Efficient Network Security Policy Enforcement With Policy Space Analysis

Xiang Wang, Weiqi Shi, Yang Xiang, and Jun Li, *Member, IEEE*

***Abstract*—Network operators rely on security services to protect their IT infrastructures. Different kinds of network security policies are defined globally and distributed among multiple security middleboxes deployed in networks. However, due to the complexity of security policy, it is inefficient to directly employ existing path-wise enforcement approaches. This paper models the enforcement of network security policy as the set-covering problem, and designs a computational-geometry-based policy space analysis (PSA) tool for set operations of security policy. Leveraging the PSA, this paper first investigates the topological characteristics of different types of policies. This heuristic information reveals intrinsic complexities of security policy and guides the design of our enforcement approach. Then the paper proposes a scope-wise policy enforcement algorithm that selects a modest number of enforcement network nodes to deploy multiple policy subsets in a greedy manner. This approach can be employed on network topologies of both datacenter and service provider. The efficiencies of the PSA tool and the enforcement algorithm are also evaluated. Compared with the header space analysis, the PSA achieves much better memory and time efficiencies on set operations of security policy. Additionally, the proposed enforcement algorithm is able to guarantee network security within a reasonable number of enforcement network nodes, without introducing many extra rules.**

***Index Terms*—Middlebox policy enforcement, policy space analysis, software-defined network.**

I. Introduction

A. Background and Motivation

NETWORK security middleboxes, such as firewall (FW), intrusion-detection system (IDS) and anti-virus gateway (AVG), are widely deployed in modern networks to deliver security services. However, due to the management complexity and the policy interaction among multiple devices, network operators face critical challenges when deploying or upgrading these middleboxes. On one hand, the number of middleboxes is comparable to the number of switches and routers in practical deployments [1], and middleboxes of the same type may be even from different vendors. These aspects induce high capital and operating expenses. On the other hand, security policies deployed on a single device or multiple devices overlap each other in most cases. The manual and local configuration of these policies often introduces intra-conflicts [2] and inter-conflicts [3] of rules, which makes policies unmanageable, increases system vulnerability, and deteriorates device performance.

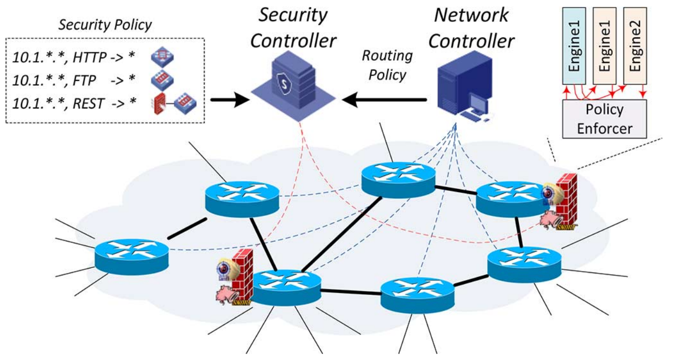

Fig. 1. Network security policy enforcement in software-defined network.

To tackle these challenges, network function virtualization (NFV) [4], combined with software-defined networking (SDN) [5], provides a promising solution. The L4–L7 security middleboxes are implemented on commodity physical servers [6], and can be activated at arbitrary network nodes. They cooperate with L2–L3 programmable switches [7] to deliver full-stack network services with flexibility and agility.

As most functions of SDN and NFV devices are largely orthogonal [8], these two types of devices are usually managed by different groups of operators. Meanwhile, the definition and enforcement of security policies must take routing decisions into consideration. Using Fig. 1 as an example, networking devices and security middleboxes are separately managed by their controllers. The security controller takes the routing policy determined by the network controller and the global security policy made by operators as its input, and decides policy enforcement points in the network. It must ensure the enforced security rules accommodate routing policy. Then, the relevant middleboxes are initiated to process security policies [9]. In this paper, the *routing policy* is deployed on *switches* and works at *L2–L3*. It specifies the traversing network paths for traffic at *packet*

Manuscript received December 06, 2014; revised August 18, 2015; accepted October 19, 2015; approved by IEEE/ACM Transactions on Networking Editor A. L. Narasimha Reddy. Date of publication December 22, 2015; date of current version October 13, 2016.

X. Wang was with the Department of Automation, Tsinghua University, Beijing 100084, China. He is now with China United Network Communications Group Co., Ltd., Beijing 100033, China (e-mail: xiang.wang.s@163.com).

W. Shi was with the Department of Electronic Engineering, Tsinghua University, Beijing 100084, China. He is now with the Department of Computer Science, Yale University, New Haven, CT 06511 USA (e-mail: weiqi.shi@yale.edu).

Y. Xiang was with the Research Institute of Information Technology, Tsinghua University, Beijing 100084, China. He is now with Yunshan Networks, Inc., Beijing 100083, China (e-mail: xiangyang@yunshan.net.cn).

J. Li is with the Research Institute of Information Technology, Tsinghua University, Beijing 100084, China, and also with the Tsinghua National Laboratory for Information Science and Technology, Beijing 100084, China (e-mail: junl@tsinghua.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNET.2015.2502402

level via *coarse-grained* rules, such as rules defined on destination IP. The *security policy* is installed on *middleboxes* and works at *L3–L7*. It specifies the security inspection for traffic at *flow level* via *fine-grained* rules, such as rules defined on classic 5-tuple, and does not involve traffic routing. Although OpenFlow-based SDN switches can match L4 packet header, most security processing must inspect packet payload in runtime to maintain application semantics, which limits the usage of these switches in the enforcement of security policies [9]. In this paper, a security policy includes multiple rules, and each rule consists of flow pattern and security action. The flow pattern indicates the values of packet-header fields, and the action may include permit/deny for access-control usage, different signature sets for deep-inspection scenarios, or service chains of multiple inspections. Packet-payload patterns of rules in deep-inspection systems [10] are separated from packet-header patterns and are viewed as the actions of security policies. Security policies are executed by the middlebox policy enforcer, which builds sessions at flow level and takes the responsibility for stateful inspections and load balancing among multiple deep-inspection engines [11], [12].

In this application scenario, it is an urgent requirement for security controllers to automate the security policy enforcement in large-scale networks, to agilely deliver on-demand security service in dynamic environment, i.e., cloud datacenter. Like the network controller exposing network service APIs to cloud orchestration tools, security services are also coordinated with other cloud services via APIs in a programming manner. Both *the selection of policy enforcement points* and *the computation of correct security rules* are essential processing, which plays a key role in the security controller.

B. Problem Statement

This paper addresses the problem of policy enforcement on middleboxes for the first time. And the solution is to find a set of enforcement network nodes for middleboxes to perform *on-datapath* inspections with globally defined security policies. Thus, the selected enforcement nodes should be first located on paths that are involved with security policies. This avoids the inefficient utilization of network bandwidth, which is produced by arbitrary traffic steering. Additionally, the solution needs to meet the following requirements:

Correctness: The algorithm should make sure that every rule in the security policy is processed at a certain network node, which guarantees the completeness of policy enforcement. Additionally, the security rules usually overlap each other, and the distributed inspection of multiple network nodes may cause rule conflicts. For instance, rules at different network nodes have different actions for the same traffic, while certain rules are enforced multiple times in the network. As a consequence, the enforcement algorithm needs to ensure the semantics of the global security policy.

Efficiency: The large number of different middleboxes increases the management complexity. Besides, more deployed devices require more wired connections, and recent studies on middlebox failures have indicated connectivity errors dominate failures across all middleboxes [13]. As a consequence, the algorithm should limit the number of policy enforcement points to a reasonable value. Moreover, it usually induces the inflation of rule number when dividing the global security policy into multiple subsets for distributed processing. More generated rules take more resources to perform rule lookup at enforcement points [14]. Therefore, the enforcement algorithm should minimize the overhead of extra rules. Furthermore, the design should take the running efficiency of algorithm into account to achieve low resource usage and high processing speed.

C. Proposed Approach

Considering these requirements, the enforcement problem of security policy on middleboxes is modeled as the set-covering problem (SCP) in this paper, which is a classic problem in combinatorial mathematics [15]. The SCP is defined as follows: *Given a set $\boldsymbol{U}$ of $\boldsymbol{n}$ elements and a collection $\boldsymbol{S}$ of subsets of $\boldsymbol{U}$, find the smallest collection $\boldsymbol{C}$ of $\boldsymbol{S}$ whose union is $\boldsymbol{U}$.* To model the policy enforcement problem, each rule in the $\boldsymbol{D}$-field security policy is regarded as a hyper-rectangle in the $\boldsymbol{D}$-dimension space. The space that the $\boldsymbol{l}^{\text{th}}$ rule covers is called the *rule space* and denoted as $\boldsymbol{S_{r_l}}$. The space that a security policy covers is called the *policy space* and denoted as $\boldsymbol{S_P}$. The space that the traffic covers at policy enforcement point $\boldsymbol{k}$ is called the *flow space* and denoted as $\boldsymbol{S_{f_k}}$. For problem modeling, $\boldsymbol{S_P}$ is mapped to $\boldsymbol{U}$, and $\boldsymbol{S_f}$ of all network nodes is mapped to $\boldsymbol{S}$. The algorithm should guarantee that the union $\boldsymbol{S_f}$ of all selected network nodes covers the global $\boldsymbol{S_P}$, while minimizing the number of policy enforcement points and maintaining the semantics of security policy. Let $\boldsymbol{K} = \{1, \ldots, k\}$, $\boldsymbol{L} = \{1, \ldots, \mathrm{l}\}$, $\boldsymbol{S_F} = \{\boldsymbol{S_{f_1}}, \ldots, \boldsymbol{S_{f_k}}\}$, $\boldsymbol{S_P} = \{\boldsymbol{S_{r_1}}, \ldots, \boldsymbol{S_{r_l}}\}$, the problem can be formally defined as follows:

$$\min \quad \sum\nolimits_{\boldsymbol{j}=1}^{\mathrm{k}} \boldsymbol{x_j}, \text{s.t.} \tag{1}$$

$$\forall \boldsymbol{i} : \bigcup\nolimits_{\boldsymbol{j}=1}^{\mathrm{k}} \boldsymbol{S_{r_{ij}}} = \boldsymbol{S_{r_i}}, \boldsymbol{i} \in \boldsymbol{L}, \boldsymbol{S_{r_{ij}}} \subseteq \boldsymbol{S_{r_i}} \cap \boldsymbol{S_{f_j}} \tag{2}$$

$$\forall \boldsymbol{j'}, \boldsymbol{j''}, \boldsymbol{j'} \neq \boldsymbol{j''} : \boldsymbol{S_{r_{ij'}}} \cap \boldsymbol{S_{r_{ij''}}} = \emptyset, \boldsymbol{j'} \in \boldsymbol{K}, \boldsymbol{j''} \in \boldsymbol{K} \tag{3}$$

$$\exists \boldsymbol{i}, \boldsymbol{S_{r_{ij}}} \neq \emptyset : \boldsymbol{x_j} = 1, \boldsymbol{i} \in \boldsymbol{L}, \boldsymbol{x_j} \in \{0, 1\} \tag{4}$$

Equations (2) and (3) guarantee the completeness of policy enforcement and the non-conflict of security rules, respectively. $\boldsymbol{S_{r_{ij}}}$ denotes the $\boldsymbol{S_r}$ that the $\boldsymbol{i}^{\text{th}}$ rule actually takes at policy enforcement point $\boldsymbol{j}$. In this model, there are two *key points* to be solved. The first one is the efficient expression and tool of flow/policy space for set operations, and the second one is the enforcement algorithm of low complexity for practical usage.

This paper first introduces the *policy space analysis* (PSA) tool. As many security rules have arbitrary range or wildcard values in multiple fields [16], each rule is expressed as a *range-based* hyper-rectangle, and the security policy is composed of multiple *non-overlapped* hyper-rectangles. This expression significantly reduces the computational complexity of set operations among multiple hyper-rectangles. Leveraging the PSA tool, this paper studies the topology/hierarchy of typical security policies, and the topological distribution of rules is statistically analyzed. The analysis result reveals the *scope* characteristics of security policies and illustrates the rationality of problem modeling.

Subsequently, this paper employs the greedy criterion to design the enforcement algorithm for practical usage. The procedure iteratively selects enforcement points until the union $\boldsymbol{S_f}$ of all selected points covers the global $\boldsymbol{S_P}$. Two heuristics can be adopted for point selection: the first one is flow-oriented and the second one is policy-oriented. The former selects the enforcement point that has the maximum flow space in the current iteration, while the latter chooses the one that intersects with more policy rules. We argue that both heuristics are reasonable and should be employed according to the applications in real deployment. Different rule densities over network flow space will lead to diverse resource and running efficiencies.

D. Key Contributions

Formulation of Middlebox Policy Enforcement: For the first time to our knowledge, this paper addresses the problem of middlebox policy enforcement. It also models the problem as the classic SCP, which lays a theoretical foundation for the final solution.

Policy Space Analysis Tool: Borrowing the idea of spatial indices in the database territory, the PSA tool is built on the basis of range-represented hyper-rectangles that are indexed by R-tree [17]. Two classes (i.e., *HyperRect* and *PolicySpace*) are abstracted. Both classes support three boolean operations (i.e., *is_equal*, *is_subset*, and *is_intersected*) and three set operations (i.e., *intersect*, *subtract*, and *union*). Most of these operations are implemented with the guidance of the concept of clipping region management from the computer graphics field.

Topological Analysis of Typical Policies: Three real policies [18] are analyzed by the PSA tool. For the first time to our knowledge, this paper illustrates the difference of these policies on topological properties, especially the enforcement point and the protected network topology, even if all this information is concealed for confidentiality [16]. Moreover, this paper shows the topological distribution of security rules. These figures depict the complexity of security policies enforced by switches and middleboxes and support the rationality of the proposed problem modeling.

Scope-wise Policy Enforcement Algorithm: Different from path-wise strategy where algorithms run over the object of network *path*, this paper designs algorithm that is executed over the object of network *node*. A certain node can involve multiple paths, accumulating a large *scope* of flow space, which helps with resource sharing. Based on this fact, the paper implements a greedy algorithm to guarantee both the efficiency and the correctness of policy enforcement. The experimental results on both datacenter and service-provider topologies show that the proposed algorithm can achieve the 93% reduction of enforced nodes, compared with path-wise algorithms.

II. Related Work

Existing works about security policy enforcement can be generally classified into two categories: the one is switch policy enforcement, and the other is middlebox policy enforcement. This section introduces these two types of work with respect to the processing tool and algorithm.

A. Header Space Analysis

The header space analysis (HSA) tool is proposed to statically debug network configurations [19]. With the HSA expression, multiple packet-header fields are abstracted to an *indiscriminate* bit-sequence. Each sequence can be regarded as an $\boldsymbol{L}$-dimension orthogonal space, where $\boldsymbol{L}$ is the bit number of the related header fields. Each packet is equivalent to a point in $\boldsymbol{L}$-dimension space, and each rule represents a hyper-rectangle. Four set operations (i.e., *union*, *intersection*, *complementation*, and *difference*) are provided for space processing. In addition, HSA also models networking devices with multiple transfer functions, whose inversed functions can also be calculated for expressing reversed processing. The HSA takes networking research a significant step beyond engineering via mathematical modeling. *However, there are three design points that impede more efficient manipulations for security policies*. The first point is the space representation in indiscriminate bits, which increases the computational complexity to a large extent. For example, to analyze the classic 5-tuple security policy, the number of HSA computing dimensions is 104, compared to 5 for the original policy. And this number becomes even larger along with the increments of field length in HSA expression. Additionally, security policies usually contain arbitrary range values, and the bit representation will encounter the inflation of range-prefix translation. The second point is the lack of non-overlapping guarantee for multiple bit-sequences in the same header space, and the results of most set operations of header space are delivered in an overlapping manner. For example, a 4-bit wildcard header space (*xxxx*) minus a 4-bit point (*1010*) is (*xxx1) union (xx0x) union (x1xx) union (0xxx)*. This overlapping of header spaces leads to more computing tasks when doing set operations, and some sub-spaces may be duplicated, which consumes more memory. Besides these upper two points, the HSA implementation does not contain an efficient indexing data structure. It employs the linear array to store bit-sequences, which makes the quick bypass of irrelevant operands of set operations almost impossible.

B. Policy Enforcement on Switches

The existing work of policy enforcement on switches falls into two groups: policy enforcement with changing routing, such as DIFANE [20] and vCRIB [21], and policy enforcement while respecting routing, such as, Palette [22] and One-Big-Switch [23]. In [23], the authors compare their differences in detail. The problem that this paper addresses is similar to the second group, and we argue that the decoupling of routing and security can bring more flexibility to system design [9]. Both Palette and One-Big-Switch employ *path-wise* strategies, where all network paths are first calculated from the routing decision. Palette makes packets of a certain path traverse the *entire* original policy when flowing through networks, and all paths comply with this design principle independently. However, not all rules in security policy are related to a certain path in real deployment, which hinders Palette from achieving optimal results. One-Big-Switch outperforms in this aspect, and makes advances in switch resource utilization when the shortest path length of network is small. However, it is difficult to directly employ these methods for middlebox policy enforcement. They aim to distribute the global policy evenly on *all* switches, which helps to take advantage of the ternary content addressable memory (TCAM) resource of network core switches. Thus, their motivations are mostly opposite to ours. Another reason is middlebox security policies usually contain a large number of wildcard rules. The path-wise enforcement strategy is prone to replicate these rules on multiple logically isolated traffic paths, which hampers the optimization of resource sharing.

C. Policy Optimization and Abnormality Detection

Existing researches about the security policy enforced by middleboxes are mostly related to firewalls. These studies fall into two main categories: policy optimization [24], [25] and policy abnormality detection [2], [3]. Gouda and Liu propose the firewall decision diagram (FDD) to optimize the firewall policy, which is able to guarantee the consistency, compactness, and completeness of policies. The FDD can also be employed to compare different firewall policies in order to ensure their explicitness. Al-Shaer *et al.* propose the policy tree to detect abnormalities (e.g., shadowing, spuriousness, redundancy, and correlation) in both standalone and distributed firewall policies. Overall, these studies implement *different data structures* to efficiently solve their *specific* problems. We argue that it is necessary to introduce a certain kind of *suitable mathematical tool*, which can lay a *foundation* for security policy analysis.

III. Policy Space Analysis

Policy space analysis is proposed to facilitate analyzing security policies. It is based on range-depicted hyper-rectangles of multiple dimensions. The PSA tool employs successful stories in both database and computer graphics territories (i.e., the spatial index and the clipping region management) to efficiently implement boolean and set operations of spaces. Leveraging the PSA tool, this paper explores the topological information of security policies. To the best of our knowledge, it is the first discussion of security policy in terms of topology.

A. Expression and Tool

1) HyperRect and PolicySpace: In general, each $\boldsymbol{D}$-field rule is viewed as a $\boldsymbol{D}$-dimension hyper-rectangle, which is abstracted to the *HyperRect* class. To build a generic analysis tool for security policies, the specific meanings of packet-header fields are omitted. Each dimension of *HyperRect* is an arbitrary range, which is qualified by *begin* and *end* points. The lower bounds of all dimension ranges are 0, and their upper bounds are not restricted. The hyper-rectangle represents both the location and the size of its related rule in the global space. The rationality of range design is derived from the following facts. A number of rules have arbitrary range values in certain fields, especially the transport-layer ports. In addition, some practical security policies may only protect a small range of hosts/servers deployed in a consecutive address prefix. This means that range values may also exist in network-address fields. To reduce the computational complexity, neither the prefix representation nor the independent bit-sequence is employed.

The *PolicySpace* class is a set of multiple *HyperRect*s. This abstraction is introduced to support operations among multiple hyper-rectangles or complex rules that may have multiple range values in every field [26]. For example, if a 2-field rule is defined in general ternary string where one wildcard exists at the central place of expression on both fields, the cross-product decomposes the rule into four hyper-rectangles (exactly four points) from the geometric view. Thus, the *PolicySpace* class can be leveraged for expression. Hyper-rectangles in a certain policy space do not overlap each other. This property is maintained to eliminate the influence of overlapping and rule priority. We argue that these two issues can be tackled with the boolean or set operations introduced in the following. To fast exclude unrelated operand hyper-rectangles, the PSA tool requires an efficient multi-dimensional spatial index data structure as the backend of *PolicySpace* class. Both the K-D tree [27] and the R-tree are competent for searching in multi-dimensional space. The variant of the former [14] has been attractive in packet classification, for its faster searching speed. However, it is difficult for K-D trees to perform balanced incremental updates, since the data structure needs to be totally rebuilt when adding or removing hyper-rectangles to keep the balance. Thus, the PSA employs the R-tree, as it can rebalance itself to adapt to the variations in data density.

2) Boolean and Set Operations: Both the *HyperRect* and the *PolicySpace* support three meta boolean operations (i.e., *is_equal*, *is_subset*, and *is_intersected*) and three meta set operations (i.e., *intersect*, *subtract*, and *union*). Other space manipulations can be implemented based on the combinations of these operations. The foundation of these meta operations is the clipping approach in the computer graphics field. This approach derives from the computational geometry but is simpler, as all hyper-rectangles are axis-aligned.

HyperRect: Most operations are intuitive. We only briefly comment on the operations of *subtract* and *union*. The *subtract* operation reveals the rule priority, which contributes to the rule de-overlapping. It makes sense to perform the subtraction of high-priority rules from low-priority rules, where the result is a policy space that low-priority rules *real* take effect. And the empty result suggests that low-priority rules are completely shadowed by high-priority rules. The *subtract* operation inspects all dimensions sequentially, and checks whether the current dimension needs split. Different inspecting sequences produce diverse operation results. As the design experience of packet classification algorithms indicates, policies usually show diverse characteristics on heuristic metrics. To take this fact into consideration, the implementation of all *HyperRect* operations, including the *subtract*ion, supports user-defined sequence via input parameters. Fig. 2 compares different results of the *subtract* operation, where $\boldsymbol{R}3$ subtracts both $\boldsymbol{R}1$ and $\boldsymbol{R}2$. The inspecting sequence of the top-right scenario is (x, y), and it generates seven hyper-rectangles. The inspecting sequence of the bottom-right scenario is (y, x), and it generates five hyper-rectangles. The *union* operation is usually employed for processing multiple overlapped rules of the same priority. It is implemented by leveraging a similar idea as the *subtract* operation. It first subtracts one hyper-rectangle from the other and then appends the subtracted hyper-rectangle to the result. Similarly, the exchange of operands usually produces different results. Fig. 3 illustrates two possible union results of two 2-dimensional hyper-rectangles. The top-right scenario includes $\boldsymbol{R}1_{\boldsymbol{2}}$ unions $\boldsymbol{R}1_{\boldsymbol{1}}$, and the

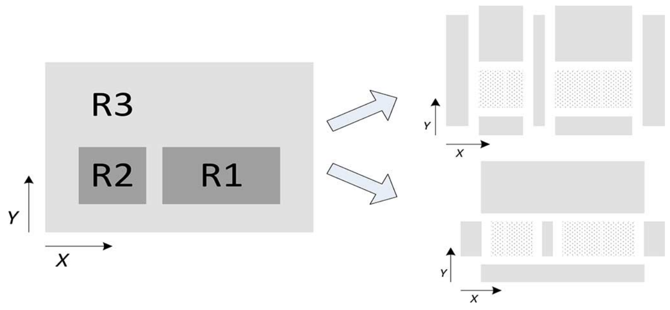

Fig. 2. *Subtract* operation among three 2-dimensional rectangles.

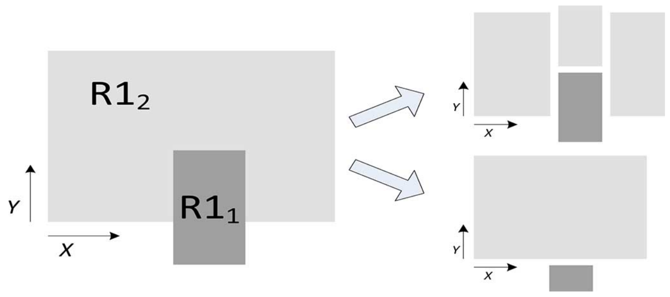

Fig. 3. *Union* operation between two 2-dimensional rectangles.

bottom-right scenario includes the opposite ones. As the *union* operation is sequence-irrelevant, the sequence that generates the minimum number of result hyper-rectangles is chosen.

PolicySpace: The *PolicySpace* instance employs an R-tree to fast index the hyper-rectangles that are intersected with the operand ones. Based on the indexing result, most operations of the *PolicySpace* are implemented as the iteration of the same ones of the *HyperRect*. Among these meta operations, neither *is_subset* nor *is_equal* can be implemented directly based on the same operations of the *HyperRect*. Different *PolicySpace* instances that cover the same area may contain diverse sets of multiple hyper-rectangles of different locations and sizes. The results shown in both Figs. 2 and 3 display the cases. Since the *is_equal* operation is equivalent to two *is_subset* operations, only the latter operation needs to be implemented. And there are two approaches. The first method is to directly employ the *subtract* operation of the *PolicySpace* class, and the empty subtraction result means the minuend policy space is equal or contained in the subtrahend one. The second method is to compare the volumes of the minuend policy space and the policy space that intersects with the subtrahend one. If both volumes are equal, this means that the minuend policy space is equal or contained in the subtrahend one. Since hyper-rectangles in a certain policy space are guaranteed not to be overlapped, it is easy to prove the correctness of this approach. This approach is much faster than the first one and thus is employed in real implementations.

B. Topological Analysis of Policy

Before designing the enforcement algorithm of a network security policy, it is suggested to first inspect different types of security policies from the perspective of topology. Different from the analysis of policy patterns in existing researches, the hierarchies of different security policy types are explored in this section, which can be employed to infer not only the topologies of protected networks but also the enforcement network nodes. Additionally, the rule distribution in policy space is depicted, taking a further step to reveal the topological characteristics of security policies.

Algorithm 1:

```
Layer_Segments(segs, reverse):
    metric ← seg_layer ← empty
    for s in segs:
        metric.append((s.len << 32) | s.begin)
    metric ← sort(metric, reverse)
    while metric:
        layer ← len(seg_layer)
        seg_layer.append(set())
        for m in metric:
            r.len ← m >> 32
            r.begin ← m & 0xFFFFFFFF
            r.end ← r.begin + r.len
            for s in seg_layer[layer]:
                if is_seg_overlap(r, s):
                    break
            else:
                seg_layer[layer].add(r)
                metric.remove(m)
    return seg_layer
```

Fig. 4. Length-first IP segement layering algorithm.

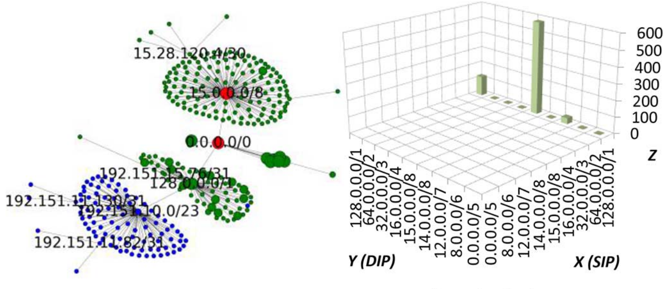

Fig. 5. ACL1 topology and rule distribution.

1) Policy Topology: The policy topology is drawn to explicitly figure out what network addresses the policy refers to and what traffic directions the policy inspects. All addresses that the security policy refers to are arranged in tree form. The procedure is the same as the layering step of the replication free grouping (RFG) algorithm [28]. Contrary to the RFG, the building procedure of policy topologies places large-range IP segments first and then small-range ones. None of the IP segments in the same layer overlap each other. Fig. 4 shows the length-first IP segment layering algorithm, which is optimized for IPv4 addresses. The unique difference between the layering step of the RFG and the building procedure of policy topologies is the value of the *reverse* input parameter, which is *false* for RFG and *true* in this case. The *metric* value is composed of the *length* and the *begin* point of segments. The *length* value taking most significant bits reflects its high priority during the layering. Each iteration of the *while* loop generates one layer. The left parts of Figs. 5–7 depict the topology of ACL1, FW1, and IPC1, respectively. These

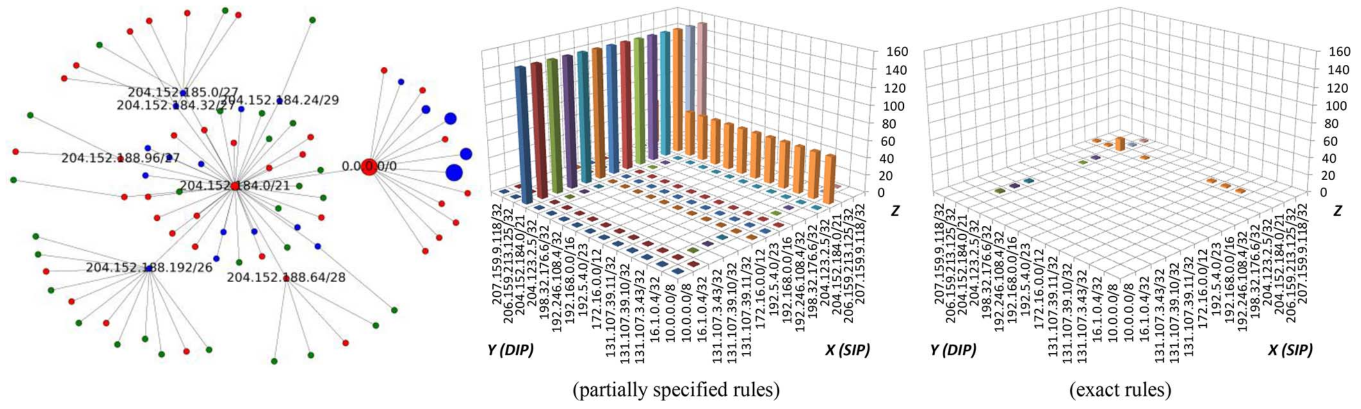

Fig. 6. FW1 topology and rule distribution.

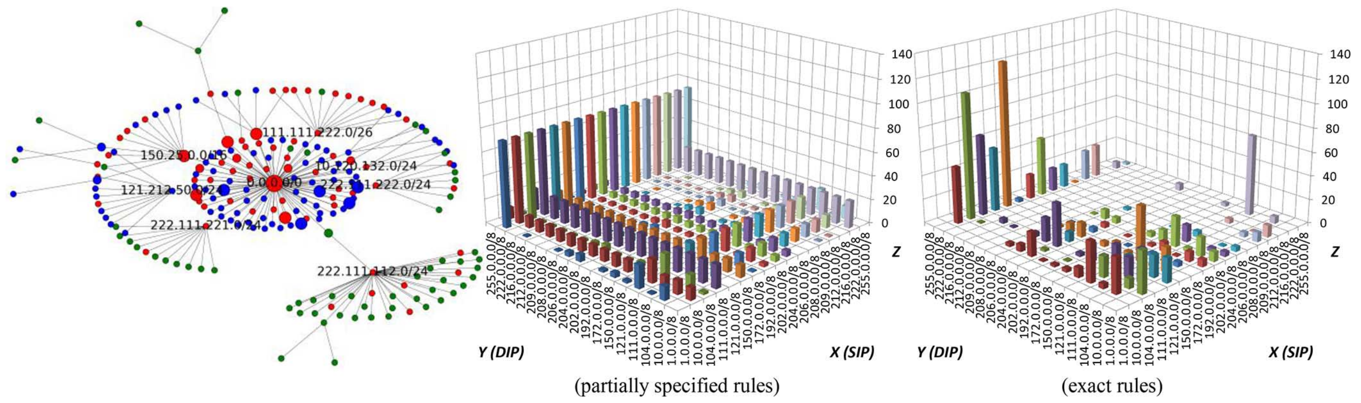

Fig. 7. IPC1 topology and rule distribution.

policies can be openly accessed at [18], which are packet classifiers in practical deployments. A node represents one IP segment referred by the policy. An edge will be added when IP segments in adjacent layers overlap. Nodes of only source IP are blue, nodes of only destination IP are green, and nodes that exist in both sides are red. The node size reflects the size of IP segments. To clearly show the referred addresses, only the nodes of top-eight degrees are labeled with their IP segments.

ACL1: In Fig. 5, the policy topology has a great number of small nodes. According to [16], the prefix lengths of these IP addresses are mostly 32. It can also be observed that these nodes are grouped into three clusters. The centers of the top-two dense clusters are 192.151.10.0/23 and 15.0.0.0/8. This suggests that this security policy is defined to focus on traffic inspection between these two IP segments, which will be proved in the following. Additionally, the directions of rules are nearly the same. The uniqueness of rule source IPs and the multiplicity of rule destination IPs imply that this security policy is probably enforced at the access point (router [16]) of 192.151.10.0/23.

FW1: In Fig. 6, the policy topology has fewer small nodes than in the ACL1 policy topology. According to [16], FW1 has a large portion of partially specified rules, and the prefix lengths of the specified fields are mostly 32. These nodes are around the topology center (i.e., the node of 204.152.184.0/21), but not the topology root (i.e., the node of 0.0.0.0/0). In addition, most labeled nodes are children of the center node, and the directions of rules are quite different from each other. This hierarchy implies that this security policy is defined to inspect traffic of hosts/subnets in 204.152.184.0/21. The uniqueness of the address cluster suggests that this security policy is possibly deployed at the gateway (firewall [16]) of 204.152.184.0/21.

IPC1: In Fig. 7, the topology root node has a great number of non-overlapped IP segments and is located at the center of the policy topology. These IP segments have different sizes, and most of them have no descendants. According to [16], rules of two exact IP addresses, rules of two prefix IP addresses, as well as rules of one exact and one prefix IP addresses account for a large proportion. Three labeled nodes are from 222.111.0.0/16 IP segment, and other labeled nodes are of discrete IP segments. The node color indicates that the directions of rules are mostly destined for 222.111.0.0/16. The diversity of IP segments and the distinct direction to a large specific IP segment suggests that this security policy is probably deployed near the core network, which may be an advanced gateway (firewall [16]) of multiple network interface cards.

2) Policy Topological Statistics: The policy topology delivers a sketchy global view for network operators to grasp at a glance. To further inspect the composition of security policies, it is necessary to analyze the topological distribution of rules in a statistical manner. The right parts of Figs. 5–7 show the information from ACL1, FW1, and IPC1, respectively. To clearly illustrate the result, the distribution of partially specified rules is shown separately. The rules of wildcards in both source IP and destination IP fields are omitted, as the number of them is extremely small. The $\mathbf{X}$ axis represents source IP, and the $\mathbf{Y}$ axis represents destination IP. The $\mathbf{Z}$ axis is the rule number.

The value ranges of both $\mathbf{X}$ and $\mathbf{Y}$ axes are same, and the units correspond to the child nodes of the topology root. In the distribution figure of the partially specified rules, the rule number is duplicated on all IP segments of the wildcard IP axis, to clarify the policy space that real rules cover. And the maximum rule number is selected when a certain space is covered by both row and column.

ACL1: There are 753 rules in this policy. The number of both wildcard rules and partially specified rules is 5, which only contributes 0.66% and is not shown in the figure. Except for these rules, all left rules fall into the source IP area related to 128.0.0.0/1 and 192.151.10.0/23 specifically, according to its policy topology. As speculated before, 579 rules are between 192.151.10.0/23 and 15.0.0.0/8 IP segments, which contribute 76.9%. The second contribution includes rules within the 128.0.0.0/1 IP segment. From its policy topology, these are rules between 192.151.10.0/23 and its adjacent IP segments.

FW1: There are 270 rules in this policy. The number of both wildcard rules is 17, which contributes 6.29%. And the number of partially specified rules is 213, which contributes 78.9%. Except these two types, the left rules are specified rules, whose contribution is 14.8%. Most rules fall into the cross-space of the 204.152.184.0/21 IP segment. As the large portion of partially specified rules indicates, this policy inspects both incoming and outgoing traffic of the gateway. And a few rules (i.e., 15 rules that contribute 6%) examine traffic within the internal network.

IPC1: There are 1550 rules in this policy. Because there are numerous discrete small IP segments around the topology root, these segments are first aggregated in accordance with the most significant byte of the IP address, which generates 17 larger IP segments. The number of both wildcard rules is only 5, which contributes 0.32%, and the number of specified rules is 1341, which contributes 86.5%. The remaining are partially specified rules, with a contribution of 13.2%. About 48.8% of rules (i.e., 756 rules) fall into the cross-space of the 222.0.0.0/8 IP segment.

From the above analysis of these three open real policies, it can be observed that security policies deployed on middleboxes are generally more complex than those deployed on switches or routers in the following aspects: 1) *the size of the involved policy space* and 2) *the overlap degree of security rules*. We argue that these differences mainly result from the different application scenarios in which these two types of policies are deployed. And it is the purpose and consideration made by policy managers that leads to these differences:

Since switches and routers are mainly employed for the large-volume forwarding and routing of traffic, these devices are widely deployed at both network core and network edge. From the view of entire network, the design of network policy should be fine-grained to precisely designate imposed objects, in which case their policy spaces are relatively small, to avoid impacting device primary functions. Additionally, the precision produces less rule overlapping, which is also advantageous for precise control in non-security application scenarios, such as quality of service (QoS).

On the contrary, security middleboxes, such as firewalls, are employed to protect servers or hosts from attack. These devices are usually installed at the choke point of subnets. Form the view of standalone network node, only a small number of flow patterns are legitimate and allowed by policies. Thus, operators must formulate policies of many wildcard rules, to effectively meet complicated security requirements. Furthermore, the flow pattern in security rules also reflects the relationship of people in organization to a certain extent, which is of large overlapping degree substantially. It makes security policies more complex than network policies in essence. Moreover, many security middleboxes integrate multiple security functions to perform chain-based security processing, which brings even more complexities to the definition of security policy.

IV. Network Security Policy Enforcement

For the *scope* characteristic depicted by the above analysis, the path-wise policy enforcement algorithms will substantially inhibit the sharing of policy lookup at certain nodes, in which case a large number of wildcard rules may be replicated. Thus, this paper models the problem as the SCP. After implementing the efficient tool for set operations of policy space, this section introduces the enforcement algorithm based on the PSA.

A. Set-Covering Algorithms

The SCP is a classic combinatorial optimization problem. The decision version of SCP is an NP-complete problem, while the optimization version is an NP-hard one. Many algorithms have been proposed, but not all of them are efficient for practical usage. The existing effective algorithms can be categorized into two groups [15]: the exact approaches and the heuristic ones. The former group solves the problem via linear programming, where the constraint is defined on the element of the global set $\boldsymbol{U}$. In the policy enforcement scenario, the elements are all exact points in the global policy space $\boldsymbol{S_P}$. Obviously, the element number of this type is too large to be efficiently calculated in practical usage. Even though the elements can be expressed as non-overlapped hyper-rectangles in policy spaces, the orders of magnitudes are still above 6 for small-scale security policies [14]. The latter group solves the problem in a greedy manner, which is selected as the design basis of the proposed middlebox policy enforcement (MBPE) algorithm. And there are *two main differences* between the greedy SCP algorithm and the MBPE algorithm: the first one is that the MBPE should tackle the conflict/overlapping of rules during the processing to guarantee the correctness of original policy semantics, and the second one is that the MBPE must work on aggregated elements (i.e., hyper-rectangles) to ensure the feasibility and efficiency of algorithms.

B. Policy Enforcement Algorithm

The greedy approach to the security policy enforcement runs iteratively. Each iteration of the procedure selects one network node whose flow space can cover most of the remaining policy space. The procedure does not stop until the space of the global security policy is fully covered. Fig. 8 shows the main frame of the algorithm. The procedure takes the rule-space list of the global security policy (***sr_list***), all network nodes (***nodes***), and their corresponding flow-space list (***sf_list***) as its inputs. The top-six lines initialize the remaining flow space (***sf_rem***) and the remaining policy space (***sp_rem***) via the unions of all elements

Algorithm 2:

```
00  Enforce_Policy(nodes, sf_list, sr_list):
01      sf_rem ← sp_rem ← empty
02      for sf in sf_list:
03          sf_rem ← sf_rem.union(sf)
04      for sr in sr_list:
05          sp_rem ← sp_rem.union(sr)
06      sp_rem ← sp_rem.intersect(sf_rem)
07      while sp_rem:
08          n ← Select_Node(nodes, sf_list, sr_list, sf_rem, sp_rem)
09          n_sf ← sf_list[n]
10          n_sf_rem ← n_sf.intersect(sf_rem)
11          n_bypass ← n_sf.subtract(sf_rem)
12          n_enforce ← empty
13          for sr in sr_list:
14              if sr.is_intersected(n_sf_rem):
15                  n_enforce.append(sr)
16          sf_rem ← sf_rem.subtract(n_sf_rem)
17          sp_rem ← sp_rem.subtract(n_sf_rem)
18          nodes.remove(n)
```

Fig. 8. Middlebox policy enforcement algorithm.

in their original lists, respectively. Then the remaining policy space is intersected with the remaining flow space to exclude the irrelevant policy space. At the following steps, the procedure decides the enforcement network node ($\boldsymbol{n}$) iteratively, according to different heuristics introduced in the following. The subtraction between the original flow space of the selected node (***n_sf***) and the remaining flow space is the spaces that should be bypassed (***n_bypass***). It is a necessity to calculate these bypassed spaces, as the semantics of security policy must be guaranteed. ***n_bypass*** means these flow spaces have been already processed at previous selected nodes, i.e., the relevant traffic will be inspected at these nodes and there is no need to examine the traffic any more. The original rules that should be processed at the currently selected node (***n_enforce***) are the ones that intersect with the remaining flow space of the currently selected node (***n_sf_rem***). Thus, the final rules that a certain network node processes are composed of the ***n_bypass*** with high priorities and the ***n_enforce*** with low priorities. Additionally, these rules can be further optimized by FDD [24]. After calculating the real enforced rules, both the remaining flow space and the remaining policy space are subtracted with the ***n_sf_rem***. The selected network node is removed from the ***nodes***, and the procedure continues to the next iteration. Taking the *HyperRect* operations as the criterion, the most complex processing of the proposed algorithm is at line13–line15 in Fig. 8. In the worst case, assuming the number of network nodes is $\boldsymbol{n}$, the number of *HyperRect* instances in ***n_sf_rem*** is $\boldsymbol{O}(\boldsymbol{n}^2)$, which means that the full mesh of point-to-point traffic traverses through the network. Therefore, the time complexity of line13–line15 is $\boldsymbol{O}(\boldsymbol{r}n^2)$, where $\boldsymbol{r}$ is the rule number of security policies. Since the outermost *while* loop of the MBPE algorithm procedure may examine all network nodes at worst, the theoretical time complexity is $\boldsymbol{O}(\boldsymbol{r}n^3)$.

At the step of node selection, there are two types of metrics that can be employed for making decisions: the number of intersected rules and the number of hyper-rectangles in flow spaces. Accordingly, there are two heuristics to select network nodes: one is to select the node whose $\boldsymbol{S_f}$ intersects with the maximum number of policy rules, the other is to select the node whose $\boldsymbol{S_f}$ has the maximum number of hyper-rectangles. Since different security policies may have very diverse topological characteristics, both heuristics have their rationalities. The former heuristic helps the procedure to efficiently locate the network nodes that probably cover a large proportion of the remaining policy space, especially on fine-grained policies. But the policies that have many wildcards may lead the flow spaces of most network nodes to intersect with these wildcard rules, which significantly reduce the algorithm efficiency. The latter heuristic can relieve the degradation of algorithm efficiency for wildcard policies. It helps the procedure to fast converge to the terminal condition, where the flow space plays a pivotal role. As a consequence, we argue that the choice of heuristics depends on policy characteristics in real deployment.

C. *Enforcement Update*

The enforced policies may require updating in the following cases: with a change of the flow space and with a change of the policy space. The former case may happen when virtual hosts migrate or link statuses change; the latter case is probably due to policy reconfigurations conducted by network operators.

With a change in the flow space, the updated network nodes and their flow spaces can be calculated by NetPlumber [29] in real time. The update procedure takes this information as its input and recalculates both the bypassed flow space and the intersected rules of the updated network nodes in the selected sequence of the first time. And the newly involved enforcement nodes are temporarily appended to the sequence for the current update. The main reason of maintaining the original selected sequence is the following observation: On one hand, the foremost selected nodes are mostly core network nodes, as they usually intersect with the majority of the global security policy. On the other hand, most changes in the flow space happen at network accesses without modifying the network backbone. Therefore, the original selected sequence needs to be held to avoid the degradation of enforcement efficiency.

With a change in the policy space, the network operator may add/update/delete policy rules. To efficiently locate the network nodes that are deployed with the updating or deleting rules, a mapping data structure between the rule and the set of its enforced nodes is maintained. When updating or deleting an existing rule, the update procedure can directly conduct the same enforcement executions on involved nodes in the selected sequence of the first time. If a new rule needs to be added in the global policy, the procedure checks the overlap between this rule and the flow spaces of enforced network nodes in the original selected sequence. And the rule is enforced at its first intersected network node. If no node is found, the enforcement executions continue to be performed on un-selected nodes.

V. Evaluation

In this section, the performance of the PSA tool and the proposed policy enforcement algorithm is evaluated. First, the experimental environment is described. Then both memory and time efficiencies of the PSA tool are evaluated. Finally, the effectiveness of the proposed algorithm with different heuristics of node selection is compared on both service-provider network and datacenter topologies.

A. Experimental Environment

Both the PSA tool and the policy enforcement algorithm are implemented in Python. The experiments are conducted on a single Intel i7-4770 3.4 GHz CPU workstation with 32 GB DDRIII memory and Fedora19 64-bit operating system. The service-provider network topologies are from [30], and the datacenter topologies are generated as 3-tier tree topologies. The HSA tool employed for evaluation is implemented in the C language. In addition to the three real policies employed in the above analysis, other policies for evaluation are generated by ClassBench [16] using its default arguments. The policies are attached to network topologies by employing a similar manner in [23]: First, the Algorithm 1 with the *reverse* value of *false* is executed to get non-overlapped IP segments of small range. Then, compared with the number of network access nodes, these IP segments may be aggregated if the number of them exceeds the one of access nodes. Otherwise, large-range IP segments are disassembled into multiple small-range ones. Last, these segments are attached to network access nodes, which mean the traffic originated from these nodes is labeled with IP in the corresponding segment as its source address. Moreover, full mesh traffic among all access nodes is assumed to traverse through networks.

B. Policy Space Analysis

An application is designed to evaluate the memory and time efficiency of the PSA, and the comparison between PSA and HSA is also conducted using this application. The application is to calculate the space that each rule in real policies takes. Due to the rule overlapping in security policies, the low-priority rules are likely to be partially shadowed by high-priority ones. The spaces that the low-priority rule real takes effect are the subtraction result between the original space of the low-priority rule and the spaces of all rules with higher priorities. It can be observed that this application, as well as the enforcement algorithm, heavily relies on the *subtract* operation of the *PolicySpace* class. Thus, the evaluation result can also reflect the memory and time efficiency of the enforcement algorithm. Furthermore, another application is designed to evaluate the design decision of non-overlapping guarantee in *PolicySpace*. The application is to judge whether two complex *PolicySpace* instances are equal. The instances employed for evaluation are the calculation result on the default rule with two different checking sequences of dimension in the upper application.

Fig. 9 compares the memory efficiency of HSA and PSA. It shows the growth curve of the element number along with the increments of the calculated rule number. The element of the header space is the bit-sequence, and the element of the policy space is the hyper-rectangle. The 100-scale policies of different types are employed for evaluation. It can be observed that the HSA fails to continue processing when encountering the rules of many wildcards. It exhausts all 32 GB physical memory of the workstation, as its *subtraction* may generate 104 overlapped bit-sequences for the worst-case scenario. In contrast, the PSA works well on all policies. And for the worst-case scenario of its *subtract* operation, it will only generate $2 \times \boldsymbol{D}$ ($\boldsymbol{D} = 5$ in this application) hyper-rectangles. Due to the need for mathematical semantics, the literal operations of the HSA, as well as the optimization of "lazy subtraction", cannot be employed in this application and the policy enforcement scenario.

The memory efficiencies of different checking sequences of the policy dimension are compared in Fig. 10. The metric for sequence decision is the number of non-overlapped segments shadowed on every dimension. The normal sequence is from the dimension of the fewest segments to the dimension of the most segments (protocol or port field is examined first), and the reversed sequence is employed for comparison (IP fields are examined first). The horizontal axis is normalized according to the rule numbers of different security policies. The curves show the increments of the element number over the increments of the policy percentage. Some parts of the curves which are parallel

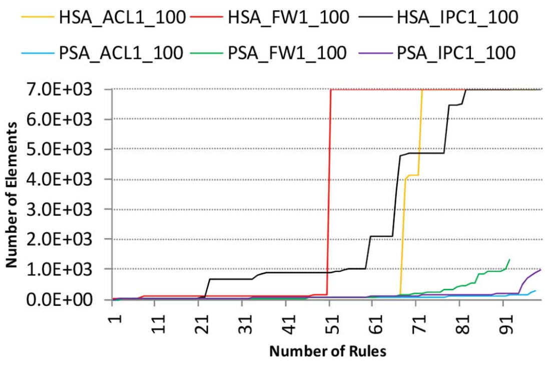

Fig. 9. Memory efficiency (HSA & PSA).

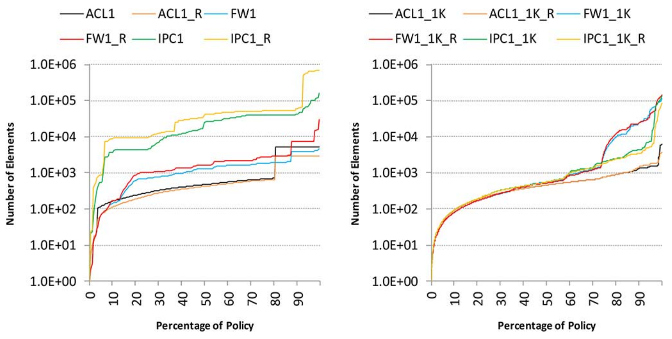

Fig. 10. Memory efficiency of different sequences (real & synthetic).

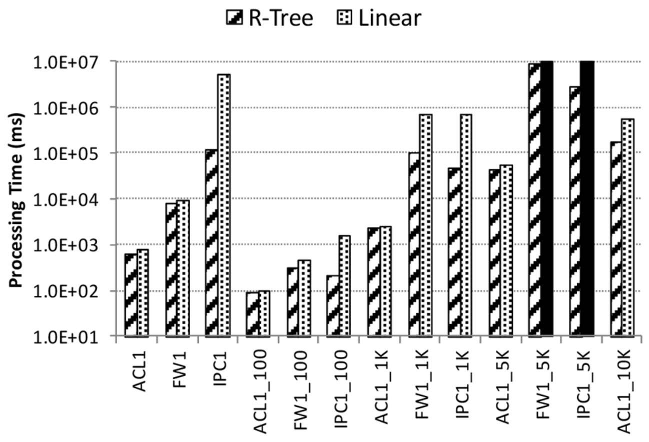

Fig. 11. Time efficiency of different index.

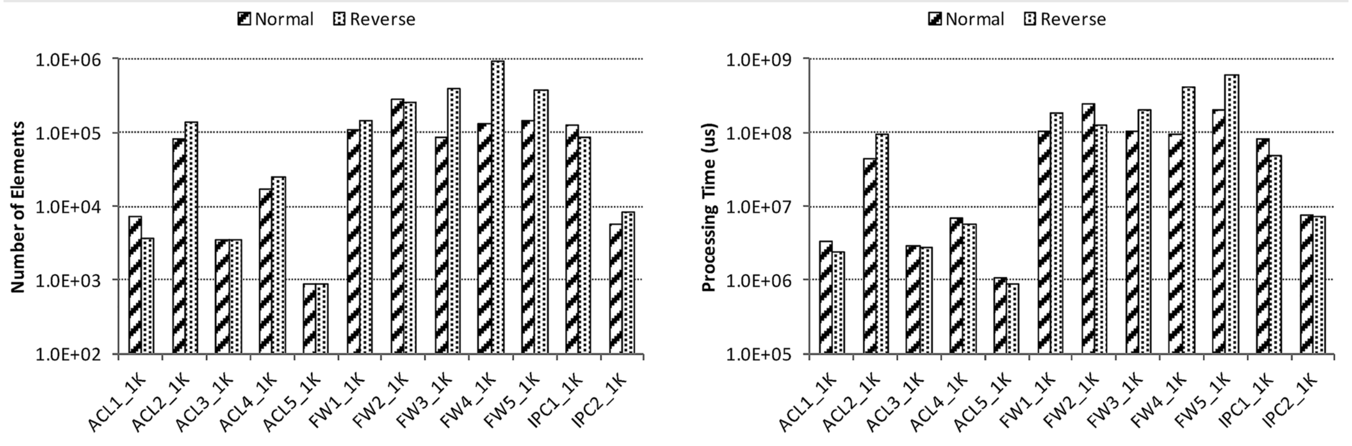

Fig. 12. Memory and time efficiency of different sequences (synthetic policies generated by all ClassBench seed files).

to the horizontal axis show the existence of shadowed/redundant rules. The left part of the figure is the result on real policies, and the right part of the figure is the result on synthetic policies. For real policies, it is observed that the maximum gap of memory efficiency is close to one order of magnitude. Additionally, for both the FW and the IPC policies, the efficiency of the normal sequence is better than the one of the reversed sequence, but an opposite result is achieved on the ACL policy. For synthetic policies, the efficiency gap between two checking sequences is not distinct. We argue that it is mainly due to the implementation of ClassBench did not take the topological characteristics of policies or the correlation of rule fields into consideration. The randomization of all fields seems to be indiscriminative and irrelative, which leads to less distinction of comparison results. Fig. 12 shows both the memory efficiency and the time efficiency on synthetic policies generated from all types of ClassBench seed files. It can be observed that the inherent characteristics of policies impact on the memory usage and the processing time to a large extent, even if these policies are of the same scale. In general, FW and IPC policies consume more memory and longer processing time than ACL policies, implying the former types of policies have more overlapping of rules. Furthermore, the difference between results of two checking sequences is also small in most cases. The big gap appearing on several FW policies is mainly caused by last few rules that have wildcards on multiple fields, leading to the explosion of memory usage and processing time.

Fig. 11 depicts the processing time of different policies. For real policies, the processing can be completed within two minutes. For synthetic policies, the large-scale complex policy requires more calculation time, as the ClassBench generator introduces the stochastic disturbance to forcibly increase the policy complexity. But in real deployments, the policy system [31] is usually designed to reduce the rule overlapping and complexity. Thus, the processing time may get shorter on real policies that are of the same scale. Even if the policy has high complexity in certain scenario, some compute-intensive works can be carried out offline in advance, to preserve the online performance. As a comparison, the processing time without R-tree index is also depicted. On small-scale or low-complexity policies, the index optimization is not quite obvious, compared to the linear storage.

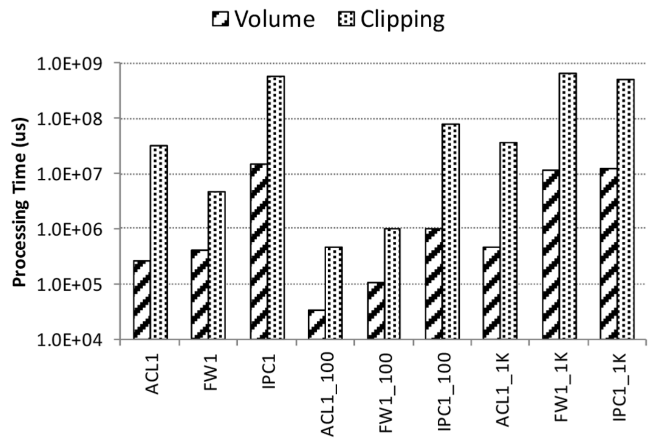

Fig. 13. Non-overlapping optimization.

Along with the increment of rule number, the time performance of PSA with R-tree support can achieve an improvement of 1–2 orders of magnitude. The PSA implementation with linear storage cannot even accomplish the processing within 2 hours on both FW1_5K and IPC1_5K.

Fig. 13 illustrates the optimization of non-overlapping in *PolicySpace* implementation. Since the hyper-rectangles in the same *PolicySpace* instance are not overlapped with each other, the volume comparison between intersected policy space and original policy space is 1–2 orders of magnitude faster than the generic clipping approach on all evaluated policies.

C. Policy Enforcement Algorithm

In order to evaluate the efficiency of the policy enforcement algorithm, the metrics of both enforced nodes and enforced rules are employed for measurement. Two types of policies are generated by ClassBench with different complexities: one of more exact rules and the other of more wildcard rules. The rule numbers of these generated policies are all about 25 K, except for AS1775, which has about 5 K rules. Additionally, the two different heuristics that are employed during the step of node selection are compared for both policy types.

Topology Properties: Tables I and II describe the properties of the service-provider networks and the datacenter networks,

TABLE I
PROPERTIES OF SERVICE-PROVIDER NETWORK

Topology	Core	Access	Max. Hop	Avg. Hop	Avg. Degree
AS4755	14	27	6	2.98	3.32
AS1755	161	139	13	5.59	3.65
AS3257	266	240	16	6.09	2.96
AS1239	370	234	11	4.18	7.5
AS7018	295	336	10	5.04	6.59
AS6461	272	382	11	4.87	4.07

TABLE II
PROPERTIES OF DATACENTER NETWORK

Topology	Access Port	Aggregation Port	Fabric Port
DC10-20-10	10	20	10
DC20-20-10	20	20	10
DC20-40-10	20	40	10

respectively. Different topologies have diverse scales and characteristics. For service-provider networks, Table I lists the numbers of both core nodes and access nodes, the maximum and the average number of all-pair shortest path lengths, and the average degree of network nodes. Table II shows the numbers of access switch port, aggregation switch port, and fabric switch port. And the total number of all access ports is the product of these three numbers.

Enforced Nodes: Fig. 14 shows the number of enforced nodes. The left part is for the enforcement of exact policies, and the right part is for the enforcement of wildcard policies. Each part compares the result of the proposed algorithm that employs two different strategies of node selection against the result of the per-source approach that exists in [20] and [21]. It can be observed that the proposed algorithm generates fewer enforced nodes than the per-source approach. For the enforcement of wildcard policies, the node number can be reduced up to 77%. For the exact policies, the reduction can achieve 93% at most. In addition, the enforcement of exact policies generally needs fewer nodes than the enforcement of wildcard policies, which is about 25%–44%. This is because the wildcard policies usually cover multiple address spaces, and the wildcard rules are replicated at multiple enforcement nodes. Furthermore, for the enforcement of exact policies, the policy-oriented strategy can achieve better results, while the flow-oriented strategy is more suitable for the enforcement of wildcard policies. As the policy space of many wildcard rules is intersected with the flow space of multiple network nodes, the policy-oriented strategy hinders the step of node selection of the algorithm from making the most progress toward the termination condition. At each step, the node that intersects with more policy rules may not cover a large flow space, which will induce more iterations. In this case, the flow-oriented strategy can outperform.

Enforced Rules: Fig. 15 shows the overhead of enforced rules, which is expressed as the ratio between the numbers of all enforced rules and the original policy rules. The results are also depicted in the former style that Fig. 14 employs. It can be observed that the proposed algorithm generates fewer extra rules than the per-source approach. These extra rules consist of two types: the replica of the wildcard rules and the bypassing

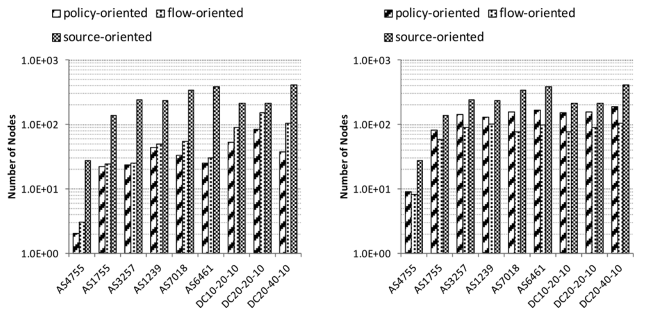

Fig. 14. Node number of the enforcements of exact and wildcard policies.

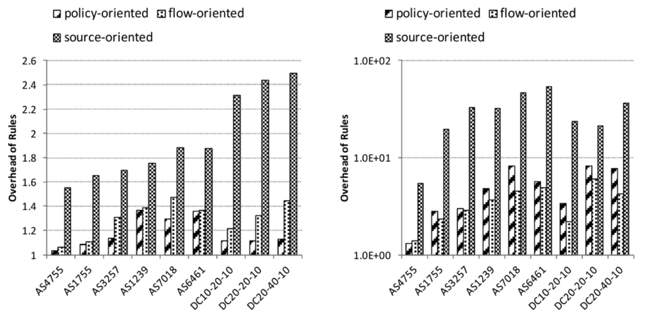

Fig. 15. Rule overhead of the enforcements of exact and wildcard policies.

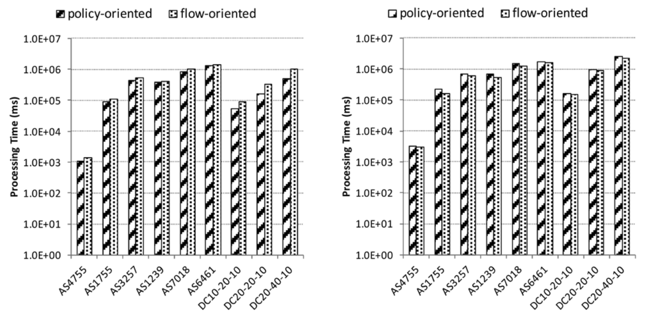

Fig. 16. Processing time of the enforcements of exact and wildcard policies.

rules that are introduced to tackle the rule overlapping. For the enforcement of exact policies, the maximum overhead is about 1.47. For the enforcement of wildcard policies, the overhead is much higher, but still within an order of magnitude. In contrast, the per-source approach generates 1.5–2.5 times more rules when enforcing exact policies, and the overhead becomes larger when enforcing wildcard policies, which is about 20–50 times.

Processing Time: The processing time of the enforcement algorithm is shown in Fig. 16. Under the above experimental conditions, the proposed algorithm has comparative processing times for both strategies of the node selection in a certain policy enforcement scenario. For the enforcement of exact policies on datacenter topologies, the flow-oriented strategy requires about twice the processing time of the policy-oriented strategy to finish. In general, for small networks, the algorithm can finish around $5s$. For medium networks, the enforcement is accomplished between $5m - 15m$. And for large networks, the processing time is around $30m$. Note that the processing time evaluated in this paper is actually the initialization time when security controllers performing a cold start. Complying with the incremental update procedure described in Section IV, security controllers can accomplish the adjustment within seconds that is much shorter than the initialization time. Nonetheless, the speed of PSA processing can certainly be improved in future work.

VI. Discussion

In this section, we discuss the relationship between the HSA and the PSA, and also the implication of the PSA for packet classification. Since the PSA analyzes policies from the point of view of the multi-dimensional space, we only focus on those algorithms based on space cutting/decomposition.

HSA is introduced for modeling networking devices, while PSA is more suitable for policy manipulation. Derived from network L2/L3 devices, such as switches and routers, HSA abstracts the behaviors of these devices as the transformations of multi-dimensional spaces. To support emerging protocols and arbitrary packet-header formats, the space is expressed as an indiscriminate bit-sequence. As the policies of these devices are usually defined in prefix, this expression can cooperate well with these policies. For network L4–L7 devices (i.e., network middleboxes), their policies are generally more complex than networking policies and usually contain plenty of heuristic information for facilitating manipulation. Thus, the bit-sequence expression may impede the exploitation of this information and lead to inefficient policy processing. Since these two types of devices are largely orthogonal [8], the PSA can acquire routing decisions from the HSA to provision multiple network services, where each tool offers its own advantages.

Packet classification has been studied for a couple of decades, and a great many solutions have been proposed. Among these existing algorithms, the ones based on space decompositions perform well in practice. But their behaviors show unsteadiness when these algorithms are applied on different policies. For example, the HyperCuts [32] algorithm achieves faster searching speed than HiCuts [33] because of its multiple cuts at each tree node, but both algorithms cannot be successfully executed on large-scale complex policies. Another example is the grouping methodology [34]. The grouping of policy rules can significantly reduce the pre-processing time and memory usage. In [28], it is observed that these improvements only appear on FW1 and IPC1 policy types. These two examples are related. As the PSA result shows, most rules in ACL1 are not overlapped. Thus, the large fan-out of both HiCuts and HyperCuts contributes to the separation of rules, and there is no need to group policy rules. But for FW1 and IPC1, the separation efficiency is reduced. We argue that the reason is the existence of *two types* of partially specified rules. In addition, their numbers are *quite comparable*. Either source IP or destination IP provides the high separability. Therefore, no matter which dimension the procedure selects, partially specified rules of the other type are largely duplicated. According to this feature, the separation of these two types of partially specified rules is enough, which has been testified by most recent research [35].

VII. Conclusion and Future Work

This paper addressed the middlebox policy enforcement for the first time to our knowledge. To efficiently deploy the global network security policy on multiple middleboxes in networks, this paper first implemented the PSA tool built on the expression of range-based hyper-rectangles. Leveraging the PSA, this paper inspected security policies in terms of topology, and exploited their scope characteristic. Then the paper modeled the middlebox policy enforcement as the SCP, and designed a greedy algorithm on the basis of the PSA. Compared to existing methodologies, the proposed algorithm achieved the 93% reduction of enforced nodes and the 91% reduction of enforced rules.

The Python version of the PSA tool helped to fast verify our superficial idea, and we will improve its processing speed by implementing it with the C language. The most recent technology of vector operations [36] can accomplish the comparison of two multi-dimensional hyper-rectangles via single instruction, which will boost the processing speed further. Additionally, it is worth reimplementing the detection algorithms of policy abnormalities, to test and evaluate the PSA tool. The other aspect of future work will include the advancement to packet classification research. Many packet classification algorithms have been proposed from different perspectives, but no single approach can adapt to all policy types. To help with their practical application, the pre-analysis of policy characteristics and the automatic algorithm selection can bring benefits to this research topic.

References

[1] J. Sherry *et al.*, "Making middleboxes someone else's problem: Network processing as a cloud service," in *Proc. ACM SIGCOMM*, 2012, pp. 13–24.

[2] E. Al-Shaer and H. Hamed, "Firewall policy advisor for anomaly discovery and rule editing," in *Proc. IFIP/IEEE IM*, 2003, pp. 17–30.

[3] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *IEEE J. Sel. Areas Commun.*, vol. 23, no. 10, pp. 2069–2084, Oct. 2006.

[4] ETSI, "Network functions virtualisation," 2014 [Online]. Available: http://www.etsi.org/technologies-clusters/technologies/nfv

[5] ONF, "Software-defined networking: The new norm for networks," 2012 [Online]. Available: https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf

[6] Palo Alto Networks, Santa Clara, CA, USA, "Palo Alto virtualized firewalls," 2014 [Online]. Available: http://www.paloaltonetworks.com/products/platforms/virtualized-firewalls/vm-series/overview.html

[7] VMWare, Palo Alto, CA, USA, "VMware NSX," 2014 [Online]. Available: http://www.vmware.com/products/nsx/

[8] J. McCauley, A. Panda, M. Casado, T. Koponen, and S. Shenker, "Extending SDN to large-scale networks," Open Networking Summit, Research Track, Apr. 2013, pp. 1–2.

[9] X. Wang, Z. Liu, B. Yang, Y. Qi, and J. Li, "Tualatin: Towards network security service provision in cloud datacenters," in *Proc. IEEE ICCCN*, 2014, pp. 473–480.

[10] Cisco Systems, San Jose, CA, USA, "Snort intrusion prevention system," 2014 [Online]. Available: http://www.snort.org

[11] V. Sekar, N. Egi, S. Ratnasamy, M. Reiter, and G. Shi, "Design and implementation of a consolidated middlebox architecture," in *Proc. USENIX NSDI*, 2012, pp. 323–336.

[12] Y. Qi *et al.*, "OpenGate: Towards an open network services gateway," *Comput. Commun.*, vol. 34, no. 2, pp. 200–208, Feb. 2010.

[13] R. Potharaju and N. Jain, "Demystifying the dark side of the middle: A field study of middlebox failures in datacenters," in *Proc. ACM IMC*, 2013, pp. 9–22.

[14] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: From theory to practice," in *Proc. IEEE INFOCOM*, 2009, pp. 648–656.

[15] A. Caprara, P. Toth, and M. Fischetti, "Algorithms for the set covering problem," *Ann. Oper. Res.*, vol. 98, no. 1–4, pp. 353–371, Dec. 2000.

[16] D. E. Taylor and J. S. Turner, "ClassBench: A packet classification benchmark," Washington University in Saint Louis, St. Louis, MO, USA, Tech. Rep. WUCSE-2004-28, May 2004.

[17] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. ACM SIGMOD*, 1984, pp. 47–57.

[18] H. Song, "Evaluation of packet classification algorithms," Last accessed 2014 [Online]. Available: http://www.arl.wustl.edu/~hs1/PClassEval.html

[19] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: Static checking for networks," in *Proc. USENIX NSDI*, 2012, pp. 113–126.

[20] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with DIFANE," in *Proc. ACM SIGCOMM*, 2010, pp. 351–362.

[21] M. Moshref, M. Yu, A. Sharma, and R. Govindan, "Scalable rule management for data centers," in *Proc. USENIX NSDI*, 2013, pp. 157–170.

[22] Y. Kanizo, D. Hay, and I. Keslassy, "Palette: Distributing tables in software-defined networks," in *Proc. IEEE INFOCOM*, 2013, pp. 545–549.

[23] N. Kang, Z. Liu, J. Rexford, and D. Walker, "Optimizing the 'one big switch' abstraction in software-defined networks," in *Proc. ACM CoNEXT*, 2013, pp. 13–24.

[24] M. G. Gouda and A. X. Liu, "Firewall design: Consistency, completeness and compactness," in *Proc. IEEE ICDCS*, 2004, pp. 320–327.

[25] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Trans. Parallel Distrib. Syst.*, vol. 19, no. 9, pp. 1237–1251, Sep. 2004.

[26] VMWare, Palo Alto, CA, USA, "VMware vCloud networking and security," 2014.

[27] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *ACM Commun.*, vol. 18, no. 9, pp. 509–517, Sep. 1975.

[28] X. Wang, C. Chen, and J. Li, "Replication free rule grouping for packet classification," in *Proc. ACM SIGCOMM*, 2013, pp. 539–540.

[29] P. Kazemanian, M. Chang, H. Zheng, G. Varghese, and N. McKeown, "Real time network policy checking using header space analysis," in *Proc. USENIX NSDI*, 2013, pp. 99–112.

[30] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, Feb. 2002.

[31] Cisco Systems, San Jose, CA, USA, "Cisco application policy infrastructure controller data center policy model," 2014 [Online]. Available: http://www.cisco.com/c/en/us/solutions/collateral/data-center-virtualization/application-centric-infrastructure/white-paper-c11-731310.html

[32] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. ACM SIGCOMM*, 2003, pp. 213–224.

[33] P. Gupta and N. McKeown, "Classifying packets with hierarchical intelligent cuttings," *IEEE Micro*, vol. 20, no. 1, pp. 34–41, Jan. 2000.

[34] B. Vamanan, G. Voskuilen, and T. Vijaykumar, "EffiCuts: Optimizing packet classification for memory and throughput," in *Proc. ACM SIGCOMM*, 2010, pp. 207–218.

[35] W. Li and X. Li, "HybridCuts: A scheme combining decomposition and cutting for packet classification," in *Proc. IEEE HOTI*, 2013, pp. 41–48.

[36] Intel, Santa Clara, CA, USA, "Intel Advanced Vector Extensions," 2014 [Online]. Available: https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions

Xiang Wang received the B.S. degree from the School of Telecommunication Engineering, Xidian University, Xi'an, Shaanxi, China, in 2007, the M.S. degree from the School of Software Engineering, University of Science and Technology of China, Hefei, Anhui, China, in 2010, and the Ph.D. degree from the Department of Automation, Tsinghua University, Beijing, China, in 2015.

He is currently a Researcher in the Cloud Computing Laboratory, China Unicom Research Institute. His research interests include software-defined networking, network functions virtualization, distributed system, and performance issues in computer networking and system architectures.

Weiqi Shi received the B.S. degree from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2015.

He is currently a Ph.D. student in the Department of Computer Science, Yale University, New Haven, Connecticut, USA. His research interests include software-defined networking, network architecture, distributed system and language programming.

Yang Xiang received the B.S. degree from the College of Software, Jilin University, Changchun, Jilin, China, in 2008, and the Ph.D. degree from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2013. Before July 2015, he was a Postdoctor in the Network Security Laboratory, Research Institute of Information Technology, Tsinghua University.

He is currently an R&D Engineer in Yunshan Networks Inc., Beijing, China. His research interests include software-defined networking, network architecture, data center interconnection, and intrusion detection.

Jun Li received the B.S. and M.S. degrees from the Department of Automation, Tsinghua University, Beijing, China, in 1985 and 1988 respectively, and the Ph.D. degree from the Department of Computer Science, New Jersey Institute of Technology, in 1997.

He is currently Dean of the Research Institute of Information Technology, Tsinghua University, Beijing, China. He is also Executive Deputy Director of the Tsinghua National Laboratory for Information Science and Technology, Beijing, China. His research interests include networking and network security, pattern recognition and image processing.

Dr. Li has been a member of the IEEE since 1996.