2012 31st International Symposium on Reliable Distributed Systems

First Step Toward Cloud-Based Firewalling

Amir R. Khakpour Alex X. Liu
Department of Computer Science and Engineering
Michigan State University
{khakpour, alexliu}@cse.msu.edu

***Abstract*—With the explosive growth of network-based services and attacks, the complexity and cost of firewall deployment and management have been increasing rapidly. Yet, each private network, no matter big or small, has to deploy and manage its own firewall, which is the critical first line of defense. To reduce the complexity and cost in deploying and managing firewalls, businesses have started to outsource the firewall service to their Internet Service Providers (ISPs), such as AT&T, which provide cloud-based firewal service. Such firewalling model saves businesses in managing, deploying, and upgrading firewalls. The current firewall service outsourcing model requires businesses fully trust their ISPs and give ISPs their firewall policies. However, businesses typically need to keep their firewall policies confidential. In this paper, we propose the first privacy preserving firewall outsourcing approach where businesses outsource their firewall services to ISPs without revealing their firewall policies to the ISPs. The basic idea is that businesses first anonymize their firewall policies and send the anonymized policies to their ISP; then the ISP performs packet filtering based on the anonymized firewall policies. For anonymizing firewall policies, we use Firewall Decision Diagrams to cope with the multi-dimensionality of policies and Bloom Filters for the anonymization purpose. This paper deals with a hard problem. By no means that we claim our scheme is perfect; however, this effort represents the first step towards privacy preserving outsourcing of firewall services. We implemented our scheme and conducted extensive experiments. Our experimental results show that our scheme is efficient in terms of both memory usage and packet lookup time. The firewall throughput of our scheme running at ISPs is comparable to that of software firewalls running at businesses themselves.**

Keywords: Firewall; Cloud Computing.

I. Introduction

A. Motivation

Firewalls impose significant cost for most businesses especially smaller ones. Nemertes estimated the cost of firewall deployment and maintenance as $116,075 for the first year and an annual cost of $108,200 for a midsize US company with 5Mbps of Internet connectivity [1]. This high cost is not surprising. First, businesses need to hire administrators for firewall deployment, maintenance, monitoring, and tuning. Businesses also need to expend on training firewall administrators on new emerging firewall technologies. Second, as new firewall technologies are being developed and new types of attacks are being launched, operational firewalls often need to be upgraded to new ones with higher capacity and capabilities.

To reduce firewall management and deployment costs, businesses begin outsourcing their firewall services to ISPs. AT&T has started offering Network-Based Firewall Services (NBFWS) [2]. Compared with the Do-It-Yourself (DIY) firewalls, the three-year return for NBFWS is estimated at $126,735, a 38% savings [1]. In this service model, a business gives its firewall policy to its ISP and the ISP performs firewall filtering on the incoming and outgoing traffic of the business based on its policy. This model saves money for businesses from management, deployment, and upgrading perspectives. From the management perspective, by this model, businesses do not need to hire security administrators. From the deployment perspective, by this model, businesses do not need to purchase firewall devices and other related equipments such as redundant power supply and backup devices. From the upgrading perspective, by this model, businesses need neither to upgrade their firewall devices as businesses grow, nor to train their administrators for new firewall technologies. Beyond the above benefits, businesses also save bandwidth as most unwanted traffic is filtered out by ISPs in firewall outsourcing. This bandwidth saving becomes more of a life-saver when a business network is under Denial of Service (DoS) attacks. For ISPs, service fees collected from customers can be a significant source of revenue for their firewall outsourcing service. ISPs can use these fees to purchase advanced firewalls and skilled IT staff to provide professional firewall service. ISPs can further provide different types of packet filtering technologies based on different charging models. In general, outsourcing security services has been a trend driven by the needs to reduce management and maintenance costs for businesses [3]–[6]. For example, the percentage of businesses that outsource their security services was 37% in 2004 and 41% in 2008 [3].

While firewall outsourcing and NBFW services are gaining momentum on the market, the privacy issue of firewall policies has emerged as a serious concern because the current firewall outsourcing model requires businesses to reveal their firewall policies to their ISPs. Nemertes explicitly states that NBFWS requires businesses to trust their ISP [1]. However, it is very difficult for businesses to trust their ISPs. Businesses typically keep their firewall policies confidential. First, firewall policies often have uncovered security holes that can be exploited by attackers. Second, firewall policies often contain confidential information (such as IP addresses assigned to servers), which also can be helpful for attackers. In practice, even within an organization, often no employees other than the firewall administrator are allowed to access the firewall policies. Even an ISP can be trusted as an organization, its employees may not be trusted. Employee theft is a serious concern [7], [8]. International Survey reports that 88% of IT employees would consider stealing data from their employers if laid off [7]. Further, the survey reported that one-third of IT staff has admitted to snooping around the network to access commercially sensitive and confidential business information as well as other employees' files [7].

B. Our Approach and Technical Challenges

To address the privacy issue and the trust requirement between businesses and their ISPs, in this paper, we propose the first privacy preserving approach to firewall outsourcing. In

1060-9857 2012

DOI 10.1109/SRDS.2012.31

this approach, businesses first anonymize their firewall policies and send the anonymized policies to their ISP; then the ISP performs packet filtering based on the anonymized firewall policies. Note that as the ISP forwards or discards ongoing network traffic based on a business network access policy, it can partially find out the access policy, yet the entire access policy, which includes sensitive information about the business network and potential security holes remains unknown to the ISP. There are two fundamental technical challenges in firewall outsourcing. The first challenge is that the anonymization has to be done in a way that the ISP can correctly filter packets based on the anonymized policy but the ISP cannot derive the original policy from the anonymized policy. The second challenge is that the packet filtering operation on the ISP side should be fast enough to satisfy the performance requirements for businesses.

C. Our Firewall Outsourcing Framework

In this paper, we propose Ladon, a privacy-preserved firewall outsourcing framework, in which a firewall access policy is anonymized using ciphered data structure by the organization and delivered to the ISP. The ISP uses the anonymized access policy to filter organizations's inbound and outbound traffic. To build such data structure that address both anonymity and performance challenges, we use Firewall Decision Diagram (FDD) [9] to represent an access policy as a conflict-free range-based decision tree over packet header fields. In addition, we use Bloom Filters [10] to represent sets and ranges in an FDD to preserve the privacy of access policy elements. Using bloom filters, all access policy elements are hashed into bit strings, so that the ISP cannot reveal the access policy rules. Using bloom filters causes new challenge. For example, as domain sizes of packet fields can be very large (*e.g.* for IP fields the domain size is 2^{32}) the bloom filters can be large and space consuming. Using our observations on FDDs for real-life firewalls that indicate that the bloom filters are highly sparse, we use some simple yet effective lossless compression methods to gain reasonable bloom filter sizes. In addition, to avoid bloom filter false positives that may cause packet misclassification, we query multiple versions of an anonymized firewall. The recommended design in this paper indicates that we can reduce false positive rate for each packet field to 10^{-12} with a small memory penalty. In other words, using this design the expected value of number of packet headers that may be misclassified is smaller than 0.0019 which is significantly less than one packet header. Moreover, our experiments show that the outsourced firewall throughput can goes up to 1.143Gbps. We call this framework Ladon, as in Greek mythology Ladon is a hundred-headed dragon who guards the Hesperides garden and a tree with golden apples.

Figure 1(a) shows an example DSL network where organizations' firewalls are outsourced to ISP Digital Subscriber Line Access Multiplexer (DSLAM). In this scheme, the ISP provides a simple software package to an organization to describe its access policy by some rules for its inbound/outbound traffic. The output consists of multiple versions of the organization anonymized firewall whose rules cannot be deanonymized except by brute forcing. The anonymized firewalls are sent to the ISP to be used for enforcing access policies on the ISP side. Figure 1(b) shows a similar scenario on Layer 3 VPN MPLS networks. In such networks, each customer site can outsource its local access policy from its Customer Edge (CE) router to its Provider Edge (PE) router while the privacy of its access policy is preserved.

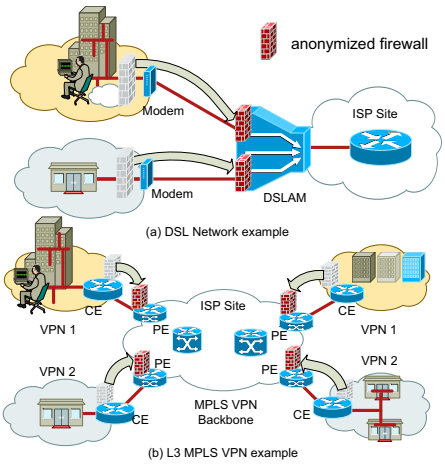


Fig. 1. Firewall outsourcing

D. Key Contributions

We make three contributions in this paper. First, we propose Ladon, the first firewall outsourcing framework that preserves the privacy of firewall rules. Second, we provide a suite of algorithms for anonymized firewall construction and querying. We introduce Bloom Filter Firewall Decision Diagram (BFFDD) as a data structure to represent the anonymized firewall as well as proving optimization techniques to improve its performance. Third, we implement Ladon and we evaluate its performance using 40 real-life firewalls collected from small businesses.

The rest of the paper proceeds as follows. We first review related work in Section II. We then introduce Bloom Filter Firewall Decision Diagram (BFFDD) and provide a detailed suite of algorithms for anonymized firewall construction and query for packet processing in Section III. In Section IV, we explain and discuss the optimization techniques using which we improve performance metrics such as privacy, query time, required memory. System privacy analysis is formulated and discussed in Section V. Section VI is dedicated to experimental evaluation. Finally, we give concluding remarks in Section VII.

II. Related Work

To our best knowledge, there is no prior work on privacy-preserving outsourcing of firewalls. In this section, we review two threads of work that are related to ours: (1) collaborative enforcement of firewall policies in Virtual Private Networks (VPNs) and (2) structure-preserving router configuration anonymization.

Collaborative enforcement of firewall policies in Virtual Private Networks has been studied in [11], [12]. The purpose of such work is to allow a firewall policy owner and a request owner (*i.e.* packet owner) to collaboratively determine whether a packet satisfies the policy *without the policy owner knowing the packet and the packet owner knowing the firewall policy*. When a visitor residing in a host network connects to the VPN server in his home network through a VPN tunnel, the incoming and outgoing packets sent in the tunnel cannot be verified by the host network firewall policy. But using such scheme, the host network firewall can enforce its

policy on the traffic in the VPN tunnel between the visitor computer and the VPN server. However, we cannot use such collaborative firewall policy enforcement schemes developed for VPN environments for firewall outsourcing because in firewall outsourcing both parties (*i.e.* the customer and the ISP) can see every packet in and out of the customer network.

In [13], Maltz *et al.* proposed a scheme for anonymizing router configurations, including access control lists (ACLs), based on prior techniques for prefix-preserving anonymization of packet traces [14]–[16]. Such schemes cannot be used for firewall outsourcing because the anonymization process does not preserve ACL semantics.

III. System Design

In this section we first describe the firewall outsourcing model and then present two basic algorithms: (1) offline construction of an anonymized firewall on the customer side and (2) querying an anonymized firewall for processing inbound/outbound traffic on the ISP side.

A. Modeling

1) System Model: To outsource a firewall, an anonymized version of the firewall is constructed on the customer side and sent over as a compact data structure to the ISP. The anonymized firewall is installed on the ISP gateway interface such that all inbound and outbound traffic of the customer can be checked against the anonymized firewall access policy. As a principle in this framework, the anonymized firewall should be designed such that the access policies remain undisclosed to the ISP, while ISP must be able to check the inbound and outbound traffic against the user access policy. Any incremental changes in ACLs require recalculation of anonymized firewall. Thus, the offline firewall construction algorithm must be efficient enough so that a customer can apply changes within reasonable delay. In practice, the ISP can offer a software package for the customer to design its rules and make its incremental changes. The software later creates the anonymized firewall and directly sends it to the ISP. Note that for outsourcing stateful firewalls, the traffic state table is kept on the ISP side as well. Using this architecture, the customer firewall can be fully outsourced to its ISP.

2) Threat Model: Network operators, in general, are more comfortable to outsource their access control policy such that the privacy of their rules is preserved. This is because, as they have no knowledge and control on the ISP side, they want to ensure that in case of employee theft or data breach on the ISP side, their access control policies are secure and undisclosed to a certain extent. Note that if the access control policies are exposed, the network administrator may need to change the network topology, the host addresses and the network configurations to hide its sensitive resources. This operation can be very expensive in terms of time and probable network outages. The basic assumption for this threat model is that a "bad" employee has limited resources to deanonymize the firewall rules. Thus, the anonymized firewall should be designed such that it is very difficult to be deanonymized in the time gap between two courses of incremental changes.

B. Offline Anonymized Firewall Construction

To anonymize a firewall, we follow two-step anonymized firewall construction algorithm in which we first convert an ACL to an equivalent Firewall Decision Diagram (FDD). Firewall Decision Diagram was introduced by Gouda and Liu in [9] as a compact data structure for representing ACLs. Second, we use Bloom Filters [10] to represent edge sets in the firewall decision diagram.

A firewall decision diagram with a decision set DS, over fields $F_1, \cdots, F_d$ is a directed and acyclic graph (DAG) that has the following five properties: (1) There is only one node called the *root* that has no incoming edges. The leaf nodes that have no outgoing edges are called *terminal* nodes. (2) Each node v has a label, denoted $F(v)$, such that $F(v) \in \{F_1, \cdots, F_d\}$ if v is a non-terminal node and $F(v) \in DS$ if v is a terminal node. (3) Each edge $e{:}u \rightarrow v$ is labeled with a non-empty set of integers, denoted $I(e)$, where $I(e)$ is a subset of the domain of u's label (*i.e.*, $I(e) \subseteq D(F(u))$). (4) A directed path from the root node to a terminal node is called a *decision path*. The node labels on decision paths are unique. (5) The set of all outgoing edges of a node v, denoted $E(v)$, satisfies the following two requirements: (i) *Consistency*: $I(e) \cap I(e') = \emptyset$ for any two distinct edges e and e' in $E(v)$. (ii) *Completeness*: $\bigcup_{e \in E(v)} I(e) = D(F(v))$.

An FDD construction algorithm, which converts a sequence of range rules to an equivalent full-length ordered FDD, is described in [17]. A full-length ordered FDD is an FDD where in each decision path all fields appear exactly once and in the same order. In order to bring down the FDD size expansion, we perform FDD reduction. An FDD is *reduced* if and only if it satisfies the following conditions: (1) no two nodes are isomorphic; (2) no two nodes have more than one edge between them. Two nodes v and v' in an FDD are *isomorphic* if and only if v and v' satisfy any of the following conditions: (1) both v and v' are terminal nodes with identical labels; (2) both v and v' are non-terminal nodes and there is a one-to-one correspondence between the outgoing edges of v and the outgoing edges of v' such that every pair of corresponding edges has identical labels and both edges point to the same node. An efficient FDD reduction algorithm that processes the nodes level by level from the terminal nodes to the root node using signatures to speed up comparisons is in [18]. For ease of presentation, in the rest of this paper, we use the term "FDD" to mean "reduced full-length ordered FDD" if not otherwise specified.

To construct the anonymized firewall, we use bloom filters to represent the edge sets in a given FDD. A bloom filter [10] is to represent a set $S = \{s_1, s_2, \cdots, s_n\}$ of n elements by a bitmap with size m. The filter uses k independent hash functions $h_1, h_2, \cdots, h_k$ with value range $\{0, 1, \cdots, m-1\}$. For mathematical convenience, the basic assumption is that the hash functions map each element to a random number uniformly in the range $\{0, 1, \cdots, m-1\}$. For each element $s \in S$, the bits $h_i(s)$ are set to 1 for $1 \leq i \leq k$. Given an item x, to verify if it is in the set, we check if all $h_i(x)$ are all set to 1 in the bitmap. If not, x is *definitely* not a member in S. If they are all set to 1, x is *probably* a member in the set. This implies that bloom filters can have false positives, but they do not suffer from false negatives.

The Bloom Filter FDD (BFFDD) is a new data structure where for a given edge e, the edge set $I(e)$ is represented by a bloom filter. Given an FDD, to construct a BFFDD, we traverse the FDD and convert edge sets to their corresponding bloom filters.

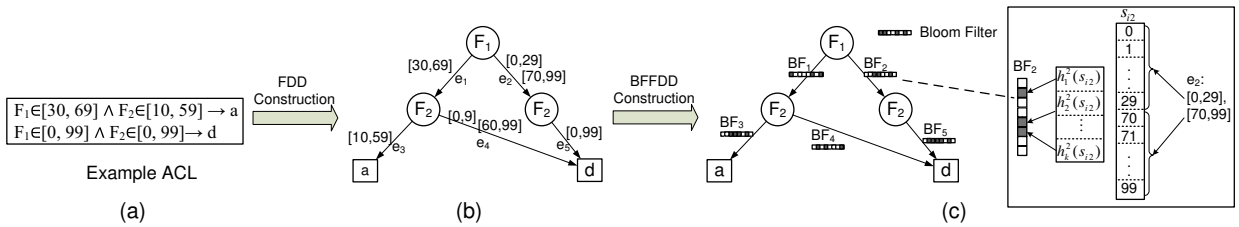


Fig. 2. An example of anonymized firewall construction: (a) Example ACL (b) Constructing a reduced FDD (c) Converting to a Bloom Filter FDD (BFFDD)

Figure 2 shows the two step anonymized firewall construction algorithm. In the first step, an example ACL over two fields (*i.e.* F_1 and F_2) with overlapping rules (Figure 2(a)) is converted to an FDD shown in Figure 2(b). In the second step shown in Figure 2(c), the edge sets are converted to their corresponding bloom filters. In this figure, we show how the edge sets of edge e_2 are represented by bloom filter BF_2.

C. Online packet filtering

Packet filtering in BFFDDs is different from packet filtering in general FDDs explained in [9]. This is because bloom filters may have false positives. This implies that although bloom filters on each level represent non-overlapping edge sets and their false positive rate is very small, some packet fields can match against more than one outgoing edge of a node in an FDD. This leads to multiple decision paths with possible multiple decisions. To deal with multiple decisions problem, we use N independent BFFDDs; thus, in case querying one BFFDD ends up with multiple decisions, we can refer to all other $N-1$ BFFDDs to detect false positives. Note that by N independent BFFDDs, we mean N BFFDDs constructed based on a single FDD but with different and independent hash functions for the bloom filters.

To query from a BFFDD and find the decision for a given packet, we need to check the packet header fields against the edges in the BFFDD. On each level of traversing the BFFDD, the corresponding packet header field is checked against bloom filters on all outgoing edges. Based on the completeness and consistency properties of FDDs, the packet field is found in at least one bloom filter on an outgoing edge as bloom filters have false positive. The result is a subtree with possibly multiple decision paths leading to decision nodes. As this procedure is repeated for N independent BFFDDs, the result is N subtrees. For each subtree, if the final decisions are the same, the decision is the final decision for a given packet. Otherwise, as bloom filters do not have false negative, all result subtrees should be common in one decision path whose decision is the final decision for a given packet. If N result subtrees share more than one decision path with multiple decisions, system administrator may assign a default decision to be chosen as the final decision for a given packet. We will show in Section V using recommended design parameters, the likelihood of false classification using this scheme is trivial. Nevertheless, such errors are detectable by the administrator and can be handled accordingly.

IV. System Optimization

A. Design Parameters

We design the bloom filters for each FDD level based on three fundamental performance metrics:

(1) Computational time: it corresponds to number of hash functions (k) for each bloom filter in a BFFDD. The online packet processing time has complexity of $O(k\omega)$ where ω is total number of edges in a BFFDD.

(2) BFFDD Space Requirement: it corresponds to bloom filter bitmap size (m). As a firewall have many BFFDDs for different customers, space is a major concern. The space complexity for a BFFDD is $O(m\omega)$.

(3) Probability of false positive: It corresponds to both m and k. Small false positive rate guarantees the correctness of firewall operation, and reduces the number of cases a firewall refers to the default decision.

To show the tradeoff between performance metrics we refer to Formula (1) that represents the probability of false positive in a bloom filter. Given a bloom filter, assuming all k hash functions are perfectly random, the probability of false positive for an element that is not in the set, also known as *false positive rate*, is:

$$f_{BF} = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-kn/m}\right)^k \quad (1)$$

By relaxing m and k, the global minimum for false positive rate is $(0.5)^k \approx (0.6185)^{(m/n)}$ where $k = (\ln 2) \cdot m/n$. Note that k must be an integer and less than the optimal value to avoid computational overhead.

B. Observations

In practice, access policies are designed such that a few services or subnets are protected from un-trusted subnets or hosts. Hence, we can intuitively claim that for a constructed FDD, the edge sets are either very small for the protected services and subnets or very large for the complementary edge sets in the field domain. Also, there are many nodes that have only one outgoing edge with an edge set equal to node's field domain. Therefore, the edge set sizes denoted by γ follows a bimodal distribution for a given FDD. To investigate this hypothesis in real-life firewalls, we examine 40 real-life firewalls and calculate the CDF function of their edge set sizes. The real-life firewalls range from very small businesses and home users with 6 number of rules to relatively large businesses with 7655 number of rules. Figure 3 shows the CDF function for IP and port fields. For all edge sets, the dotted lines in Figure 3(a) shows a bimodal distribution where 84.56% of IP fields outgoing edges have $\gamma \leq 65836$, and 15.19% IP fields outgoing edges have $\gamma \simeq 2^{32}$. Similarly, the dotted lines in Figure 3(b) shows 26.67% of port fields outgoing edges have $\gamma \leq 1025$, and 72.78% of port fields outgoing edges have $\gamma \simeq 65535$.

Motivated by this observation, we propose optimization techniques to remove redundant bloom filters and compress the rest of the bloom filters efficiently.

C. Removing Redundant Bloom Filters

As the domain size for IP fields is very large, depending on the hardware specifications and FDD size, offline BFFDD

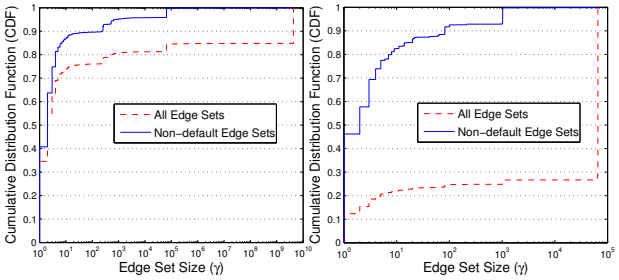


Fig. 3. CDF of edge set sizes on real-life FDDs

construction may take few days to be completed. This long construction time is problematic when a customer wants to immediately apply its incremental changes on its firewall access policies. Based on the bimodal distribution of γ, this is clear that most of the construction time is dedicated to build bloom filters for the edge sets in the second mode of γ (*i.e.* for IP fields, $\gamma \simeq 2^{32}$ and for port fields, $\gamma \simeq 65535$). To resolve this problem, for a given node, we single out the outgoing edge with maximum edge set size (γ) and we call it the *default edge*. Accordingly, the rest of the outgoing edges are called the *non-default edges*. Comparing solid lines that show the number of non-default edges with dotted lines that denote the number of all edges in Figure 3(a) and (b), we can infer that most of non-default edges are in first mode and default edges are in the second mode. The new BFFDD is constructed such that we only construct bloom filters for the edge sets of non-default edges, while the default edges do not have a bloom filter. Using this scheme, we almost exclude building bloom filters for edges in second mode from the BFFDD construction algorithm. This optimization is indeed required to exponentially reduce the offline construction time and BFFDD size in real-life BFFDDs.

Using default edges, the packet filtering algorithm is as follows. For a given BFFDD, we initialize a subtree that contains all decision paths for a given packet. To find all decision paths, we recursively traverse a BFFDD and add non-default edges whose bloom filter contains the packet header fields. However, because of bloom filter's false positives, we also add the default edges along with the non-default edges. The similar procedure is done in parallel for N independent but similar BFFDDs with different hash functions. If the terminal nodes of any of N subtrees share a single decision, this decision is returned as the final decision; otherwise, we need to consider all N BFFDDs to inquire for the packet decision. As BFFDDs have independent and different hash functions, we compare N subtrees together to find common decision paths. As we recursively traverse the subtrees, for all outgoing edges of a given node, if a non-default edge is repeated in all N subtrees it will be added to the result subtree and the node's default edge will be considered as faulty edge. Otherwise, the default edge will be added to the result subtree and all non-default edges are considered as faulty edges.

Figure 4 shows an example of query processing in a BFFDD. Figure 4(a) and (b) show the BFFDD construction with default edge for the example ACL shown in Figure 2(a). Given a packet with two fields where $F_1 = 38$ and $F_2 = 20$, we check it against two BFFDDs, $BFFDD_1$ and $BFFDD_2$. Note that in both BFFDDs, edges e_2, e_3 and e_5 are default edges and e_1 and e_4 are non-default edges. According to the FDD shown in Figure 2(b), the decision path for the given packet is $e_1 \rightarrow e_3 \rightarrow a$. Figure 4(a) shows the result subtree for $BFFDD_1$ where the first fields of packet is found in BF_{11} and also the second field of the packet is mistakenly found in BF_{41}. As default edges are always chosen on each level of the result subtree, the decision paths for the packet are $e_1 \rightarrow e_3 \rightarrow a$, $e_1 \rightarrow e_4 \rightarrow d$, and $e_2 \rightarrow e_5 \rightarrow d$. On the other hand, Figure 4(b) shows the result subtree for $BFFDD_2$ where the first fields of packet is found in BF_{12} and the second field of the packet is not found in BF_{42}. Similarly, the decision paths for the packet on $BFFDD_2$ are $e_1 \rightarrow e_3 \rightarrow a$ and $e_2 \rightarrow e_5 \rightarrow d$. Now, to detect which non-default edge has been chosen correctly, we look for the similar non-default edges in both result subtrees. As e_1 is included in both result subtrees, we add it to the final subtree and we discard the default edge in that level which is e_2. Accordingly, e_5 will also be discarded. On the next level, since e_4 is not found in the second result subtree, e_4 is considered as faulty edge and therefore the default edge e_3 is chosen to be added to final subtree. The final result is the decision path shown in Figure 4(c), $e_1 \rightarrow e_3 \rightarrow a$ which is the correct decision path.

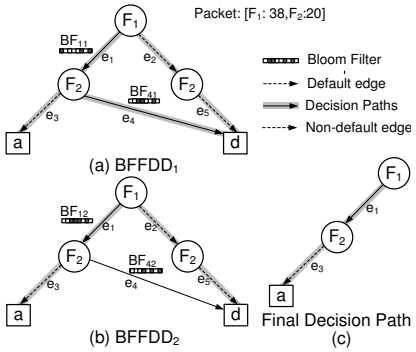


Fig. 4. Example query on BFFDD with default edges

D. Bloom Filters Compression

Considering Formula (1), to have very small false positive rate (*e.g.* $f_{BF} \leq 10^{-4}$), bloom filters may be designed for very large m values (*i.e.* may be very large in space). For instance, let $m = cn$ for a relatively small constant c. If we set $f_{BF} \leq 10^{-4}$, for small number of hash functions (*e.g.* $k \leq 3$), we need to choose $c \geq 64$. Accordingly, for each outgoing edge from IP field nodes (source and destination IP) with $n = 2^{32}$, for each outgoing edge from port field nodes (source and destination ports) with $n = 2^{16}$, and for each outgoing edge from protocol field with $n = 256$, we need bloom filters with size of $m = 256GB$, $m = 4MB$, and $m = 16KB$, respectively. However, as we may have hundreds of outgoing edges for each field, the amount of memory we need for bloom filters to represent all edge sets in an FDD is far beyond a reasonable space requirements. Therefore, we need to choose compression techniques with high compression rates to address this problem. On the other hand, to query bloom filters fast and efficiently, we need to choose those compression and decompression methods such that the compressed bloom filter can be queried directly or if it needs to be decompressed for querying, the complexity of decompression algorithm must be extremely low so that it can be queried fast and efficiently in terms of memory.

To choose the right compression methods, we look at the statistical features of the bloom filters. The solid lines in Figure 3(a) and (b) indicate that for IP fields, 9.71% of non-default edges have less than 65836 IP addresses (*i.e.* 0.0015% of IP addresses field domain) and for port fields, 99.7% of non-default edges have less than 1025 port numbers (*i.e.* 1.056% of port field domain). This clearly indicates that the bloom filters on non-default edges have very small set sizes and they are highly sparse. Hence, we use compression techniques suited for very sparse bit strings. There are many lossless compression techniques that are widely used in practice [19]. In this paper, we compare three compression methods namely, Hash Lists, Run-Length Encoding (RLE), and Golomb coding as follows:

Hash Lists. A straight-forward solution to store very sparse bloom filters is to keep the indices of 1s in a list. Let δ be the number of 1s in a bloom filter. Assuming $m \gg \gamma$ (using relatively good k random hash functions), we can estimate $\delta \simeq k\gamma$. Using a sorted hash list, the query time complexity for an element (using binary search) is $O(\log \delta)$. The required space for a compressed bloom filter using a hash list is $\log m \cdot \delta$. Note that all logarithms in this paper are in base 2, unless otherwise specified.

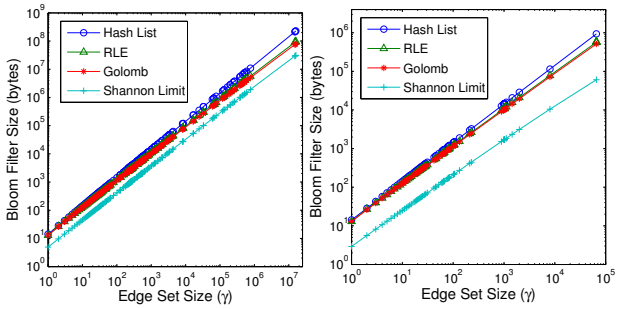
Run-Length Encoding. Run-Length Encoding (RLE) is one of the most popular dictionary based compressing methods that works efficiently on sparse bit strings. The basic idea behind RLE is to replace a long sequence of the same symbol by a shorter sequence so-called "run", which is usually the length of that sequence. RLE is very simple and its query time is significantly low compared to other compression methods.

Golomb Coding. Golomb Coding is an entropy-based data compression method which is highly effective on bit strings with redundancies [19]. Golomb code works based on a parameter M, referred to as the group size. Once M is determined, the runs of 0s in the bit string are mapped to groups of size M. An integer $x = Mq+r+1$ (which is the run length here) is represented by q 1s and a zero as delimiter (*i.e.* unary coding) along with r. We choose $M = \lceil \log_p(1+p) \rceil$, where p is the probability that 1 occurs ($p = \delta/m$). Note that it has been proven in [19] that golomb is the best prefix code if $M = \lceil \log_p(1+p) \rceil$.

Figure 5 compares the compression rate of above compression methods for IP and port fields for 40 real-life BFFDDs. The bloom filters are designed with $c = 64$ and $k = 3$. Figures 5(a) and (b) show that the bloom filter sizes increase linearly as the edge set sizes increase. The regression line for compression methods indicates that using RLE we can compress the hash lists with compression rates of 2.37 and 1.61 for IP and port fields, respectively. Using Golomb the compression rates for IP and port fields are 2.88 and 1.82, respectively.

Figure 6 compares the query time of above compression methods for IP and port fields for 40 real BFFDDs. Figure 6(a) and 6(b) show that the bloom filter query time for RLE and Golomb increase almost linearly as the edge set sizes increase. However, the query time is almost constant ($\simeq 10\mu s$) for sorted hash lists.

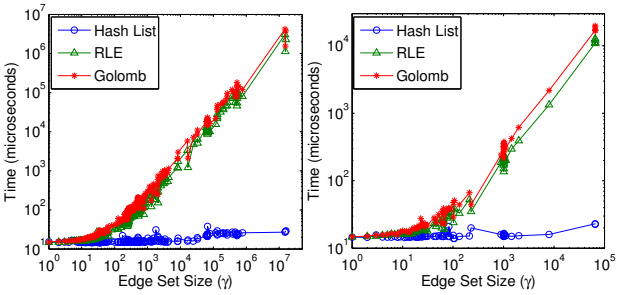
Other methods. There are other entropy-based methods such as Arithmetic Coding that may have better compression rate and be closer to Shannon limit; however, it can be very expensive in time and memory to query as it requires



(a) Bloom filter sizes for source and destination IP addresses

(b) Bloom filter sizes for source and destination port numbers

Fig. 5. Bloom filter sizes after compression



(a) Bloom filter query time for source and destination IP addresses

(b) Bloom filter query time for source and destination port numbers

Fig. 6. Bloom filter query time after compression

to decompress them. On the other hand, there are some well-known run-length encoding schemes like Word-Aligned Hybrid (WAH) [20] and (Byte-aligned Bitmap Code) BBC [21] that are used in bit index compression that has very small query time; however, their compression rate is not as good as entropy-based techniques.

E. Node Composition and Decomposition

To modify the size of bloom filters we can compose and decompose FDD nodes such that the semantics of a firewall remains intact. As mentioned, the size of a bloom filter on an outgoing edge is dependent on the node domain size and the edge set size.

Node Composition. Given two nodes v_1 and v_2, we can combine them to a new node v_{12} using a cross product function $\mathcal{F} : D(F(v_1)) \times D(F(v_2)) \rightarrow D(F(v_{12}))$ such that the semantics of the firewall remains intact.

Node composition can be used to improve the BFFDD privacy, since it increases a node domain size. This will be further explained in Section V. However, as cross product of the edge sets can result in very large edge sets, the BFFDD size may grow exponentially. Thus, node composition is selectively used for specific subtrees whose information is more critical for a user. This may include the subtree containing set of server IP addresses and their corresponding open ports that the user may care the most among all rules.

Node Decomposition. Given node v_1, we can decompose v_1 into two nodes v'_{11} and v'_{12} using an inverse cross product function $\mathcal{F}^{-1} : D(F(v_1)) \rightarrow D(F(v'_{11})) \times D(F(v'_{12}))$ such that the semantics of the firewall remains intact.

Node decomposition can be used to have smaller bloom filters to save memory. However, we need to ensure that in the final BFFDD, there is a node whose domain size satisfies the privacy requirements. For instance, to keep the

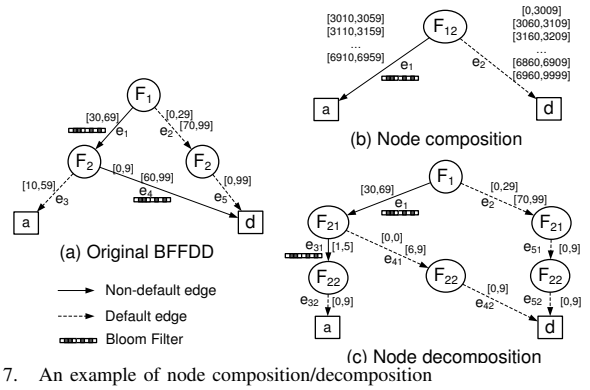


Fig. 7. An example of node composition/decomposition

complexity of rule deanonymization to $O(2^{32})$, we may decompose either source IP or destination IP node but not both of them. Using node decomposition for some BFFDDs is highly recommended. Because, as shown in Figures 5(a) and 6(a), if the size of edge set is very large (larger than one class B address) the bloom filter and its query time are relatively large. Although based on our observation the chance that an FDD has very large edge set is very trivial ($\simeq 0.3\%$), domain decomposition can be used on an edge to create smaller edge sets for better compression and query time.

These two functions clearly describe a loose trade-off between the privacy and the performance metrics including required memory and query time for a bloom filter. For a BFFDD, the intensity of this trade-off may be variable based on semantic of the rules. It also depends on nodes and their outgoing edge set sizes that are composed or decomposed. This is because the number of bloom filters as well as their corresponding edge set sizes are very important in performance metrics. This is further studied in Section VI.

Figure 7(a) shows an example 2-dimensional BFFDD. For node composition, we compose node F_1 and F_2 to node F_{12} (shown in Figure 7(b)). By node composition, the complexity of rule deanonymization is increased from $O(100)$ to $O(1000)$. Also, the bloom filters are reduced to one; however, the edge sets are modified from two edge sets with 40 and 50 elements to one edge set with 2000 elements. Figure 7(c) shows an example of node decomposition where F_2 is decomposed into two nodes F_{21} and F_{22}. By node composition, one edge set with 50 elements is decomposed to only one non-default edge with an edge set containing 5 elements. Note that in this example, the complexity of rules deanonymization is the same as the original BFFDD ($O(100)$).

F. Flow Caching

To increase the throughput of an outsourced firewall, we can use flow caching on the ISP side. That is, for each customer, when the first packet of a flow is queried from the customer BFFDD, the packet header with its corresponding decision is stored in the customer's packet cache table on the ISP side. Accordingly, for incoming and outgoing packets, if the packet headers are found in the customers cache table, the corresponding decision will be returned. Otherwise, it will be queried from the customers BFFDD. The cache table size and cache replacement policy may vary depending on the customer's expected number of concurrent flows, the percentage of discarded traffic, and the firewall specifications (mainly memory) on the ISP side.

G. Whitelisting

Even though using multiple versions of BFFDD significantly reduces the chance of misclassification using anonymized firewalls, we can use whitelisting technique to avoid any BFFDD misclassification. Using BFFDDs with basic design, the bloom filters false positives that lead to multiple decision paths with multiple decisions are detectable when we construct the BFFDD. The whitelist can be sent to the ISP as a hash table so that the whitelisted packet headers remains confidential. However, using default edges, the misclassification errors are categorized into two detectable and undetectable errors. The detectable errors occur if all N subtrees share a faulty edge, while a true positive edge is a non-default edge and multiple decision paths lead to different decisions.

On the other hand, the undetectable errors occur if all N subtrees share a faulty edge, while a true positive edge is the default edge and the final decision is different from the true packet classification decision. In this scheme, only detectable errors can be eliminated by whitelist. Note that we only discuss a single false positive on each node level because occurrence of several false positives on each node level is very unlikely to happen.

V. System Analysis

A. Misclassification in BFFDDs without Default Edges

Let f_{BF}^i be the false positive probability for a bloom filter on i-th level of an FDD. Having N BFFDDs, let FC be the event that the decision of a packet be falsely determined by the classifier. Let A be the event that an element be falsely found in a bloom filter with probability of f_{BF}^i on all N BFFDDs (*i.e.* the BFFDD query result is a subtree with multiple paths). Let B be the event that the result subtree leads to different decisions. Let C be the event that the default policy decision is different from actual decision for the packet. Thus, event FC occurs, if A, B, and C occurs. As A, B and C are independent, the probability of misclassification can be shown as follows:

$$P(FC) = P(A \cap B \cap C) = P(A)P(B)P(C) \quad (2)$$

Event A occurs if we have false positive on any level of the BFFDD for all N BFFDDs. Assuming that the bloom filters are independent and designed with same parameters for all N BFFDDs, the probability that a wrong edge be chosen is $(f_{BF}^i)^D$. Based on Boole's inequality, the upper-bound for $P(A)$ is as follows:

$$P(A) < \sum_{i=1}^{d} (f_{BF}^i)^N \quad (3)$$

In case we assume that the distribution of decision nodes in a BFFDD is uniform, $P(B) = 1 - 1/|DS|$. Moreover, based on the assumption it is clear that $P(C) = 1/|DS|$. Using formula 2, the upper-bound for probability of misclassification in a BFFDD is as follows:

$$P(FC) < \frac{1}{|DS|}(1 - \frac{1}{|DS|})(\sum_{i=1}^{d} (f_{BF}^i)^N) \quad (4)$$

Note that as $f_{BF}^i \ll 1$, we overlook cases in which there are more than one false positive on each node level. Using the recommended design parameters, $f_{BF} = 0.96 \times 10^{-4}$. This indicates using 3 BFFDDs ($N = 3$) with $|DS| = 2$, for each packet field $P(FC) < 8.84 \times 10^{-13}$. We also can conclude that the expected value of number of IP addresses to be queried with false positive is 0.0038 which is significantly less than 1. Similarly, the expected value of number of port and protocol

number are 5.8×10^{-8} and 2.26×10^{-10}, respectively. Let $E(FC)$ be the expected value of the number of packets that can be misclassified. Using Formula (4), we have,

$$E(FC) < \frac{1}{|DS|}(1 - \frac{1}{|DS|})(\sum_{i=1}^{d} |D(F(v_i))|(f_{BF}^i)^N) \quad (5)$$

Using our design parameters, the expected value of the number of packets that are misclassified is less than 0.0019 which is significantly less than 1.

B. Misclassification in BFFDDs with Default Edges

Using BFFDDs with default edge, the misclassification happens in two cases: (1) On a given node, if a packet header matches against two non-default edges where one is true positive and one is false positive (2) On a given node, if a packet header matches against only one non-default edge the true positive is the default edge. Let $A1$ denote the event for the first case and $A2$ denote the event for the second case. Let D be the event that the final decision is a wrong decision. Using Formula (2), the probability of misclassification for a BFFDD with default edges is $P(FC) = P(A1)P(B)P(C) + P(A2)P(D)$.

Let ρ_i be the probability that a packet header truly matches against a non-default edge on a node in i-th level of an FDD (*i.e.* $1 - \rho_i$ is the probability that a packet header truly matches against a default edge on a node of i-th level in an FDD). Hence, using formula (3) for nodes with more than three outgoing edges, we have, $P(A1) < \sum_{i=1}^{d} \rho_i (f_{BF}^i)^N$ and $P(A2) < \sum_{i=1}^{d} (1 - \rho_i)(f_{BF}^i)^N$.

Let $\Gamma(\epsilon_i)$ denotes the size of the edge set for default edge ϵ_i on a node in i-th level in an FDD. Thus, $\rho_i = \frac{1}{|D(F_i)| - \Gamma(\epsilon_i)}$. Assuming the decision nodes in a BFFDD is uniform, we can deduce $P(D) = 1/|DS|$. Therefore, the upper-bound for probability of misclassification on a BFFDD with default edge is as follows:

$$P(FC) < \frac{1}{|DS|}((1 - \frac{1}{|DS|})(\sum_{i=1}^{d} \rho_i (f_{BF}^i)^N) + \sum_{i=1}^{d} (1 - \rho_i)(f_{BF}^i)^N)$$

$$< \frac{1}{|DS|^2}(\sum_{i=1}^{d} \rho_i (f_{BF}^i)^N + |DS| \sum_{i=1}^{d} (f_{BF}^i)^N) \quad (6)$$

Using the recommended parameters ($N = 3$, $f_{BF} = 0.96 \times 10^{-4}$, $|DS| = 2$), and assuming $\rho_i \to 1$, we can calculate the upper-bound of the probability of misclassification as $P(FC) < 0.6642 \times 10^{-12}$. Using Formula (6), the expected value of the number of packets to be misclassified is as follows:

$$E(FC) < \frac{1}{|DS|^2}(\sum_{i=1}^{d} \rho_i |D(F(v_i))| (f_{BF}^i)^N + |DS| \sum_{i=1}^{d} |D(F(v_i))| (f_{BF}^i)^N) \quad (7)$$

Using our design parameters and basic assumptions in this section, the expected value of the number of packets that are misclassified is less than 0.0057 which is significantly less than 1.

C. Privacy Analysis

For a "bad" employee to deanonymize a business access policy represented by N BFFDDs, he first needs to access all of them. He then needs to brute force them by all possible packets. The complexity of finding all the firewall rules by this method is $O(2^{104})$ (as there are 2^{104} possible packet headers). But if he somehow knows the FDD tree structure and has access to bloom filters (depending on the implementation it can be possible), he needs to brute force each bloom filter individually. The complexity of deanonymizing a bloom filter on one edge whose parent node is node v is $O(|D(F(v))|)$. Using BFFDD tree structure, once he deanonymizes all the bloom filters, he can construct an ACL including non-overlapping rules representing the customer firewall rules. To increase the deanonymization complexity, we use selective node composition to secure sensitive network information and significantly impede firewall deanonymization. For instance, if we compose nodes for sensitive server IP addresses with nodes for their services port numbers, the complexity of rule disclosure is increased from $O(2^{32})$ to $O(2^{48})$.

VI. Experimental Results

A. Implementation and ACLs

We implemented Ladon in C++ and evaluated it using real-life ACLs. We focused the measurement on the space required to store the anonymized firewalls and firewall query time. We concluded that Ladon is indeed efficient enough to be used in practice for small and mid-size businesses which are interested to outsource their network firewalls to their ISP site.

We conduct experiments on 40 real-life ACLs gathered from small/mid-size businesses such as university departments and small companies. The size of ACLs varies from 6 rules to 7652 rules with 2 to 4 number of decisions. The largest prefix that we have in our ACLs is a class A address (*i.e.* netmask /8 or 255.0.0.0). Our experiments are run on a UNIX machine with an Intel(R) Xeon(R) Quad-cores CPU running at 2.66GHz and 16GB of RAM.

Note that in this section, all BFFDDs are constructed with default edges, because our testbed's resources were not enough for construction of BFFDD without default edge for all real-life firewalls.

B. Offline BFFDD Construction

As a design parameter, we can choose the order of the nodes in a BFFDD. In our scheme, as memory is a major concern, we calculate BFFDDs with all possible orders (5!=120 cases) and then for each firewall we choose the one with the minimum size. Note that this is only done first time to find the order of the smallest BFFDD. The results show that finding the best node order to have BFFDD with minimum size is very important. The median of maximum improvement ((max size)/(min size)) can go up to 16 for 40 real-life FDDs. Interestingly, for some FDDs (*e.g.* ACL 23), the maximum improvement goes up to 7400. This indicates the semantic of the firewall rules are very important in number of non-default edges and size of bloom filters. The average of BFFDD construction time is 1.1 minute. The average offline BFFDD construction time is indeed short enough for customers to apply their incremental changes within reasonable time.

C. Memory and Query Time Evaluation

To evaluate the BFFDDs performance metrics, we keep using our recommended design parameters where number of hash functions is three ($k = 3$) and bloom filters are 64 times bigger than the domain size ($c = 64$). Figure 8(a) shows the number of rules, number of default edges, and number of non-default edges for 40 real-life BFFDDs. This figure indicates that using default edges, we remove 35.17% of the bloom filters for a basic BFFDD. Figure 8(b) shows the BFFDD size compressed using hash list, RLE and Golomb. Note that in this figure, the firewalls are sorted based on number of edges in their FDD. The results clearly indicate that the size of 37 out of 40 firewalls are less than 1MB. The average compression

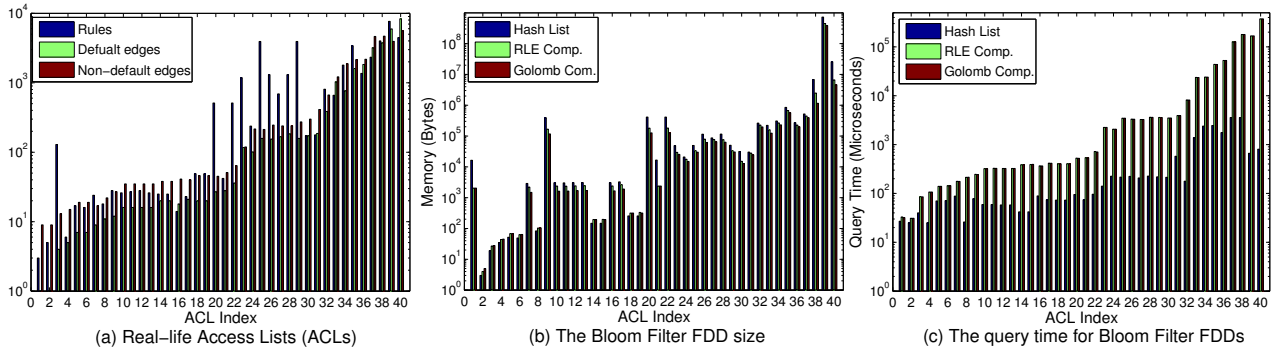


Fig. 8. Firewall outsourcing evaluation on 40 ACLs with their space requirements and query time

rate for RLE and Golomb (comparing to hash list) are 1.68 and 2.063, respectively. Figure 8(c) shows the query time for compressed BFFDDs. The results illustrate that all firewalls can be queried in less than 3.5 milliseconds using the hash list. However, the BFFDD query times after using RLE and Golomb, are respectively 29.023 and 28.998 times increased compared to hash list.

D. Firewall Throughput Evaluation

To evaluate the firewall throughput, we compare the throughput of an outsourced firewall with a firewall using sequential search packet classification algorithm which is commonly used in commercial firewalls. Sequential search is an exhaustive packet classification algorithm in which the firewall classifier searches the rules sequentially from the first rule to find the first match [22]. For the throughput evaluation, we use a sample of real-life packet traces from UMASS wireless access points and computer science network, which is publicly available on UMASS Packet Repository [23]. Our sample contains 3 million packets of 106,283 flows with total payload size of 1.14GB. To have a reasonable evaluation, for each ACL, we create a mapping table to change the most frequent packet IP classes to the ACL's source and destination network IP ranges. This mapping provides a correspondence between the packet traces and the ACLs for the throughput evaluation. Note that the firewalls has flow caching functionality where each packet decision is cached in the system for 30 seconds. Figure 9 (a) and (b) shows the throughput of BFFDDs and firewalls with sequential search in Mbits per second (Mbps) and packet per second (pps). The results show that in 80% of the ACLs, the total packet processing time for BFFDDs is 4.72 (on average) times more than the sequential search and in 20% of the ACLs, the packet processing time for the sequential search is 2.84 (on average) times more than BFFDDs. However, in 80% of the ACLs. the throughput of a firewall with sequential search is 2.08 times more than the throughput of an outsourced firewall, and in 20% of the ACLs, the throughput of an outsourced firewall is 2.56 times more than the throughput of a firewall with sequential search. The results also indicate that while the throughput of a firewall with sequential search varies between 24.840Mbps to 1.627Gbps (or 7,936pps to 5.2×10^5pps), the throughput of an outsourced firewall varies between 13.346Mbps to 1.143Gbps (or 4,264pps to 3.652×10^5pps) for different ACLs. Yet, as this throughput is relatively more than the conventional bandwidth for home and small businesses, the throughput is not a real concern for outsourced firewalls.

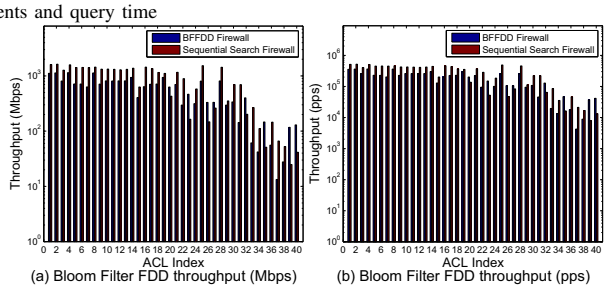
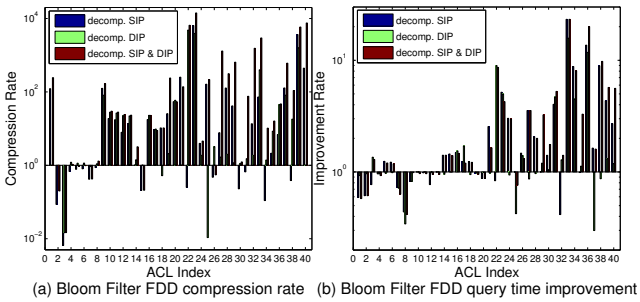


Fig. 9. Outsourced firewall throughput analysis

E. Node Composition and Decomposition Evaluation

To evaluate node composition, we selectively use some BFFDDs and we combine source IP and protocol fields together for partial FDDs in some pilot BFFDDs. The basic results show that the construction time as well as memory increase linearly as the number of elements in edge sets increases. However, as the number of elements in the edge sets increase exponentially, the BFFDD size and its construction time dramatically increase. Thus, in order to control BFFDD size and construction time, we need to first choose the partial FDD that we want to secure and then choose the FDD node ordering such that we minimize the size of the BFFDDs. We evaluate the BFFDD size using node composition by some example cases and the results indicate that node composition can increase the construction time and BFFDD size from order 2 to order of 10^4, based on the number of node compositions.

Using node decomposition, on the other hand, we expect significantly smaller BFFDDs, but more number of levels and edges. Figure 10(a) and (b) shows the improvement of BFFDD size and query time, respectively. We evaluate the query time and BFFDD size in three cases: when (1) source IP node, (2) destination IP node and (3) both source and destination IP node are decomposed into nodes with domain sizes of 2^{16}. Figure 10(a) indicates that using node decomposition (1), (2) and (3), we can improve the BFFDD size in 65%,85%, and 82.5% of the firewalls, respectively. The average compression rate is 294.22, 280.75, and 1074.2, respectively. Note that figure 10(a) only considers the BFFDD size using hash list. Figure 10(b) also shows the improvement in BFFDD query time in 65%,52.5%, and 67.5% of the firewalls, using cases (1), (2), and case (3), respectively. The average query time improvement is 2.77, 2.12, and 3.36, respectively. Note that in cases (1) and (2) the complexity of deanonymizing rules is equal to $O(2^{32})$, whereas in (3), since both IP nodes are decomposed, the complexity of deanonymizing rules is equal to $O(2^{16})$. Node decomposition is highly recommended for



(a) Bloom Filter FDD compression rate (b) Bloom Filter FDD query time improvement

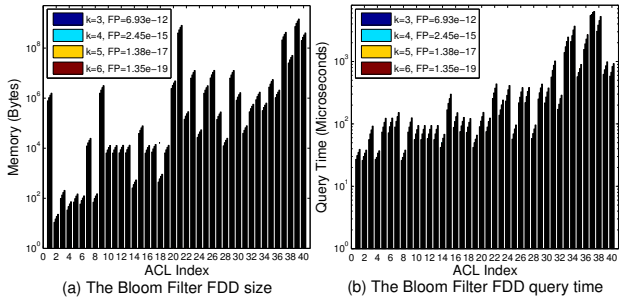
Fig. 10. BFFDD compression and query time improvement after node decomposition

firewalls with large edge sets. For instance, firewall 32 with size of 535MB is rebuilt by size of 159.3KB, 340.22KB, and 93KB for cases (1), (2) and (3), respectively. Node decomposition also reduces the construction time by order of 47.5, 3.37 and 23.15 for cases(1), (2) and (3), respectively.

F. False Positive Evaluation

Figure 11 shows the tradeoff between false positive rate versus BFFDD size and its query time. In this experiment, the bloom filter size is 64 times larger than the element set ($c = 64$); however, number of hash functions increases from $k = 4$ to $k = 6$.

The results in figure 11(a) indicate a constant growth in BFFDD size (using hash list) for all firewalls by order of 1.32, 1.67, and 2.01, for false positive rates 2.45×10^{-15}, 1.38×10^{-17}, and 1.35×10^{-19}. The growth in BFFDD size is because the number of 1s in bloom filters (δ) increases. Figure 11(b) also demonstrates roughly constant increase in BFFDD query time by order of 1.21, 1.43, and 1.65 for candidate false positive rates, respectively. This is also predictable, because more number of hash functions per bloom filter increases the query time linearly. Increasing false positive has slight impact on construction time. The construction time is increased by order of 1.11, 1.28, and 1.46, for candidate false positive rates, respectively.



(a) The Bloom Filter FDD size (b) The Bloom Filter FDD query time

Fig. 11. BFFDD size and query time with different false positive probability

VII. Conclusion

Privacy preserving outsourcing of firewall services benefits ISPs and customers significantly; however, this is a very difficult technical problem. This paper makes the first step towards this new direction. We introduced the Ladon framework that allows firewalls to be outsourced from customers to ISPs while preserving privacy. We presented detailed algorithms to build and query an anonymized version of the firewall using BFFDD. We defined the framework performance metrics containing required memory, query time, false positive rate, and privacy. We studied different methods for building a reasonable tradeoff between these metrics including compression methods, redundant bloom filter removal, and node composition/decomposition method. We showed that bloom filter compression is effective as bloom filters are highly sparse. We removed the edges in a BFFDD with maximum edge set for each node to decrease the construction time and anonymized firewall size. We suggested selective node composition to improve the privacy with some memory penalty. Besides, we proposed node decomposition to reduce BFFDD size and its query time. We finally implemented Ladon and evaluated it using 40 real-life firewalls. The results showed that Ladon is an effective framework for outsourcing firewall rules.

References

[1] T. Ritter, "Network-based firewall: Extending the firewall into the cloud," *http://www.business.att.com/content/whitepaper/Nemertes_DN0496_Network-Based_Firewall_Services_May_2009.pdf*, 2009.

[2] "AT&T-Network-Based Firewall," http://www.business.att.com/enterprise/Service/business-continuity-enterprise/eb_firewall_security/network-based-firewall-enterprise/.

[3] R. Richardson, "CSI/FBI computer crime and security survey," 2008.

[4] M. Whitworth, "Outsourced security the benefits and risks," *Network Security*, pp. 16–19, 2005.

[5] "Strategies for outsourcing security measures," *http://www.cisco.com/en/US/netsol/ns546/net_value_proposition09186a0080145ad7.html*, 2002.

[6] J. Grady, "The little guy fights back: Outsourcing firewall management makes economic sense for smaller companies - service providers," *Telecommunications. (http://findarticles.com/p/articles/mi_m0TLC/is_1_36/ai_83150961/)*, 2002.

[7] "Survey: 88% of ICT Employees Would Steal," *http://www.scoop.co.nz/stories/BU0811/S00203.htm*, 2008.

[8] R. B. McAfee and P. J. Champagne, *Effectively managing troublesome employees*. Quorum Books, 1994.

[9] M. G. Gouda and A. X. Liu, "Structured firewall design," *Computer Networks Journal (Elsevier)*, vol. 51, no. 4, pp. 1106–1120, March 2007.

[10] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of ACM*, vol. 13, no. 7, pp. 422–426, 1970. [Online]. Available: citeseer.nj.nec.com/article/chandranmenon95trading.html

[11] J. Cheng, H. Yang, S. H. Wong, and S. Lu, "Design and implementation of cross-domain cooperative firewall," in *Proc. IEEE Int. Conf. on Network Protocols (ICNP)*, 2007.

[12] A. X. Liu and F. Chen, "Collaborative enforcement of firewall policies in virtual private networks," in *Proc. PODC*, 2008.

[13] D. A. Maltz, J. Zhan, G. Xie, H. Zhang, G. Hjálmtýsson, A. Greenberg, and J. Rexford, "Structure preserving anonymization of router configuration data," in *Proc. IMC*, 2004.

[14] M. Harvan and J. Schönwälder, "Prefix- and lexicographical-order-preserving ip address anonymization," in *10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006.

[15] R. Ramaswamy and T. Wolf, "High-speed prefix-preserving ip address anonymization for passive measurement systems," *IEEE/ACM Transaction in Networking*, vol. 15, pp. 26–39, 2007.

[16] R. Pang, M. Allman, V. Paxson, and J. Lee, "The devil and packet trace anonymization," in *Proc. ACM SIGCOMM*, 2006.

[17] A. X. Liu and M. G. Gouda, "Diverse firewall design," in *Proc. Int. Conf. on Dependable Systems and Networks (DSN-04)*, June 2004, pp. 595–604.

[18] A. X. Liu, C. R. Meiners, and E. Torng, "TCAM razor: A systematic approach towards minimizing packet classifiers in TCAMs," *IEEE/ACM Transactions on Networking, to appear*.

[19] D. Salomon, *Data Compression: The Complete Reference*. Springer, 2006.

[20] A. S. K. Wu, E.J. Otoo, "An efficient compression scheme for bitmap indices," *Research Report, Lawrence Berkeley National Laboratory*, 2004.

[21] G. Antoshenkov, "Byte-aligned bitmap compression," *Data Compression Conference*, 1995.

[22] D. E. Taylor, "Survey & taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238–275, 2005.

[23] "Umass Trace Repository, http://traces.cs.umass.edu/."