuitively, a blacklist with low entropy (*i.e.,* high clustering) contains IP addresses in a few prefixes and thus is easier to block.

We compare Algorithm 1 to a k-means, which is general yet prefix-agnostic. k-means is a well-known clustering problem [18]: the goal is to partition all observations (in our context, IP addresses) in k clusters such that each address belongs to the cluster with the nearest mean. We use the most common algorithm to solve the problem, known as Lloyd’s algorithm [14], which uses an iterative refinement technique.

BLOCK-ALL. We ran Algorithm 1 in these Scenarios I and II 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, which is the least favorable input for our algorithm and has the highest CD (shown in dashed line in Fig.4). Compared to BLOCK-ALL, which blocks all bad IPs, BLOCK-SOME allows the operator to trade-off lower CD for some unfiltered bad IPs by appropriately tuning the weights assigned to good (w_g) and bad (w_b) addresses. For simplicity, in Fig. 5, we assign the same weight w_g to all good addresses, and the same weight w_b to all bad addresses. Fig. 5 shows results for two different values of $W = \frac{w_b}{w_g}$. (However, we note that our framework has the flexibility to assign different weights to each individual IP address.)

In Fig. 5(a), 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[2] made by BLOCK-SOME to block all bad IPs and become similar to BLOCK-ALL.



(a) CD/N vs. F_{max}

(b) CD/N vs. C_{max}

Fig. 6. **Optimal Solution of FLOODING in Scenario III.** In (a), we show the normalized collateral damage, CD/N, as a function of the number of available filters, F_{max}, when C is fixed, $C = C_{max}$. In (b), we show CD/N as a function of the available capacity, C_{max}, when F is fixed, $F = F_{max}$.

C. Simulation of FLOODING and DIST-FLOODING

Simulation Scenario III. We consider a web server under 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 $F_{max} = 12,000$ filters are available (consistently with the discussion in footnote 1) and we vary $F = 1, ...F_{max}$. Before the attack, $5,000$ good sources are picked from [16] and utilize 10% of the capacity. During the attack, the total bad traffic is $10C = 1$Gbps and is generated by a typical blacklist (141,763 bad source IPs), based on

[2] Since 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.5(b)). In the final phase, the algorithm releases single good IPs, which are less important and all bad IPs are blocked similarly to BLOCK-ALL.

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_{max}, and of the bottleneck capacity C, respectively.

We simulate two baselines, namely *uniform rate-limiting* and *source address prefix filtering (SAPF)* [15], and we compare them to our approach. The *uniform rate limiting* approach drops the same fraction of traffic of all incoming flows, which is a common practice in DDOS attacks [19]. Since, in a typical DDOS attack, bad sources outnumber the good sources, uniform rate-limiting penalizes dis-proportionally the good sources. While this solution is always applicable and requires only one rate limiter, more filters (ACLs) can drastically reduce the collateral damage. We simulated one of the algorithms in [15], referred to as *SAPF positive*, which selects F_{max} prefixes to allow and denies traffic generated by all IP addresses outside those prefixes[4]. It uses clustering to find $O(F_{max})$ initial IP addresses to allow. Then, it greedily shortens the length of the allowed prefixes, thus allowing more traffic, until the available capacity is completely used. This heuristic solves the FLOODING problem but does not guarantee an optimal solution. In our simulations, the optimal solution found by our algorithm, for the same number of ACLs, reduces the collateral damage by about a factor of 2. Another observation is that, because of its heuristic nature, SAPF does not necessarily decrease monotonically with F_{max}, consistently with [15], which might result in inefficient use of the available filters. In summary, compared to [15], our work finds the optimal solution, at the lowest possible complexity (linear in the input size) and for a wider range of problems (including but not limited to FLOODING).

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 drastically 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. 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 types of resources resources (number of filters and capacity) increase.

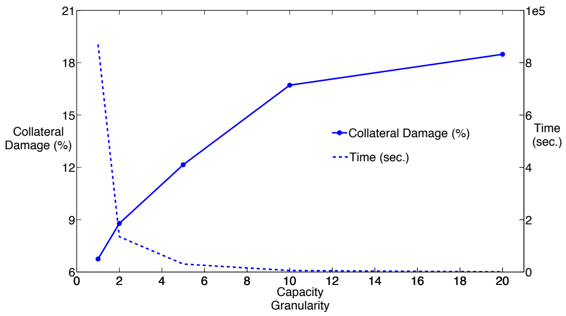


Fig. 7. **Heuristic solution of FLOODING in Scenario IV.** An approximate solution (higher CD) is obtained by solving only the sub-problems $z_p(f, n\Delta C)$ for $n \in \mathbb{N}$ and $C = n\Delta C \leq C_{max}$, $n = 1, 2\dots$. The coarser the capacity increments ΔC, the fewer sub-problems we need to solve, but at the cost of higher collateral damage. In this scenario, increasing ΔC 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.

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, $z_p(f, c)$, for all possible values of $f \leq F_{max}$ and $c \leq C_{max}$, we consider discrete increments of capacity $c = n\Delta C$, with step size ΔC. If $\Delta C = \min\{w_{ip}\}$, the finest granularity of c is considered, and the problem is optimally solved. If $\Delta C > \min\{w_{ip}\}$ we may get a sub-optimal solution, but we reduce the computation cost, as fewer iterations are required to solve the DP.

Simulation Scenario IV. We consider again a DDOS attack launched by 61,229 different bad source IPs, based on the Dshield.org logs. The available capacity, $C = 100$ Mbps. Before the attack, the legitimate traffic consumes $\frac{1}{2}C =$ 50Mbps. During the attack, the total bad traffic generated is $100C =$ 10Gbps. This scenario is more challenging than 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 a CD that is only 6% of the total good traffic. Larger values of ΔC 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.

[3] However, 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_{max}, C_{max}, w_i to maintain a constant ratio between the number of IPs and F_{max}, and, the total flow generated and C_{max}.

[4] In [15], there are three heuristics proposed for generating ACL lists: a *positive* algorithm (which specifies prefixes to allow and blocks all other traffic), a *negative* (which specifies prefixes to block and accepts all other traffic), and a *mixed* (with possibly overlapping prefixes). In this paper, we compare our optimal algorithm for the FLOODING problem only against the positive algorithm. The positive algorithm was found, in [15], to perform the best (*i.e.,* have the lowest collateral damage for the same number of ACLs), followed closely by the mixed algorithm. The negative algorithm performed the worse; this is why we do not compare against it, although it is more similar to our approach.

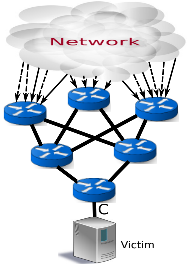


Fig. 8. **Distributed Flooding.** This topology exemplifies the part of a potentially larger ISP topology involved in routing and blocking traffic towards victim V. The edge routers receive all incoming, malicious and legitimate, traffic towards victim V and route it through shortest-paths with ties broken randomly. Any of the traversed routers (indicated with circles) can be used to deploy ACLs and block the malicious traffic.

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 through its ISP, as 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 λ, and avoid the redundant overlap of prefixes at multiple locations. This reduces the collateral damage significantly, *i.e.,* by $\sim 50\%$.

Increasing the number of routers has two effects. On one hand, it increases the total number of filters available to block source IPs and thus can potentially reduce the overall collateral damage. On the other hand, deploying more routers increases the communication overhead required to coordinate them. In Fig. 10 we study the trade-off between reduced collateral damage and increased communication overhead, as the number of routers increases. For simplicity, we assume that each router has the same number of filters, *i.e.,* $F_{max} = F^{u}_{max}\ \forall u \in \mathcal{R}$. As we can see in Fig. 10, increasing the number of routers provides a significant benefit in terms of reducing the collateral damage, while the communication overhead increases only linearly with the overall number of filters available, as discussed in Section . Depending on the bandwidth available and on the tolerable level of collateral damage, a network administrator can decide how many routers should be deployed to filter the attack sources. Our work provides the framework to do so in a principled way.

V. Related Work

The bigger picture. 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 optimize the use of filtering - a mechanism that already exists on the Internet today and is a necessary building block of any bigger solution.

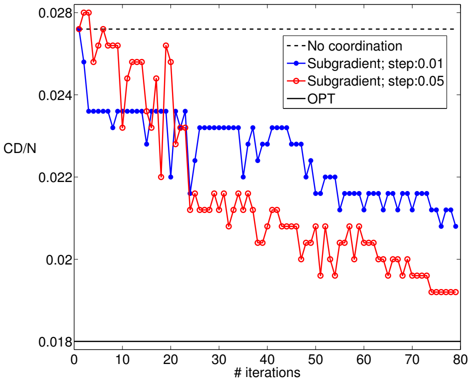


Fig. 9. **Evaluation of DIST-FLOODING in Scenario III.** Results are shown for the distributed algorithm and two different values for the step size (0.01 and 0.05) of the subgradient method. The dashed line shows the case of "No Coordination", *i.e.,* when each router acts independently.

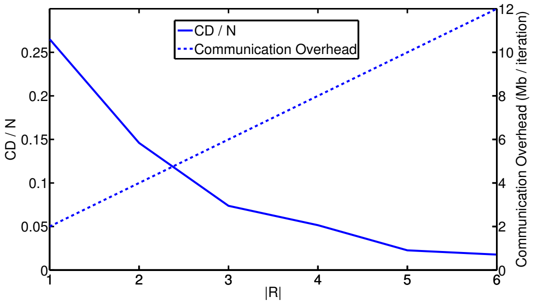


Fig. 10. **Distributed Flooding: CD/N and Communication Overhead vs.** $|R|$ On the x-axis is the number of routers, $|R|$. On the left y-axis is the normalized collateral damage, CD/N. On the right y-axis is the overall communication overhead (MB / iteration) required to coordinate routers. As CD/N decreases faster than the communication overhead increases, there is a sweet spot (3 routers, for this input blacklist) where we benefit from large reduction of CD/N while incurring a modest increase in the communication overhead required.

More specifically, we focus on the optimal selection of which prefixes to block. The filtering rules can be then propagated by filtering protocols [19]–[21], and ideally installed on routers as close to the attack sources as possible. We note, however, that such protocols typically assume the ability to filter traffic at arbitrarily fine granularity and focus on *where* to place the filters rather on *which* filters to pick. Therefore, they are complementary but orthogonal to this work.

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.

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 [20].

The problems studied in this paper are also related to firewall configuration to protect public-access 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, while we focus on resource allocation.

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 [15]. In this work, we exploit these characteristics for efficient prefix-based filtering of malicious traffic.

Prefix selection. The work in [15] studied source prefix filtering for classification and blocking of DDoS traffic, which is closely related to our FLOODING problem. The selection of prefixes in [15] was done heuristically, thus leading to large collateral damage. 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 to 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 the 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: $\sum_{j\in\mathcal{N}} w_j x_j \leq C, \sum_{j\in\mathcal{N}} x_j \leq k$. 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]. 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 simulation results.

In [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 and we developed 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].

VI. Conclusion

In this paper, we introduce a 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.

Acknowledgements

We are grateful to P. Barford and M. Blodgett at the University of Wisconsin, Madison, for making the 6-month `Dshield.org` logs available to us. We would also like to thank M. Faloutsos for insightful discussions.

References

[1] W. P. Cisco Systems, "Understanding acl on catalyst 6500 series switches," http://www.cisco.com/en/US/products/hw/switches/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_paper09186a00801a1a55.shtml.

[3] M. Collins, T. Shimeall, S. Faber, J. Janies, R. Weaver, M. De Shon, and J. Kadane, "Using uncleanliness 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. Spatscheck, J. Van Der Merwe, and R. Vasudevan, "Analyzing large ddos attacks using multiple data sources," in *Proc. of ACM 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] S. Venkataraman, S. Sen, O. Spatscheck, P. Haffner, and D. Song, "Exploiting network structure for proactive spam mitigation," in *Proc. of the USENIX Security Symposium*, Boston, MA, USA, Aug. 2007.

[8] Y. Xie, F. Yu, K. Achan, E. Gillum, M. Goldszmidt, and T. Wobber, "How dynamic are IP addresses?" in *Proc. of ACM SIGCOMM*, Kyoto, Japan, Aug. 2007.

[9] J. Zhang, P. Porras, and J. Ullrich, "Highly predictive blacklisting," in *Proc. of the USENIX Security Symposium*, San Jose, CA, USA, July 2008.

[10] Dshield.org, "Dshield: Cooperative network security community - internet security," http://www.dshield.org.

[11] H. Kellerer, U. Pferschy, and D. Pisinger, *Knapsack problems*. Springer Verlag, 2004.

[12] G. Varghese, *Network Algorithmics: an interdisciplinary approach to designing fast networked devices*. Morgan Kaufmann, 2005.

[13] S. Boyd and L. Vandenberghe, *Convex Optimization*. Cambridge University Press, 2004.

[14] T. Kanungo, D. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Wu, "An efficient k-means clustering algorithm: Analysis and implementation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 881–892, 2002.

[15] G. Pack, J. Yoon, E. Collins, and C. Estan, "On filtering of DDoS attacks based on source address prefixes," in *SecureComm*, Istanbul, Turkey, Sept. 2006.

[16] E. Kohler, J. Li, V. Paxson, and S. Shenker, "Observed structure of addresses in IP traffic," in *Proc. ACM SIGCOMM Workshop on Internet Measurment*, Marseille, France, Nov. 2002.

[17] Z. Chen and C. Ji, "Measuring network-aware worm spreading ability," in *Proc. of IEEE INFOCOM*, Anchorage, AK, USA, May 2007.

[18] R. Xu, D. Wunsch *et al.*, "Survey of clustering algorithms," *Neural Networks, IEEE Transactions on*, vol. 16, no. 3, pp. 645–678, 2005.

[19] R. Mahajan, S. Bellovin, S. Floyd, J. Ioannidis, V. Paxson, and S. Schenker, "Controlling high bandwidth aggregates in the network," *ACM Computer Communications Review*, vol. 32, pp. 62–73, July 2002.

[20] K. Argyraki and D. Cheriton, "Scalable network-layer defense against internet bandwidth-flooding attacks," *IEEE/ACM Transactions on Networking*, vol. 17, no. 4, pp. 1284–1297, Aug. 2009.

[21] X. Liu, X. Yang, and Y. Lu, "To filter or to authorize: Network-layer DoS defense against multimillion-node botnets," in *Proc. of the ACM SIGCOMM*, Seattle, WA, USA, Aug. 2008.

[22] "High Performance Packet Classification," www.hipac.org/performance_tests/results.html.

[23] E. Al-Shaer et al., http://www.mnlab.cs.depaul.edu/projects/SPA.

[24] S. Venkataraman, A. Blum, D. Song, S. Sen, and O. Spatscheck, "Tracking dynamic sources of malicious activity at internet-scale," in *Proc. of Neural Information Processing Systems (NIPS)*, Whistler, Canada, Dec. 2009.

[25] Y. Zhang, S. Singh, S. Sen, N. Duffield, and C. Lund, "Online identification of hierarchical heavy hitters: algorithms, evaluation, and applications," in *Proc. of the ACM SIGCOMM Workshop on Internet Measurement*, Taormina, Italy, Oct. 2004.

[26] C. Estan, S. Savage, and G. Varghese, "Automatically inferring patterns of resource consumption in network traffic," in *Proc. of ACM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, Karlsruhe, Germany, Aug. 2003.

[27] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava, "Diamond in the rough: Finding hierarchical heavy hitters in multi-dimensional data," in *Proc. of ACM SIGMOD International Conference on Management of Data*, Paris, France, June 2004.

[28] P. Gilmore and R. Gomory, "A linear programming approach to the cutting stock problem-Part II," *Operations Research*, vol. 11, no. 6, pp. 863–888, 1963.

[29] A. Caprara, H. Kellerer, U. Pferschy, and D. Pisinger, "Approximation algorithms for knapsack problems with cardinality constraints," *European Journal of Operational Research*, vol. 123, no. 2, pp. 333–345, 2000.

[30] V. Li and G. Curry, "Solving multidimensional knapsack problems with generalized upper bound constraints using critical event tabu search," *Computers & Operations Research*, vol. 32, no. 4, pp. 825–848, 2005.

[31] A. Bagchi, N. Bhattacharyya, and N. Chakravarti, "LP relaxation of the two dimensional knapsack problem with box and GUB constraints," *European Journal of Operational Research*, vol. 89, no. 3, pp. 609–617, 1996.

[32] I. de Farias Jr and G. Nemhauser, "A polyhedral study of the cardinality constrained knapsack problem," *Mathematical Programming*, vol. 96, no. 3, pp. 439–467, 2003.

[33] F. Soldo, A. Markopoulou, and K. Argyraki, "Optimal filtering of source address prefixes: Models and algorithms," in *Proc. of IEEE INFOCOM*, Rio de Janeiro, Brazil, Apr. 2009.

[34] F. Soldo, K. El Defrawy, A. Markopoulou, B. Krishnamurthy, and J. Van Der Merwe, "Filtering sources of unwanted traffic," in *Information Theory and Applications Workshop*, San Diego, CA, USA, Jan. 2008.

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 intrenships 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.

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.

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 Arastra Inc. (2005). Her research interests include network coding, network measurements and security, and online social networks. She received the NSF CAREER award in 2008.