Computer Communications 37 (2014) 64–76

Contents lists available at ScienceDirect

Computer Communications

journal homepage: www.elsevier.com/locate/comcom

Source address filtering for large scale networks ☆

Mingwei Xu [a,b], Shu Yang [a,b,*], Dan Wang [c], Fuliang Li [a,b], Jianping Wu [a,b]

[a] Department of Computer Science and Technology, Tsinghua University, Beijing, China
[b] Tsinghua National Laboratory for Information Science and Technology, Beijing, China
[c] Department of Computing, The Hong Kong Polytechnic University, Hung Hom, KL, Hong Kong

ARTICLE INFO

Article history:
Received 26 May 2013
Received in revised form 21 September 2013
Accepted 28 September 2013
Available online 9 October 2013

Keywords:
Source address filtering
Distributed filtering
Network security

ABSTRACT

Source address filtering is very important for protecting networks from malicious traffic. Most networks use hardware-based solutions such as TCAM-based filtering, however, they suffer from limited capacity, high power consumption and high monetary cost. Although software, such as SRAM, is larger, cheaper and consumes less power, the software-based solutions need multiple accesses in memory, which as a result bear much more additional lookup burden.

In this paper, we propose a new software-based mechanism. In our mechanism, routers cooperate with each other, and each only checks a few bits rather than all bits in source addresses. Our mechanism can guarantee the correctness, i.e., filtering all malicious traffic. We formulate it as an optimization problem where the loads across the network can be optimally balanced. We solve the problem by dynamic programming.

With the increasing number of filters, storage could also become a bottleneck for source address filtering. Our mechanism improves this by distributing filters among different routers. We re-formulate the problem by adding an additional storage constraint. Then we prove that the problem is NP-Complete, and propose a heuristic algorithm to solve it.

At last, using comprehensive simulations with various topologies, we show that the mechanism greatly improves both lookup burden and storage space. We conduct a case study on China Education and Research Network 2 (CERNET2), the largest pure-IPv6 network in the world. Using CERNET2 configurations, we show that our algorithm checks less than 40 bits on each router, compared with 128 bits in IPv6 addresses.

1. Introduction

Despite a significant breadth of research, malicious traffic problems such as DDoS attack and scanning, remain an important problem today [1]. Packet filtering is a prevalent mechanism for preventing malicious traffic. Due to the importance of source addresses, source address filtering is widely adopted in current ISPs. Traditionally, ingress routers will maintain a blacklist, which is the set of source addresses that should be filtered. During the past years, the blacklist has increased explosively, largely enabled by botnets and other platforms for launching attacks. This situations are even worse in large scale networks. In 2003, more than 20,000 sources appeared in an attack against an online betting site [2]. In 2007, a storm botnet was reported to include 50 million sources [3]. In 2008, more than 800,000 unique malicious IP sources addresses were reported everyday [4]. Under the devastating security crisis, the blacklists in ISPs are continually expanding to defend against malicious traffic from possible attackers.

To implement the IP blacklist, TCAM is currently the de facto industry standard. TCAM can achieve wire speed as it enables parallel matching [5]. However, TCAM stoarge space is limited due to its high cost and power consumption. The line-card in Cisco 12000, which is a typical core router, can only accommodate 20000 entries. With more and more malicious traffic, TCAM-based solutions can not accommodate so many entries, and can not defend against today's most severe attacks, not to mention larger attacks in the near future, where millions of sources are expected [6]. Limited storage even makes some TCAM-based solutions allow part of the malicious traffic for better aggregation [7]. Even worse, the growth of TCAM size can not keep pace with the explosively increasing number of filters in the foreseeable future.

Although software-based solutions can provide larger space and accommodate more filters, they are not widely used currently because they need multiple accesses during a single lookup. For

☆ The research is supported by the National Basic Research Program of China (973 Program) under Grant 2009CB320502, the National Natural Science Foundation of China (61073166), the National High-Tech Research and Development Program of China (863 Program) under Grants 2011AA01A101.

* Corresponding author. Address: Room 9-402, East Main Building, Tsinghua University, 100084 Beijing, China. Tel.: +86 (0) 10 62785822; fax: +86 (0) 10 6260364.

E-mail address: yangshu@csnet1.cs.tsinghua.edu.cn (S. Yang).

0140-3664/$ - see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.comcom.2013.09.013

example, current high performance routers usually use SRAM, and the largest SRAM chip has 144 Mb (288 Mb SRAM are on the roadmap of major vendors) [8], thus a large fraction of software-based solutions are using lookup tries [9]. Thus, software-based solutions may introduce large latencies and serious congestions, especially when facing burst traffic, despite their fast speed.[1]

Traditionally, the filters are stored in border routers, which is the choke point that transit traffic is sure to pass by. As a result, the border routers have to bear the additional processing burden. In this paper, we try to balance the load by designing a distributed mechanism where more routers can share the lookup burden. All routers along a path can work cooperatively to handle the source-IP filtering correctly. Such a design scales better facing increased filtering requirements, especially in large scale networks.

Although many distributed solutions already existed [12,6], they distribute tasks to routers by filters, that is, different routers checks different address blocks. Unlike previous solutions, our mechanism assigns different bits to routers. Such that each router only checks a few bits rather than all in source addresses. In this way, the load is balanced across the network, each router bears less additional lookup burden, and achieve fast lookup speeds more easily. The mechanism guarantees correctness, i.e., filtering all malicious traffic, by letting all routers along a path cooperatively check all bits in source addresses.

1.1. Simple example

To illustrate our basic idea, we use an example in Fig. 1. In the network, packets will travel through ingress router a towards egress router d, and there are three source prefixes to be filtered: 1**, 00* and 010. Conventionally, filters will be stored only at the ingress router a. Thus router a needs to access memory up to 3 times when a packet arrives, which brings heavy burden on router a. In our mechanism, each router only checks 1 bit, i.e., a checks the 0_{th},b checks the 1_{st}, and c checks the 2_{nd} bit. When a packet with source 010 arrives at router a, it will be delivered towards the egress router along the path $\{a,b,d\}$. With the new mechanism, router a checks the 0_{th} bit first, and moves the pointer from the root trie node to the 1_{st} level; then it passes the packet along with the intermediate pointer to router b, which checks the 1_{st} bit, moves the pointer to the 2_{nd} level and passes the information to c;c will check the 2_{nd} bit, concludes that the packet falls in the blacklist and should be filtered. In this way, each router bears less burden, and the load is balanced across the network. The amortized burden on each router would be much lower.

In this paper, we generalize the example by formulating it as an optimization problem where we need to balance the load across the network, given that (1) the total bit set to be checked, which can be computed using the blacklist; (2) the network topology information, including the location of ingress and egress routers; (3) the spare capacity on each router for source address filtering, in other words, each router has a limitation on the extra burden. To solve the problem, we develop a dynamic programming based algorithm, which can find the optimal solution.

Although SRAM provides larger storage space than TCAM, it could still be subject to bottlenecks considering the rapidly increasing number of malicious sources. To mitigate this problem, we propose that storage (like the uni-bit trie in Fig. 1) can be divided among multiple routers, such that each router only stores one part of the total storage to be looked up. We introduce a new problem by adding an additional storage constraint, and then prove the problem to be NP-Complete and propose a heuristic algorithm to solve it.

[1] The maximum clock rate of SRAM is 400 MHz, while TCAM is 266 MHz, the price of SRAM is 10–100 times lower than TCAM [10,11].

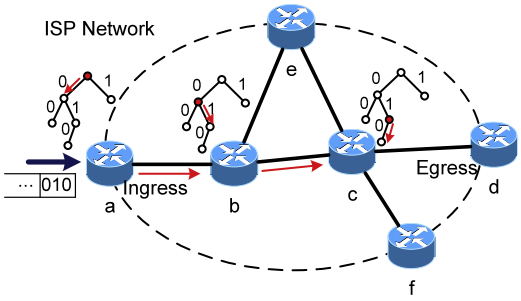


Fig. 1. Router a is the ingress and d,e,f are egress routers. The source address has 3 bits, and there exist 3 source filters: 1^{**}, 00^{*}, 010. The source filters are organized as a uni-bit trie.

To evaluate our mechanism, we conduct various simulations using both real and BRITE generated topologies. We show that our mechanism can balance the load across routers much better, and greatly reduce the number of bits that should be checked on each router. With storage constraint, our mechanism provides new room for storing large number of malicious sources. Using the real configurations from China Education and Research Network 2 (CERNET2, which is the world's largest IPv6 network, including 59 Giga-PoPs), we also conduct a case study. We show that, throught using our mechanism, each router in CERNET2 only needs to check at most 40 bits rather than whole 128 bits in IPv6 addresses. Using real data-traces, we also evaluate the overheads brought by our mechanism in both data and control planes. The results show that the overheads caused by our mechanism are quite low, this further prove that our mechanism is feasible in real networks.

The paper is organized as follows: We present the related work in Section 2. Section 3 is devoted to design overview of the new mechanism. We formulate the problem and present the optimal algorithm in Section 4. In Section 5, we take the storage constraint into consideration. Section 6 shows our implementation design. We evaluate our mechanism in Section 7, conduct a case study in Section 8, and conclude our paper in Section 9.

2. Related work

A significant body of research works have been devoted to battle against DDoS and spoof problems with filters. For example, most current networks use ingress access lists [13] or static ACLs (Access Control Lists) [14] to keep malicious traffic our of the networks. TCAM, which is a scarce resource, is the de facto standard for storing blacklist. However, with the exponentially increasing of blacklist, TCAM-based filtering fails to accommodate so many filters due to its limited storage space, high cost and high power consumption [15].

Due to lack of hardware memory space, many solutions have been proposed to reduce the number of filters. Pack et al. [16] reduces the number of source prefixes through aggregation. Yi et al. [17] utilizes bloom filters, which occupies much smaller storage space, to defend against malicious traffic. Some solutions [7,16] even allow part of the malicious traffic for better aggregation of source prefixes, which are likely to cause collateral damage. In [18], the bayesian decision theory based on attacking history is used to optimize the set of filters in blacklists.

In [6], a distributed filtering mechanism is studied to reduce collateral damage. To save the scarce TCAM resources, it resolves the problem as a resource allocation problem. With this mechanism, routers only stores part of all filters, and different routers defend against different IP address blocks along a path. However, the

scheme is based on ACLs that reside in TCAM, but in most cases only border routers have ACLs (e.g., smart edge network). Besides, the distribution results are computed based on the blacklists, once any filter changes, e.g., adding or deleting an IP prefixes, the routers have to re-compute and this increases additional control overheads. Compared with [4], our scheme only re-computes when the total bit set[2] that should be checked or topology changes.

In a previous paper [19], we have already presented the mechanism that optimally distributes bits to routers. In this paper, we will further present an improved mechanism that can also save storage space, because storage space in software-based solutions, especially in high performance SRAM, is also limited in current routers. Besides, we illustrate the implementation design and validate our results with more comprehensive evaluations.

In Table 1, we compare all possible schemes (distributed by bits& hardware scheme does not exist). We can see that our scheme is the only one that performs well with all metrics.

In this paper, we suppose that a network is defending itself without cooperation of other networks. There are many previous works that assume collaboration between different ASes or networks, such as "push-back" [20] and BGP black-holing [21]. Our work is orthogonal with them.

3. Design overview

3.1. Assumptions

To restrict the scope of our study, we first make a few assumptions: (1) We assume the existence of a blacklist, which can be constructed based on either historical data [3] or attacking information from other hosts [22]. Constructing the blacklist is orthogonal to our paper; (2) We assume that we can insert additional information between IP and MAC headers like MPLS, or in other positions so as to carry necessary information between adjacent routers; (3) We assume that the routers are less likely to be attacked and intra-domain communications are secure, despite our efforts to take fail-safe measures into account; (4) We admit that using our scheme, we do not completely prevent malicious traffic at the border routers. Some ISPs do tolerate the existence of malicious traffic inside their networks [12], but others may want to keep it away to prevent DDoS or other attacks [23]. We will take these factors into consideration and study them in more detail in our future work. We only focus on router cooperation in this paper, as this is the first step towards software-based distributed filtering.

3.2. Distributed software-based filtering

3.2.1. Trie creation process

A blacklist, i.e., a set of filters, exists on each ingress router. Traffic that enters into the network should be discarded, if it hits any of the filters on the ingress router. The blacklists can be obtained through existing methods. Different ingress routers may have different blacklists, because they may be connected to attackers from different sources.

We can use a trie to represent a blacklist. Here we use the simplest uni-bit trie, where each trie node is assigned with a unique index. In this paper, we call the trie *B-trie* (Blacklist-trie). Continuing the example in Fig. 1, given the blacklist on router *a*, we show the B-trie after assigning indexes to trie nodes. (see Fig. 2)

[2] When there are may filters, the total bit set changes much less frequently. Besides, we can set the total bit set to be all bits in source addresses, such that routers only re-compute when topology changes.

Table 1
Comparison between different filtering schemes.

Schemes	Metrics		
	Lookup speed	Storage space	Overheads[b]
Distributed by filters & Hardware	Fast	Large[a]	High
Distributed by filters & Software	Slow	Large	High
Distributed by bits & Software	**Fast**	**Large**	**Low**
Centralized & Hardware	Fast	Low	
Centralized & Software	Slow	Large	

[a] The storage space is larger if routers along the path have ACLs, however, only border routers have ACLs practically.
[b] Control overheads caused by distributed schemes.

3.2.2. Packet processing

Unlike ingress-based centralized filtering, where the ingress routers check all bits, the routers in our mechanism only check a portion of the total bits. Formally,

Definition 1. Given a blacklist, the set of bits that a router has to check are called **delegated bits** of the router for the blacklist.

For example, in Fig. 1, the delegated bits for router *a*,*b* and *c* is $\{0_{th}\}$,$\{1_{st}\}$ and 2_{nd} bits respectively.

Each router will maintain such an indexed B-trie together with the delegated bits. When a packet enters into the network, the ingress router will insert an additional header that has two fields: (1) an ingress router (IR) field that denotes the ingress router the packet goes through; and (2) a node index (NI) field that denotes the trie node where the lookup process should start. Initially, the NI field is set to be the index of the root node. When the packet leaves the network, the egress router will remove the additional header.

When a packet arrives at a router, the processing flow is shown in Fig. 3. The router should first judge whether the additional header exists or not. If it does not exist, the router just delivers the packet to the next hop. Otherwise, the router extracts the IR and NI field from the additional header. According to the IR field, the router can identify the blacklist (or B-trie) associated with the ingress router. According to the NI field, the router can locate a trie node in the B-trie. Starting from the trie node, the router looks up the delegated bits of the source address. If the lookup process stops at an intermediate trie node, the router will override the NI field with the index of the intermediate trie node, and deliver the packet to the next hop. If the source address matches a filter, the packet will be discarded. Otherwise, the router will remove the additional header and deliver the packet to the next hop, because the packet does not match any filter.

For example, in Fig. 4, we show the lookup process on router *c* in Fig. 1. Router *c* will receive a packet with an additional header, where the IR field denotes that the packet comes from ingress router *a*, and the NI field denotes that the routers should start looking up from trie node with index 5. Because the delegated bits of router *c* is $\{2_{nd}\}$, the router extracts the 2_{nd} bit from the source address, and gets its value 0. Thus we traverse to the left child with index 6. Because trie node 6 is related with the filter 010, so the packet matches a filter, and the packet will be discarded.

4. Optimal covering scheme

To optimally share the load among different routers across a network, we formulate the problem and present the algorithm in this section. We put more details in [19].

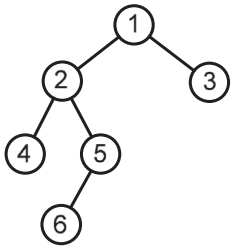


Fig. 2. B-trie after assigning indexes to trie nodes.

4.1. Problem formulation

$G = (V,E)$ is a network, where V represents the set of routers, and E represents the set of links. Let $\mathcal{R}$ be the set of ingress routers, where traffic enter into the network. Let $\mathcal{L}$ denote a path, which is a sequence (ordered set) of routers. Let $\mathcal{P}^r$ ($r \in \mathcal{R}$) be all possible paths that a packet may travel from r to egress routers. Let $\mathcal{T}^r = \{b_0, b_1, \ldots, \}$, ($r \in \mathcal{R}, 0 \leqslant b_i \leq 31$ for IPv4, and $0 \leqslant b_i \leqslant 127$ for IPv6) be the ordered set of bits (i.e., $b_i < b_j$, if $i < j$) that should be checked for traffic that enter into the network from $r \in \mathcal{R}$. Each router only checks part of the bits (called *delegated bits*) in source addresses, let $\mathcal{B}^r_v \subseteq \mathcal{T}^r$ denote the delegated bits that v should check for traffic from r, and $\overrightarrow{B}^r = (\mathcal{B}^r_{v_1}, \mathcal{B}^r_{v_2}, \ldots), v_i \in V$ be a vector that represents a *covering scheme*. Along any path from the ingress router r to any egress routers, all routers check $\mathcal{T}^r$ cooperatively from higher[3] to lower bits. The capacity of each router is constrainted due to limited CPU resources. We define this constraint as the maximum number of additional bits in source addresses that a router can look-up. Let C_v be the capacity of router v. To evaluate the processing load, let $f^r(v)$ be the function of utilization on router v for r, i.e., $f^r(v) = \frac{|\mathcal{B}^r_v|}{C_v}$.

Due to security considerations, some ISPs do not want malicious traffic penetrates deeply into their networks. To model this, we define *maximum depth*, which is the maximum hops that malicious traffic could travel into the networks. Let $d(u,v)$ denote the hops (or distance) between router u and v,k be the maximum depth, e.g., ingress filtering is a degenerate case where $k = 0$. ISP administrator can choose k to make a trade-off between security and load. The notation list of this section is in Table 2. The problem is formulated as following.

Problem 1. Given the set of bits to be checked $\mathcal{T}^r$ for an ingress router r, find a covering scheme $\overrightarrow{B}^r$, where (1) any bit in $\mathcal{T}^r$ will be covered by a router along any path $p \in \mathcal{P}^r$; (2) each router v checks less bits than its capacity, i.e., $|\mathcal{B}^r_v| \leqslant C_v$; (3) the successor router should check lower bits; (4) the router that is more than k hops away from r should not check any bit, such that the maximum utilization on all routers $\max_{v \in V, r \in \mathcal{R}} f^r(v)$ is minimized.

The solution to the problem is called the *optimal covering scheme*.

4.2. Finding the optimal covering scheme

In this section, we present the algorithm *Opt-Cover()* to find the optimal covering scheme using dynamic programming.

We can obtain a spanning tree rooted at an ingress router and towards all egress routers, and we call it *covering tree*, which has to cover all bits to be checked, more specifically, all paths from root to any leaf nodes should cover all bits.

Let $\mathcal{O}(v,n)$ be the Min-max utilization if covering tree rooted at node v has to cover n bits, $\mathcal{O}_j(v,n)$ be the Min-max utilization if covering tree rooted at node v has to cover n bits, and v itself has to check j bits. Let $\mathcal{N}(v,n)$ be the number of delegated bits on v if Min–max utilization is achieved on the covering tree rooted at v has to cover n bits. Let *Parent*(v) be the parent node of v on the covering tree. Algorithm Opt-Cover() computes the delegated bits of each node as follows.

```
2: Cons-Cover()
   Input        : H^r
   Output       : S^r_v, ∀v ∈ V
   Initialzation: S^r_v = ∅, ∀v ∈ V, cut ← all leaf nodes
1  begin
2    PreOrder traverse the Covering tree and push nodes into Stack
3    while Stack ≠ null do
4      v = Pop(Stack), t = |H^r| − 1 //*
5      [h]t is the total number of trie nodes
6      for i = 0, 1, ..., t do
7        if v is a leaf node then
8          if W(t − i, t) ≤ |S^r_v| then Q(v,i) = (|L^r(x_{t−i}) − L^r(x_t)|)/C_v, M(v,i) = i
9          else Q(v,i)=+∞, M(v,i) = i //*
10         [h]v has to store i nodes
11       else
12         for j = 0, 1, ..., i do
13           if W(t − i, t − i + j) ≤ |S^r_v| then
               Q_j(v,i) = (|L^r(x_{t−i}) − L^r(x_{t−i+j})|)/C_v  else Q_j(v,i) = +∞
             //*
14           [h]if v store j nodes, Min-max utilization is foreach child node u of
             v do
15             Q_j(v,i) = max{Q_j(v,i), Q(u,i − j)}
16         Q(v,i) = min_{j=0,1,...,i} Q_j(v,i)
17         M(v,i) = j, where Q_j(v,i) = Q(v,i)
18   if Q(r, |H^r| − 1) ≤ 100% then
19     PostOrder traverse the Covering tree and push node into Stack, pt = 0
20     while Stack ≠ null do
21       v = Pop(Stack)
22       S^r_v = {x_k, Parent(x_k)|pt ≤ k < pt + M(v,pt)}, pt = pt + M(v,pt)
23 end
```

The input of Algorithm Opt-Cover() is the bit set that has to be checked for traffic from an ingress router, and the output is the delegated bits on each router. In line 6, the node is a leaf node, thus it has to check all i bits. In line 8 to line 14, the node is an intermediate node, thus it can share the load with its children. If it checks j bits itself, then each of its children has to check $i - j$ bits, and the max utilization is computed through line 12, where current max utilization is stored in $\mathcal{O}_j(v,i)$. In line 12, we compute the optimal covering scheme (for the covering tree rooted at v) by selecting the scheme that results in minimum utilization.

Theorem 1. *Algorithm 1 finds the optimal covering scheme. The complexity of Algorithm 1 is* $O(|V| \times (|\mathcal{T}^r|^2))$.

Proof. We prove the first part by induction method. To prove that Algorithm 1 computes the optimal covering scheme, we only need to prove that $\mathcal{O}(r, |\mathcal{T}^r|)$ is the Min–max utilization. If v is a leaf node, then $\mathcal{O}(v,0) = 0$. If v is an intermediate node and $i \leqslant n$, suppose that Algorithm 1 computes $\mathcal{O}(u,k)$ correctly, where u belongs to the children of v and $k \leqslant i$. Then according to line 12, if v checks j bits, the optimal covering scheme would be $\max\{\mathcal{O}_j(v,i), \mathcal{O}(u,i-j)\}$, because $\mathcal{O}(u,i-j)$ is already the Min–max utilization on u. According to line 13, we compare all possibilities of dividing the bits between v and its sub-tree, and find the minimum, which is the optimal solution. Thus, $\mathcal{O}(v,i)$ is the Min–max utilization. Through reduction, we prove that $\mathcal{O}(r, |\mathcal{T}^r|)$ is the Min–max utilization.

We can see that *Stack* has less than $|V|$ nodes initially. The number of loops in line 5 and line 10 is less than $|\mathcal{T}^r|$. The number of loops in line 11 is bounded by a constant. Thus the theorem gets proved. □

[3] Here, we define the most significant bit as the highest order bit.

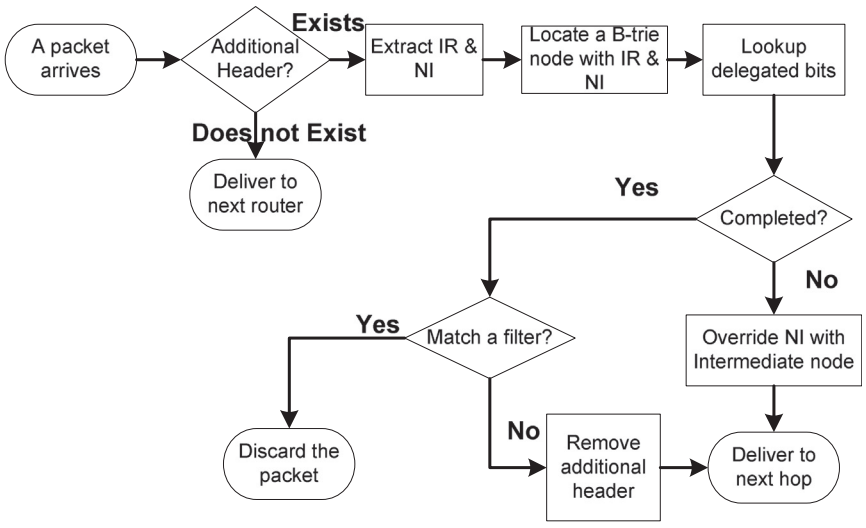


Fig. 3. Flow chart of packet processing on a router.

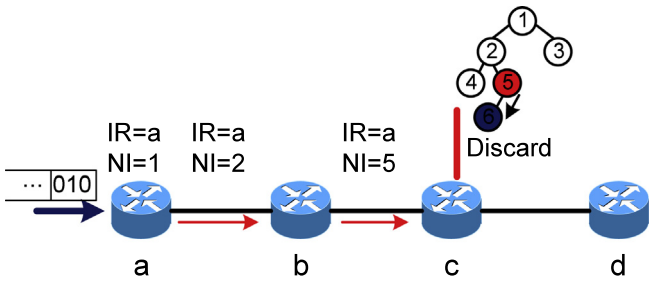


Fig. 4. Lookup process on router c in Fig. 1.

The complexity of Opt-Cover() is quite low, and linearly increases with V. Thus the algorithm brings only a few overheads in control plane, and adapts to large scale network.

4.3. Theoretical analysis of total delay

With the optimal covering scheme, the load on each router can be reduced. Although we focus on load balancing in this paper, distributed filtering also brings other benefits. One of them is reducing the transmission delay of a packet that traverses a network.

We only consider the delay caused by filtering. The ingress-based filtering can be seen as a simple M/M/1 queue model.[4] Suppose the traffic rate is N_p packets per second, then the arrival rate on the ingress router r is $\lambda = |\mathcal{T}^r| \times N_p$, which is the number of required memory accesses per second. The service rate is $\mu = C_r \times N_p$, which is the maximum number of memory accesses per second that the router can bear. According to the Little's formula [24], the average delay spent on filtering a packet is $\frac{1}{\mu-\lambda} = \frac{1}{(C_r-|\mathcal{T}^r|)\times N_p}$.

With distributed filtering, although more routers have to check the packets, each router will check less bits. Let $\hat{\mathcal{L}}$ be a path that packets will flow on. For each router $v \in \hat{\mathcal{L}}$, the service rate is still $C_v \times N_p$, while the arriving rate becomes $|\mathcal{B}_v^r| \times N_p$. Thus, the total delay is $\sum_{v\in\hat{\mathcal{L}}} \frac{1}{(C_v-|\mathcal{B}_v^r|)\times N_p}$.

We use an example to illustrate the benefits. Fig. 5 shows the total delay caused by filtering when a packet traverses the network under a specific configuration. We set the parameters as follows, $|\mathcal{T}^r| = 32$ (IPv4 address length), N_p = 100,000 (according to the traffic rate on a border router of CERNET2), $|\hat{\mathcal{L}}| = 3$ (less than the diameter of a common AS [25]), and $C_v = C_r, v \in \hat{\mathcal{L}}$ (assume all routers have the same capacity across the network). In Fig. 5, we can see that with ingress-based filtering, the total delay will bloat when the capacity of ingress router decreases. With distributed filtering, the total delay will increase much more slowly. Especially when the router capacity is close to $|\mathcal{T}^r|$, i.e., 32. For example, when router capacity is 32.1, the total delay reaches 100 ms with ingress-based filtering, but only 13 ms with distributed filtering. Note that if router capacity is less than $|\mathcal{T}^r|$, ingress-based filtering will cause packet loss (or miss), while distributed filtering still has enough capacity to handle the packets. The above results indicate that, with distributed filtering, we can use low/middle capacity routers to defend against large scale attack, which can save much upgrading cost for ISP providers.

Table 2
Notation list in Section 4.

Notation	Definition
V	the set of routers
$\mathcal{R}$	the set of ingress routers
$\mathcal{L}$	a path (including an ordered set of routers)
r	an ingress router
$\mathcal{P}^r$	all paths that a packet may traverse from r to egress routers
$\mathcal{T}^r$	the set of ordered bits that should be checked for packets from r
$\mathcal{B}_v^r$	the set of bits that node v checks for packets from r
$\overrightarrow{B}^r$	a covering scheme for the ingress router r
C_v	capacity of router v
$f^r(v)$	utilization function on router v for ingress router r
$\mathcal{O}(v,n)$	Min–max utilization if the tree rooted at v covers n bits
$\mathcal{O}_j(v,n)$	Min–max utilization if the tree rooted at v covers n bits, and v checks j bits

5. Optimal covering scheme with storage constraint

Although software-based mechanisms provides larger space, there may exist too many filters to be stored on a single router. For example, if a B-trie contains 3,000,000 filters, and each trie node in B-trie occupies 64 bits[5], then the total storage is almost

[4] Using M/D/1 model, we will get similar results.

[5] We use the simplest structure, where one trie node has two pointers, each occupies 32 bits.

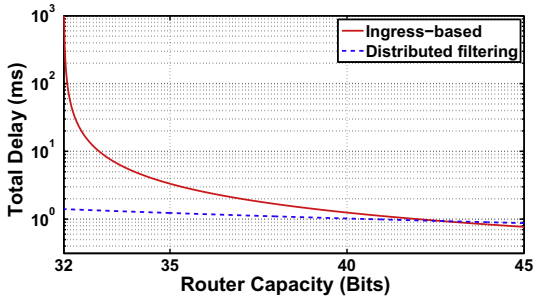


Fig. 5. Total delay as a function of router capacity.

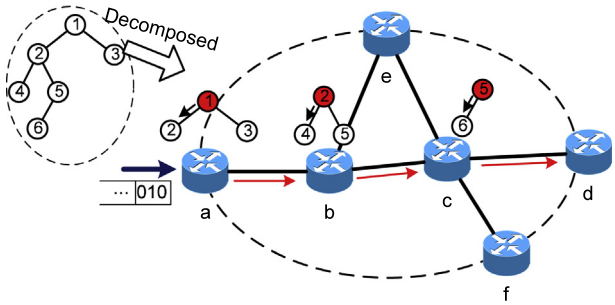


Fig. 6. Distributed source address filtering with storage constraint, where each router can only store 3 trie nodes.

192M bits, which is beyond the capacity of current largest SRAM chip available in the market [8].

Besides sharing load and sharing delay, sharing storage among routers is also possible with distributed filtering. For example, in Fig. 1, if each router can only store 3 trie nodes, then none router can accommodate the trie with 6 nodes. However, we notice that each router checks part of the total bits, such that each router only has to check part of the B-trie. Thus, each router only has to store a portion of all trie nodes. For example, in Fig. 6, we show that if router a stores trie nodes 1, 2, 3, router b stores trie nodes 3, 4, 5, and router c stores trie nodes 5, 6. The filtering process is the same with Fig. 1.

With storage distributed, the lookup process is the same with that in Fig. 3. But here we divide the B-trie by trie nodes, rather than bits. Formally,

Definition 2. Given a blacklist (B-trie), the set of trie nodes that a router has to store are called **delegated trie nodes** of the router for the blacklist.

Different with delegated bits, there exists intersection among the delegated trie nodes of different routers. For example, in Fig. 6, although router a has already stored trie node 2, router b still has to store trie node 2. This is because trie node 2 is the leaf trie node on a, while it is also the root node on b.

Thus, given the additional storage constraint, our objective is to find a covering scheme that minimizes the maximum utilization among all routers, and satisfies the storage constraint at the same time. Here the covering scheme indicates a scheme that distributes the trie nodes to routers across the whole network.

5.1. Problem formulation

Let $\mathcal{H}^r$ be the B-trie (a set of trie nodes) constructed by all filters in the blacklist on ingress router r. For a trie node $x \in \mathcal{H}^r$, we define $L^r(x)$ as trie level of x in the B-trie.[6] Let $\mathcal{S}_v^r \subseteq \mathcal{H}^r$ be the set of trie nodes that node v has to store for the blacklist on r. For two trie nodes $x, y \in \mathcal{H}^r$, define $x \sqsubseteq y$ as node x be the ancestor of y or y itself on $\mathcal{H}^r$. Each node has limited storage space, we use S_v to denote the maximum trie nodes that v can store. The notation list of this section is in Table 3.

[6] Let the root node be in the highest level of the B-trie.

Table 3
Notation list in Section 5.

Notation	Definition
$\mathcal{H}^r$	the B-trie constructed by all filters in the blacklist
x	a trie node in the B-trie
$L^r(x)$	the level of x in the B-trie
$\mathcal{S}_v^r$	the set of trie nodes v has to store for the blacklist on r
S_v	the maximum trie nodes v can store

The problem is different with the problem stated in Section 4.1: (1) there is an additional constraint on the storage space of each node; (2) all nodes along a path should cover all trie nodes rather than all bits. Then we re-formulate the problem with storage constraint as follows,

The solution to the problem is called the *optimal covering scheme with storage constraint*.

Problem 2. Given the B-trie $\mathcal{H}^r$ for an ingress router r, find a covering scheme with storage constraint $\overrightarrow{S}^r$, where (1) any trie node in $\mathcal{H}^r$ will be covered by a router along any path $p \in \mathcal{P}^r$; (2) each router v checks less bits than its capacity, i.e., $\max_{x \in \mathcal{S}_v^r} L^r(x) - \min_{x \in \mathcal{S}_v^r} L^r(x) \leqslant C_v, \forall v \in V, r \in \mathcal{R}$; (3) each router stores less trie nodes than S_v (4) the successor router should check children of trie nodes checked by previous routers, such that the maximum utilization on all routers $\frac{\max_{x \in \mathcal{S}_v^r} L^r(x) - \min_{x \in \mathcal{S}_v^r} L^r(x)}{C_v}$ is minimized.

5.2. Finding the optimal covering scheme with storage constraint

Although there is only one additional constraint, the problem with storage constraint is totally different with the problem without storage constraint. This is because: (1) the search space becomes much larger when we divide the B-trie by trie nodes; (2) the search space does not follow dynamic structures. For example, in Fig. 7, we show some of the covering schemes that satisfy the storage constraint continuing the example of Fig. 6. In this section, we will first prove that the constrained problem is NP-Complete, and then give an efficient heuristic algorithm to solve the problem.

Theorem 2. *Finding the optimal covering scheme with storage constraint is NP-Complete.*

Proof. It is easy to see that the decision problem of validating a given covering scheme is solvable in polynomial time. Therefore, finding the optimal covering scheme with storage constraint is in NP class. To show this problem is NP-hard, we reduce the Bisection of Tree problem to it; the former is known to be NP-Complete [26].

The Bisection of Tree problem is: given a trie T composed of n trie nodes, find a partition of the trie into 2 connected sets of size at most $\lceil n/2 \rceil$ each, such that the number of edges connecting trie nodes in different sets, called the cut size, is minimized.

Suppose there are two routers connected as a network, and one acts as the ingress router. The B-trie is T. Each router has limited storage capacity. The ingress router can store $\lceil n/2 \rceil$ trie nodes at most, while the other router can store at most $\lceil n/2 \rceil + k$ trie nodes. For $0 \leqslant k \leqslant |T|$, we try to find the optimal covering scheme with storage constraint. Suppose k is the minimum value that produces a feasible covering scheme, where T is divided into T_1 and T_2, and

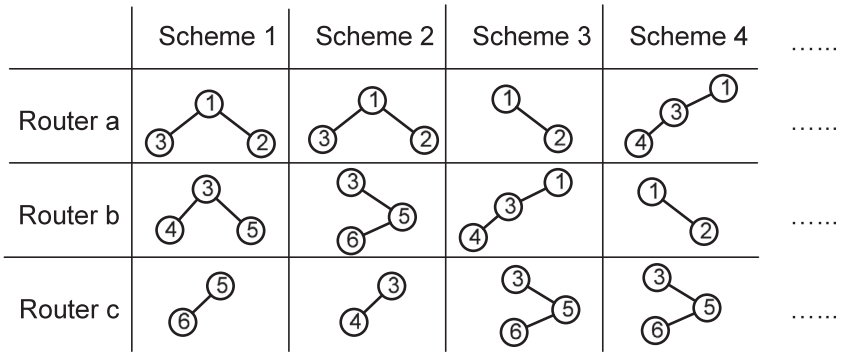


Fig. 7. Several covering schemes that satisfy the storage constraint given in Fig. 6.

the ingress router stores all trie nodes in T_1, while the other router stores all trie nodes in T_2. Then the Bisection of T is T_1 and $T_2 - T_1$, and the minimum cut size is $|T_2| - (|T| - |T_1|)$. This is because the downstream router must store all trie nodes on the edges that connect T_1 and T_2. □

Note that by using algorithm Opt-Cover(), we can also distribute the storage to different routers, such that each router only stores the trie nodes that are related with the corresponding bits. However, the partition is un-balanced. For example, if we replace the B-trie in Fig. 6 with another B-trie in Fig. 8(a). Suppose that each router can store 7 trie nodes. By using Opt-Cover(), we know that router *a*,*b* and *c* each should check the 0_{th},1_{st} and 2_{nd} bit, the partition of storage among routers *a*,*b* and *c* is shown in Fig. 8(b). Thus, router *a* stores 3 trie nodes, *b* stores 6 trie nodes, and *c* stores 12 trie nodes. We can see that *c* can not accommodate so many trie nodes, due to the fact that storage is unevenly distributed.

We therefore develop an efficient heuristic Algorithm *Cons-Cover()* to find the optimal covering scheme with storage constraint. This is different from the solution in Section 4.2, where the covering tree has to cover all bits. With storage constraint, the covering tree should cover all trie nodes of the B-trie.

Heuristically, we try to find a sub-optimal solution and achieve polynomial complexity, by reducing the search space of the problem. We enumerate all trie nodes with the principle of top-down-left-right, and name them x_0,x_1,x_2,... in sequence. We define node *v completely-covers (or C-cover)* trie node x_i, such that: (1) the ancestors of v on the covering tree do not have to store x_i (indicating v stores both x_i and parent of x_i); (2) if an upstream router C-covers trie node x_i, and a downstream router C-covers trie node x_j, then $i < j$. With this stricter definition, we can solve the problem using dynamic programming.

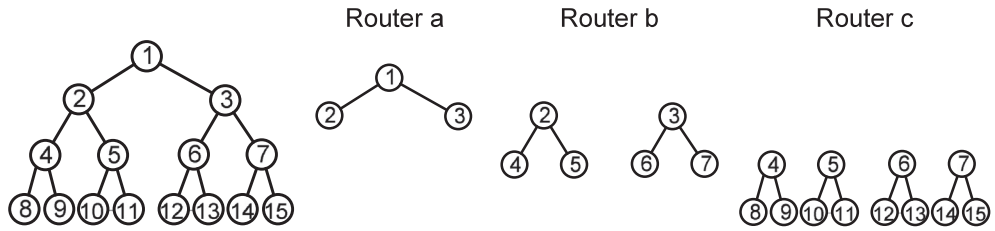
Intrinsically, if a covering tree has to cover a set of trie nodes $\{x_k, x_{k+1}, \ldots, x_{k+i-1}\}(0 \leqslant k \leqslant \mathcal{H}^r - 1), 0 < i \leqslant \mathcal{H}^r - k$, and the root node stores trie nodes $\{x_k$,x_{k+1}, ... ,$x_{k+j-1}\}(j < i)$. Then each sub-tree rooted at children of the root nodes has to store trie nodes $\{x_{k+j}$,-x_{k+j},x_{k+j+1}, ... ,x_{k+i-1},*Parent*(x_{k+j}),*Parent*(x_{k+j+1}), ... ,*Parent*(x_{k+i-1})$\}$. *For example, if the covering tree composed of routers a,b,c,e has to cover the B-trie in Fig. 8(a), and the root node a stores trie nodes {1,2,3,4,5,6,7}, then the sub-tree composed of b,c,e should store trie nodes {4,5,6,7,8,9,10,11,12,13,14,15}. Suppose that we know the optimal covering scheme with storage constraint for each sub-tree, then we can compute the optimal covering scheme with storage constraint of the entire tree by trying different number of trie nodes that would be stored at the root node.*

In Algorithm 2, let $\mathcal{Q}(v,n)$ be the Min-max utilization if the covering tree rooted at node v has to C-cover n trie nodes, $\mathcal{Q}_j(v,n)$ be the Min-max utilization if covering tree rooted at node v has to C-cover n trie nodes, and v itself has to C-cover j trie nodes. Let $W(i,j)$ be the number of trie nodes a tree node should store to C-cover trie nodes $\{x_k | i \leqslant k \leqslant j\}$, i.e., $W(i,j) = |\{x_k$,*Parent*$(x_k) | i \leqslant k \leqslant j\}|$. For example, in Fig. 8(a), $W(8,15) = 12$. Let $\mathcal{M}(v,n)$ be the number of trie nodes that v C-covers if Min-max utilization is achieved if the covering tree rooted at v has to C-cover n bits. Algorithm Cons-Cover() computes the delegated trie nodes for each node as follows.

```
1: Opt-Cover()
   Input          : T^r
   Output         : B_v^r, ∀v ∈ V
   Initialzation  : B_v^r = ∅, ∀v ∈ V
1  begin
2      PreOrder traverse the Covering tree and push nodes into Stack
3      while Stack ≠ null do
4          v = Pop(Stack)
5          for i = 0, 1, ..., |T^r| do
6              if v is a leaf node then
7                  O(v,i) = i/C_v,  N(v,i) = i  //*
8                  [h]v has to check all i bits
9              else
10                 for j = 0, 1, ..., i do
11                     O_j(v,i) = j/C_v  //*
12                     [h]If v itself checks j bits in i
                       //*
13                     [h]Then the Max utilization is
14                     foreach child node u of v do
15                         O_j(v,i) = max{O_j(v,i), O(u,i-j)}
16                 O(v,i) = min_{j=0,1,...,i} O_j(v,i)
17                 N(v,i) = j, where O_j(v,i) = O(v,i)
18     if O(r,|T|^r) ≤ 100% then
19         PostOrder traverse the Covering tree and push node into Stack
20         while Stack ≠ null do
21             v = Pop(Stack)
22             len(v) = len(Parent(v)) + N(v, |T| - len(Parent(v)))
23             B_v^r = {T_i^r | |T^r| - 1 - len(v) ≤ i < |T^r| - 1 - len(Parent(v))}
24 end
```

The input of Algorithm Cons-Cover() is the B-trie for an ingress router, and the output is the delegated trie nodes on each router. Similar with Algorithm Opt-Cover(), Algorithm Cons-Cover() also first traverses from leaf nodes to root node to compute the trie node number that each router has to C-cover. After computing the optimal covering scheme with storage constraint, we can compute the delegated trie nodes that each router has to store. In Fig. 9, we show the optimal covering scheme with storage constraint in Fig .8(a), where each router can store 7 trie nodes at most.

Theorem 3. *Algorithm 2 finds the optimal covering scheme. The complexity of Algorithm 2 is* $O(|V| \times (|\mathcal{H}^r|^2))$.



(a) A B-trie that has 15 trie nodes (b) The optimal covering scheme computed by Opt-Cover()

Fig. 8. An example for computing covering scheme with storage constraint using Opt-Cover().

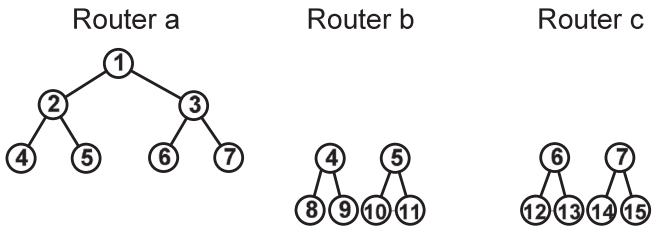


Fig. 9. Optimal covering scheme with storage constraint following the example in Fig. 8(a).

Proof. Using induction method, the proof for the first part is similar with the proof in Theorem 1. *Stack* has less than $|V|$ nodes initially. The number of loops in line 5 and line 10 is less than $|\mathcal{H}^r|$. Thus the theorem gets proved. □

6. Implementation design

We implement our mechanism in a centralized way, because (1) computing the optimal covering scheme requires a holistic view of the network which is more easily accomplished with a centralized approach; (2) routers do not have to do many additional computations.

6.1. Implementation overview

In Fig. 10, we show the overview of our implementation. We setup a centralized controller that will collect the information of network topology and filter set. The procedure of the implementation is as follows.

1. The controller obtains the network topology. This can be accomplished through existed IGP (Interior Gateway Protocol) protocols; For example, we can use OSPF as a simple, straightforward method to distribute these information, although OSPF does not scale well.
2. The ingress routers send the blacklist to the controller. Initially, the ingress routers send all filters in the blacklist to the controller. After that, the filters are updated incrementally;
3. The controller computes the optimal covering scheme (with or without storage constraint). It also generates configurations for each router, for example, the delegated bits (or trie nodes) that each router has to check (or store);
4. The controller sends the configurations to each router. Without storage constraint, both blacklist and delegated bits should be sent to routers. With storage constraint, only delegated trie nodes should be sent to routers;
5. After receiving and setting up the configurations, routers process the packets as in Fig. 3.

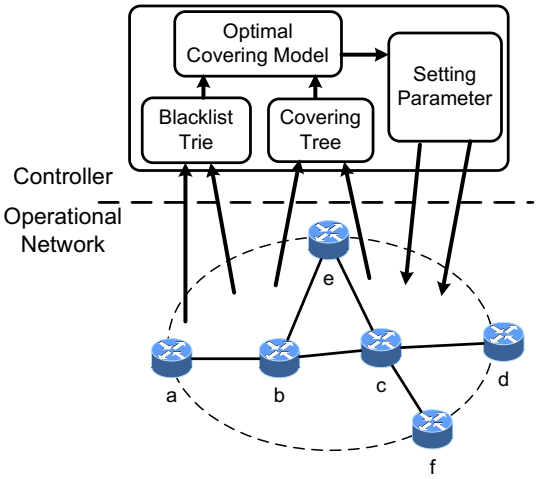


Fig. 10. Implementation overview.

6.2. Encoding and storing trie nodes

After constructing the trie, routers should encode each node with a node index. The encoding results should be consistent across different routers, i.e., different routers follow the same encoding rule. In our mechanism, we assign index 1,2,3,... one-by-one to each trie node following the principle of top-down-left-right. After encoding the trie, the router should store the nodes in memory, such that routers can access the trie node in constant time using the node index. Without storage constraint, routers just store all trie nodes linearly and continuity according to their indexes. For example, in Table 4(a), we show the storage structure that stores the trie in Fig. 8(a) on all routers without storage constraint.

However, with storage constraint, the continuity of trie nodes may be broken, for example, a router may store trie node set {4,5,8,9,10,11}. Thus the node index may not be equal to the physical position of each trie node. However, we notice that on a router, all entry trie nodes, i.e., the delegated trie nodes that are not children of any delegated trie nodes on the router, satisfy the continuity property. For example, in Fig .8(b), the entry trie nodes 4, 5 on router *b*, and the entry trie nodes 6, 7 on router *c* both satisfy continuity. This is because according to Algorithm Cons-Cover(), the trie nodes stored on a router are in the form of $\{x_k, x_{k+1}, x_{k+2}, \ldots, Par\text{-}x_k, x_{k+1}, x_{k+2}, \ldots, Parent(x_k), Parent(x_{k+1}), Parent(x_{k+2}), \ldots\}$ *and all trie nodes are encoded following the principle of top-down-left-right rule.*

Thus, we can store these entry nodes linearly in storage. When a packet arrives, the router extracts the node index and finds a trie node in storage by subtracting an offset. For example, Table. 4(b) shows that the storage structure on each router with storage constraint. On router *b*, node 4 and 5 are stored in physical positions 1 and 2. When a packet with node index 4 arrives, the router

Table 4
Storage structure without and with storage constraint following the example in Fig. 8.

Position	Node	Left Child	Right Child
(a) *Without storage constraint*			
All routers			
1	1	2	3
2	2	4	5
3	3	6	7
4	4	8	9
5	5	10	11
6	6	12	13
7	7	14	15
8	8	/	/
9	9	/	/
10	10	/	/
11	11	/	/
12	12	/	/
13	13	/	/
14	14	/	/
15	15	/	/
(b) *With storage constraint*			
Router a (offset: 0)			
1	1	2	3
2	2	4	5
3	3	6	7
4	4	8	9
5	5	10	11
6	6	12	13
7	7	14	15
Router b (offset: 3)			
1	4	3	4
2	5	5	6
3	8	/	/
4	9	/	/
5	10	/	/
6	11	/	/
Router c (offset: 5)			
1	6	3	4
2	7	5	6
3	12	/	/
4	13	/	/
5	14	/	/
6	15	/	/

subtracts the offset 3 from 4, thus we can get the physical position of node index 4. For other trie nodes except the entry nodes, the router also stores them linearly in storage. However, the routers should rewrite the left and right child values. For example, the left child of node 4 on router *b* is 3 rather than 8, because node 8 is currently in the physical position of 3. Centralized controller in Fig. 10 is responsible for these rewrite operations.

6.3. Discussions on some practical issues

- *Inter-operability*: To achieve better performance when defending against attacks, we have to change the routers. With the new mechanism, routers are no longer talking IP, and they have to understand a special shim header, compute the optimal covering scheme, and forward packets based on the new behaviors. This can be seen as a trade-off between security and inter-operability. Besides, many routers have already implemented shim headers, such as MPLS shim header. If not occupied, they can be used directly to carry the needed information.
- *Setting up and router failures*: During setting up and router failures, the optimal covering scheme (with or without constraint) may be not installed on some routers. During this time, we can use ingress-based filtering for a short time, and switch back to distributed filtering when all routers are synchronized. This will increase the utilization on ingress routers for a short period.
- *Topology and filter changes*: When topology changes, the controller has to re-compute and distribute a new covering scheme, it may introduce congestion for a short period of time. In practice, we believe this is tolerable for current routers, because network paths are normally stable (topology changes daily [27]). Filters, or blacklists, change more frequently. Without storage constraint, we can set the total bit set (which should be checked) to be all bits of source address, to prevent oscillation when the filter set changes. With storage constraint, we will study its incremental updates in our future work.

7. Performance evaluation

7.1. Simulation setup

We evaluate the algorithms using both BRITE [28] generated and real topologies. We will discuss a case study on CERNET2 in the next section.

7.1.1. BRITE topology

The sizes of the generated topologies are from 100 to 500. We set the network degree (average number of links per new router) to be 2 to 10. The number of bits to be checked is set to be 32 (for IPv4). On each router, the capacity constraint is 4 to 64. The number of border routers is 2 to 20 [29,30]. We randomly select one router as ingress router among all border routers, and others as egress ones. The maximum depth k is set to be infinity, i.e., malicious traffic should be filtered inside the network. The important default parameters are in Table 5. Other parameters are in [19].

7.1.2. Real topology

To evaluate the performance of our mechanisms in real topologies, We obtain the topology of CERNET, that is a medium-scale IPv4 ISP with 110 routers and 238 links in it. We also obtain four Rocketfuel topologies (AS 1221, AS 1239, AS 3257, AS 6461) according to [31]. The details of real topologies are also in [19].

7.1.3. Real blacklist

To evaluate the optimal covering scheme with storage constraint, we use data from Dshield.org [4] – a repository collecting intrusion reports from over 1000 organizations, and construct a B-trie using its blacklist. The blacklist contains 807,838 different source IPv4 addresses, and the B-trie has 1,498,253 trie nodes (after compression [32]). Every node has two pointers, each occupies 32 bits. Thus, the total storage requirement surpasses 90 Mb. Current largest SRAM chip in the market is 36 Mb [33], and most routers have less than 36 Mb spare SRAM storage space.

For comparison, we set ingress-base filtering as a benchmark. Our evaluation metric is the Min–max utilization (utilization for short) across the network. The results are averaged by 100 independent and random experiments. We make the source code developed in this paper public in [34].

7.2. Simulation results

7.2.1. Optimal covering scheme without storage constraint

Fig. 11 shows the impact of the network size and compares our mechanism Opt-Cover() with ingress-based filtering. The number

Table 5
Default parameter table of brite-generated topologies.

No. routers	No. border routers	Placement	Links/New router	k
300	5	Random	3	$+\infty$

of routers changes from 100 to 500. In Fig. 11, we can see that the Min–max utilization decreases with network sizes when using Opt-Cover(). As an example, if the network has only 100 routers, the Min-max utilization is 39.65%, when the network size increases to 500 routers, the Min–max utilization is only 29.74%. Clearly, when the network is larger, the paths from ingress to egress are stretched, where more routers can cooperatively share the load. Compared with the traditional ingress-based filtering, Opt-Cover() has much lower utilization. Opt-Cover() stays below 40% while ingress-based filtering stays around 155% (here, we do not limit the router capacity, i.e., utilization can exceed 100%). Actually, ingress-based filtering is insensitive with network size. This is obvious because the ingress router filters all malicious traffic. In Fig. 11, we also plot the error bar, which represents one standard deviation, we can see that the error bar of Opt-Cover() is much narrower, indicating that the performance of Opt-Cover() is much more stable, this is because that more routers can share the load and this reduce the unpredictability. We can also see that the bottom of the error bar of ingress filtering is still higher than the top of the error bar of Opt-Cover(). This indicates that the performance of Opt-Cover() outperforms ingress-based filtering with high confidence.

Fig. 12 shows the impact of network degree, i.e., links/(new router). We see that the Min–max utilization of Opt-Cover() increases when network degree increases, this is because that the paths a packet travels become shorter when degree increases. However, because the average degree of Internet is less than 3 [31,35], we believe that ISPs can obtain essential benefits using our mechanism.

In Fig.13, we study the relation between the number of border routers and Min-max utilization. We see that the Min-max utilization of Opt-Cover() increases with more border routers. For example, the utilization reaches almost 50% with 20 border routers. This is because more paths from ingress to egress routers lead to more constraints. However, most networks have <10 border routers [29], so our conclusion remains the same as Fig. 12. Besides, we can also see that the with of the error bar decreases with the number of border routers, this is because in our experiments, we repeatedly let each border router be the ingress router. Thus more border routers cause more experiments, and decrease the width of error bar.

Fig. 14 shows the performance of Opt-Cover() with different topologies. The results further prove the results we get using BRITE generated topologies. With kinds of topologies, the performance of Opt-Cover() is much better than traditional ingress-based filtering. And the performances of Opt-Cover() is related with the sizes and degrees of different topologies. For example, AS 6461 is small (has only 128 nodes) while the degree reaches almost 3.0, such that the Min-max utilization in AS6461 is the highest (35.78%) using Opt-Cover().

In Fig. 15, we show the trade-off between maximum depth and Min–max utilization. We can see that the Min-max utilization decreases with the maximum depth, because larger depth can make more routers share the load. For example, when the depth is 0(this degenerates into ingress-based filtering), the Min–max utilization is 149%. If the depth increases to be 1, the Min-max utilization decreases to be 67%. Thus, we can conclude that ISP networks can benefit a lot if they allow the malicious traffic into their networks, even by only one hop.

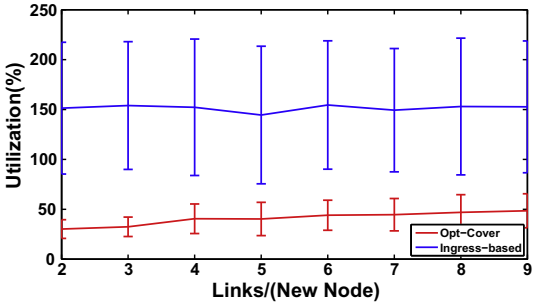


Fig. 12. Utilization as a function of network degree.

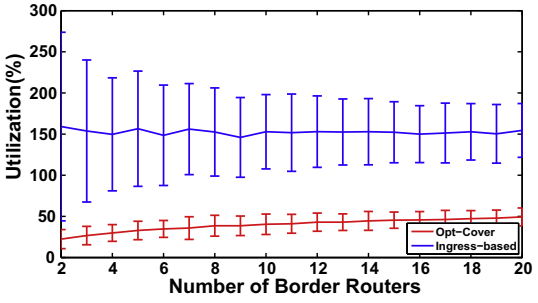


Fig. 13. Utilization as a function of number of border routers.

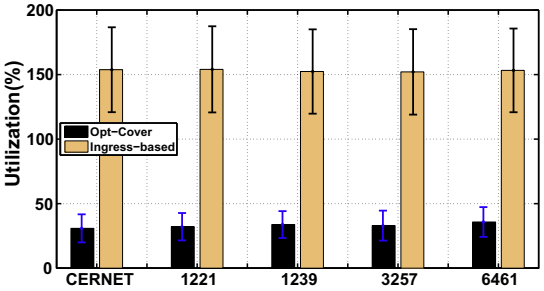


Fig. 14. Utilization on real topologies.

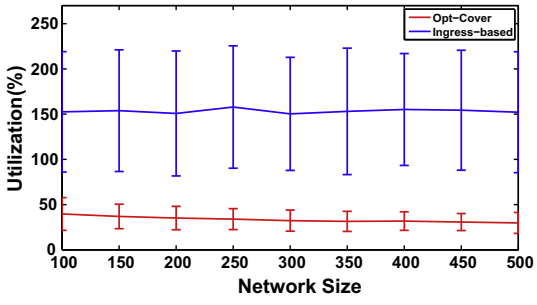


Fig. 11. Utilization as a function of network size.

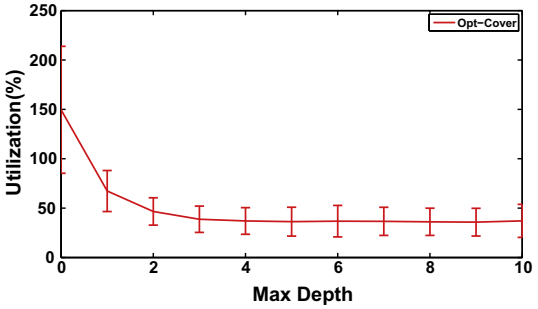
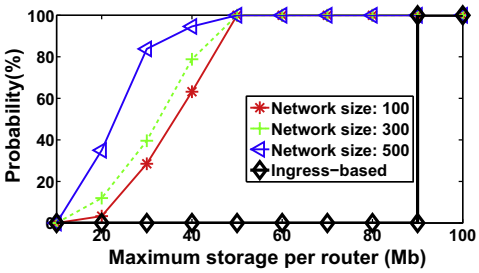
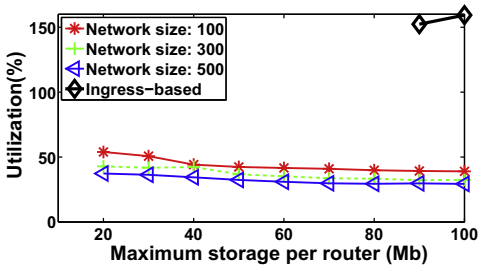


Fig. 15. Trade-off between maximum depth and utilization.

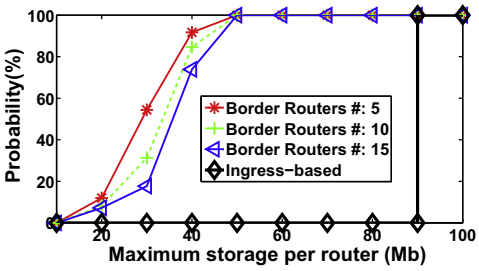


(a) Probability of complete block as a function of router storage with different network sizes

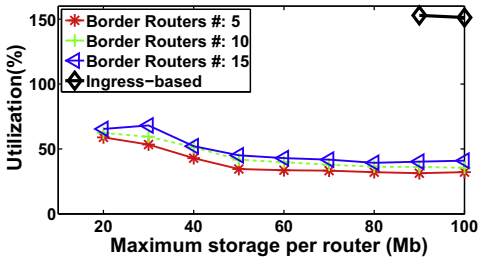


(b) Utilization as a function of router storage with different network sizes

Fig. 16. Comparison of Cons-Cover() and ingress-based filtering with different network sizes.



(a) Probability of complete block as a function of router storage with different number of border routers



(b) Utilization as a function of router storage with different number of border routers

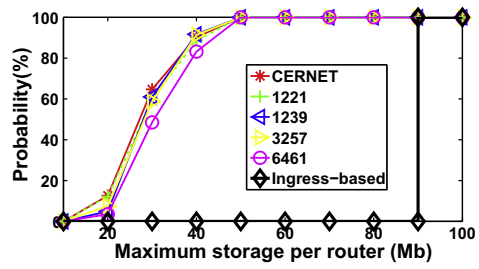
Fig. 17. Comparison of Cons-Cover() and ingress-based filtering with different number of border routers.

7.2.2. Optimal covering scheme with storage constraint

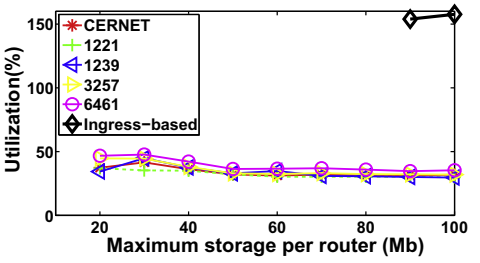
In this sub-section, we take storage constraint into consideration. We set the spare storage space of each router to be 10 to 100 Mb. Note that when routers do not have enough storage space, the blacklist can not be completely blocked. We define *complete block* as every filter (or source address) in the blacklist can be blocked by some routers.

Fig. 16(b) shows the relationship between probability of complete block and maximum spare storage per router, within different network sizes. We can see that with ingress-based filtering, the probability is zero when the storage space is less than 90 Mb, because complete block will never be possible unless the ingress router can accommodate the whole B-trie. With Cons-Cover(), complete block is possible with less storage on each router. For example, if each router has 50 Mb spare storage space, the probability is 100%. Because at least two routers are on the path that a packet flows on, we can divide the storage between them. When the storage per router is 20 Mb, the probability is about 35.0% with network size of 500, but only 3.3% with network size of 100. This is because when the network is larger, there will be more routers on the path to share the storage space.

Fig. 16(b) shows the min–max utilization as a function of storage space on routers with different network size. We can see that the utilization of ingress-based filtering still keeps around 150%, while the utilization of Cons-Cover() is much lower. When the router storage is larger, e.g., more than 50 Mb, the utilization of Cons-Cover() is similar with that of Opt-Cover(), because the storage constraint is almost always satisfied. However, when routers have less storage space, the utilization of Cons-Cover() will increase. For example, when the storage space is 20 Mb, the utilization reaches 53.9% with network size of 100. This is because less storage limits



(a) Probability of complete block as a function of router storage on real topologies



(b) Utilization as a function of router storage on real topologies

Fig. 18. Comparison of Cons-Cover() and ingress-based filtering with real topologies.

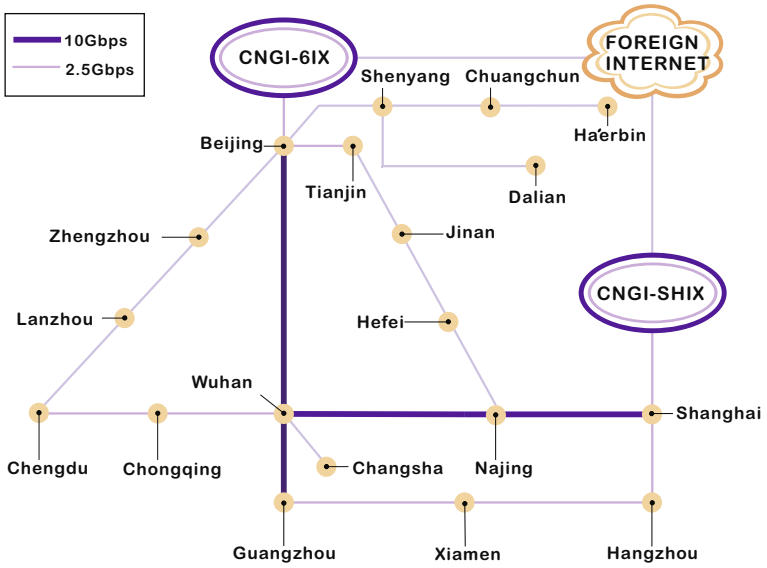


Fig. 19. CERNET2 topology.

the possible solution space. However, the utilization of Cons-Cover() is always smaller than ingress-based filtering.

Fig. 17 compares Cons-Cover() with ingress-based filtering with different number of border routers. We can see that Cons-Cover() can achieve complete block with less router storage, while ingress-based filtering needs more than 90 Mb. When the number of border routers decreases, the probability of complete block increases, because more routers on the path can share the storage. Besides, when the router storage space decreases, the utilization of Cons-Cover() will increase. For example, with 15 border routers and 30 Mb router storage space, the utilization of Cons-Cover() reaches 68.1%. However, it is still smaller than the utilization of ingress-based filtering.

At last, Fig. 18 compares Cons-Cover() with ingress-based filtering under real topologies. The results are similar with BRITE generated topologies. Under all topologies, Cons-Cover() can achieve complete block with less storage, the utilization of Cons-Cover() decreases with router storage, and is much smaller than ingress-based filtering. The performances under different topologies show slight differences. For example, for Cons-Cover() under AS 6461, the probability of complete block is smaller, and the utilization is higher than other topologies. This is because it has smaller network size and higher degree than other topologies.

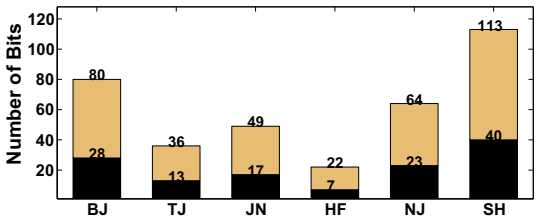


Fig. 20. Capacity of each router, and number of additional bits that each router has to check of source addresses.

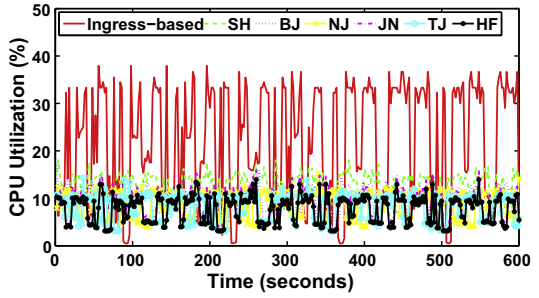


Fig. 21. CPU utilization on different routers.

8. A case study

Finally, using the real configurations of CERNET2 (which is the largest IPv6 network [36]), we conduct a case study. The topology of CERNET2 is in Fig. 19. CERNET2 has two international exchange centers connected with the foreign Internet, in Beijing (CNGI-6IX) and Shanghai (CNGI-SHIX). Currently, we want to filter malicious traffic along a pre-defined path from CNGI-6IX to CNGI-SHIX, i.e., {Beijing (BJ), Tianjin (TJ), Jinan (JN), Hefei (HF), Nanjing (NJ), Shanghai (SH)}. The capacity of each router is limited, as shown in the top bar of Fig. 20, estimated using the history data on each router.

We obtain the FIB (Forwarding Information Base) information of CERNET2 and use it as a whitelist to block malicious traffic.[7] The FIB of CERNET2 has 6973 IPv6 prefixes, which can be structured with a trie of 13280 trie nodes. Suppose that each trie node consumes 64 bits (each has two 32 bits pointers as usually implemented), thus the whole trie consumes less than 1 Mb storage. Storage is not a problem in this case, we only use Opt-Cover() to reduce the Min–max utilization.

Traditional ingress-based filtering takes place in either Beijing or Shanghai, and needs to check 128 bits in source addresses. The additional processing burden exceeds the spared capacity. And the maximum utilization among all routers reaches 160% when filters are only stored in Beijing.

Fig. 20 shows the results computed by Opt-Cover() in the bottom bar. The routers only need to check at most 40 bits (in Shanghai) in source addresses. The maximum utilization is only 36.11%, on the router of Tianjin.

To evaluate the overheads in data plane, we implement our scheme in Click router [37]. We deploy the Click in virtual machines on desktops with AMD Athlon II X2 245 2.91 G CPU and 2

[7] IPv6 is in the initial stage as an IPv6 network, so it does not have a blacklist. Note that Opt-Cover() and Cons-Cover() remains the same after using whitelist.

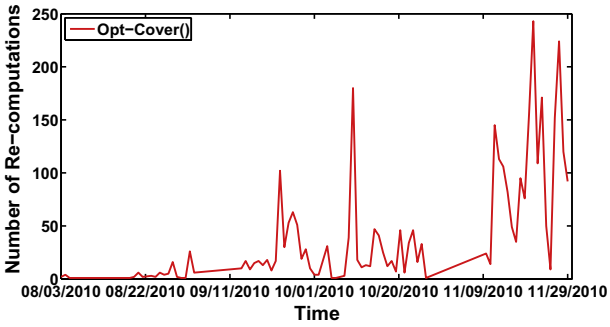


Fig. 22. Number of re-computations.

GB DRAM. We set up 6 routers, each representing a router on the path from Beijing to Shanghai. We assign virtual resources to each router in proportion to their capacity (the maximum one is equal to the capacity of the desktop). Besides, we collect 10 minutes traffic on the router of BJ during Oct, 2011. The total traffic volume is 457 GB. We replay these packets, and let them flow from router BJ to router SH.

Fig. 21 shows the CPU utilization of each router, including the router on BJ with ingress-filtering, and routers on BJ, TJ, JN, HF, NJ, SH with distributed filtering. We can see that with ingress-based filtering, the CPU utilization of the ingress router is much higher than other routers. For example, the maximum utilization of ingress-based filtering can reach 37.9%, while the maximum utilization among all routers of distributed filtering is only 18.1%. We can also see that the load is almost balanced among different routers when we use the distributed filtering scheme.

To evaluate the control plane overheads, we use the real data traces of topology changes during four months in 2010. In Fig. 22, we show the number of re-computations, which consumes most CPU resources on the controller. The maximum number of re-computation times can reach several hundreds/day. However, due to the low complexity of Opt-Cover() (it costs less than 1ms for one computation within CERNET2 topology, according to our experiments), it will not be a problem even for current PCs.

Currently, the CPU utilization of CERNET2 routers is less than 20%. By setting up a high performance centralized controller, we believe that the additional overheads in both data and control planes of our scheme are bearable in CERNET2.

9. Conclusion and future work

In this paper, we proposed a new distributed filtering mechanism, where routers inside networks can work cooperatively to filter the malicious traffic. This mechanism reduces the number of accesses to memory, and balances the load across the networks. We formulated the problem as finding a distributed scheme where the load is optimally balanced, and all bits are checked inside the networks. Our scheme can also reduce the memory storage on each router by letting each router store only part of the blacklist. We formulated the problem by adding an additional storage constraint. We proved that the new problem is NP-Complete, and proposed a heuristic algorithm to solve it.

Through simulations, we showed that the performance could be greatly improved through our mechanism under various topologies. Using case study in CERNET2, we proved that our scheme should be feasible in the real world.

References

[1] A. Yaar, A. Perrig, D. Song, Siff: a stateless internet flow filter to mitigate ddos flooding attacks, in: Proc. IEEE Symposium on Security and Privacy, Berkeley, CA, 2004.
[2] K. Argyraki, D.R. Cheriton, Scalable network-layer defense against internet bandwidth-flooding attacks, IEEE/ACM Trans. Netw. 17 (2009) 1284–1297.
[3] X. Liu, X. Yang, Y. Lu, To filter or to authorize: network-layer dos defense against multimillion-node botnets, in: Proc. ACM SIGCOMM'08, Seattle, WA, USA, 2008.
[4] Dshield dataset. <http://www.dshield.org>.
[5] V.C. Ravikumar, R. Mahapatra, Tcam architecture for ip lookup using prefix properties, Micro, IEEE 24 (2) (2004) 60–69.
[6] F. Soldo, A. Markopoulou, K. Argyraki, Optimal filtering of source address prefixes: models and algorithms, in: Pro. IEEE INFOCOM'09, Rio de Janeiro, Brazil, 2009.
[7] F. Soldo, K. El Defrawy, A. Markopoulou, B. Krishnamurthy, J. van der Merwe, Filtering sources of unwanted traffic, in: Information Theory and Applications Workshop, 2008, San Diego, California, 2008.
[8] C. Hermsmeyer, H. Song, R. Schlenk, R. Gemelli, S. Bunse, Towards 100 g packet processing: challenges and technologies, Bell Lab. Tech. J. 14 (2) (2009) 57–79.
[9] G. Varghese, Network Algorithmics: An Interdisciplinary Approach to Designing Fast Networked Devices, Morgan Kaufmann, Waltham, MA, 2005.
[10] W. Jiang, Q. Wang, V. Prasanna, Beyond tcams: an sram-based parallel multi-pipeline architecture for terabit ip lookup, in: Proc. IEEE INFOCOM'08, Phoenix, AZ, 2008.
[11] Network capabilities: The good, the bad, and the ugly.
[12] V. Sekar, R. Krishnaswamy, A. Gupta, M.K. Reiter, Network-wide deployment of intrusion detection and prevention systems, in: Proc. ACM CoNext'10, Philadelphia, Pennsylvania, 2010.
[13] F. Baker, F. Savola, Ingress Filtering for Multihomed Networks, RFC 3704, Best Current Practice, Mar. 2004.
[14] R. Beverly, A. Berger, Y. Hyun, k. claffy, Understanding the efficacy of deployed internet source address validation filtering, in: Proc. ACM IMC'09, Chicago, Illinois, USA, 2009.
[15] A. Liu, C. Meiners, E. Torng, Tcam razor: a systematic approach towards minimizing packet classifiers in tcams, IEEE/ACM Trans. Netw. 18 (2) (2010) 490–500.
[16] G. Pack, J. Yoon, E. Collins, C. Estan, On filtering of ddos attacks based on source address prefixes, in: Proc. IEEE Securecomm'06, Baltimore, MD, 2006.
[17] F. Yi, S. Yu, W. Zhou, J. Hai, A. Bonti, Source-based filtering scheme against ddos attacks, Int. J. Database Theory Appl. 1 (1) (2008) 2008–2011.
[18] M. Goldstein, C. Lampert, M. Reif, A. Stahl, T. Breuel, Bayes optimal ddos mitigation by adaptive history-based ip filtering, in: Proc. IEEE ICN'08, Cancun, Mexico, 2008.
[19] S. Yang, M. Xu, D. Wang, J. Wu, Source address filtering for large scale network: a cooperative software mechanism design, in: Proc. IEEE ICCCN'12, Munich, Germany, 2012.
[20] A. Keshariya, N. Foukia, DDoS Defense Mechanisms: A New Taxonomy, Springer, Berlin Heidelberg, 2010.
[21] D. Turk, Configuring BGP to Block Denial-of-Service Attacks, RFC 3882, Informational, Sep. 2004.
[22] F. Soldo, A. Le, A. Markopoulou, Predictive blacklisting as an implicit recommendation system, in: Proc. IEEE INFOCOM'10, San Diego, CA, 2010.
[23] K. chan Lan, A. Hussain, D. Dutta, Effect of malicious traffic on the network, in: Proc. Passive and Active Measurement Workshop (PAM), 2003.
[24] D. Gross, J. Shortle, J. Thompson, C. Harris, Fundamentals of Queueing Theory, Wiley, 2011.
[25] T. Lappas, K. Pelechrinis, M. Faloutsos, S. Krishnamurthy, A simple conceptual generator for the internet graph, in: Proc. IEEE LANMAN'10, Long Branch, New Jersey, 2010.
[26] A.E. Feldmann, L. Foschini, Balanced partitions of trees and applications, in: STACS'12, 2012, pp. 100–111.
[27] A. Markopoulou, G. Iannaccone, S. Bhattacharyya, C.-N. Chuah, Y. Ganjali, C. Diot, Characterization of failures in an operational ip backbone network, IEEE/ACM Trans. Netw. 16 (2008) 749–762.
[28] Brite: Boston university representative internet topology generator. <http://www.cs.bu.edu/brite>.
[29] B. Huffaker, A. Dhamdhere, M. Fomenkov, K. Claffy, Toward topology dualism: improving the accuracy of AS annotations for routers, in: Proc. PAM'10, Zurich, Switzerland, 2010.
[30] P. Barford, A. Bestavros, J. Byers, M. Crovella, On the marginal utility of network topology measurements, in: Proc. ACM IMW'01, San Francisco, California, USA, 2001.
[31] N. Spring, R. Mahajan, D. Wetherall, T. Anderson, Measuring isp topologies with rocketfuel, IEEE/ACM Trans. Netw. 12 (1) (2004) 2–16.
[32] R. Draves, C. King, S. Venkatachary, B. Zill, Constructing optimal ip routing tables, in: Proc. IEEE Infocom'99, New York, NY, 1999.
[33] Router fib technology. <http://www.firstpr.com.au/ip/sramip-forwarding/router-fib/>.
[34] Source code for source address filtering for large scale networks. <http://routing.netlab.edu.cn/project/twod-ip/code.rar>.
[35] M. Faloutsos, P. Faloutsos, C. Faloutsos, On power-law relationships of the internet topology, in: Proc. ACM SIGCOMM'99, Cambridge, Massachusetts, United States, 1999.
[36] J. Wu, J.H. Wang, J. Yang, Cngi-cernet2: an ipv6 deployment in china, SIGCOMM Comput. Commun. Rev. 41 (2) (2011) 48–52.
[37] The click modular router project. <http://www.read.cs.ucla.edu/click/click>.