A Traffic-aware Top-N Firewall Approximation Algorithm

Ho-Yu Lam, Donghan Wang, H. Jonathan Chao
Department of Electrical and Computer Engineering
Polytechnic Institute of New York University, Brooklyn, NY

***Abstract*—Packet classification is widely used in various network security and operation applications. Two of the main challenges are the increasing number of classification rules, amount of traffic and network line speed. In this paper, we investigate an approximation algorithm for selecting the top-N most frequently matched subset of rules from the original ruleset. The goal is to obtain Top-N rules that covers as much traffic as possible while preserving the dependency relationships. Through simulations, we show that our approaches the optimal while runs in seconds, allowing online adaptation to changing traffic patterns.**

I. Introduction

As one of the critical network security components, firewall is deployed in virtually all operational networks. A firewall is typically deployed at strategic points of the network such that it will inspect most if not all of the traffic. It is crucial to maintain high classification throughput.

There is extensive research on enhancing the performance of individual firewall [1] and ruleset optimization [2]–[7]. However, in addition to the performance limitation of individual firewalls, the common practice of perimeter-based deployment is starting to affect the efficiency and scalability of emerging data center networks.

In a data center network, Firewalls are often deployed between the layer 3 (e.g. Internet Protocol) core network and the layer 2 (e.g. Ethernet) access layer. Virtual Local Area Network (VLAN) are widely used to partition the network different security domains such that traffic between domains traverses the Firewalls. While this works well on enforcing security policies, it is inefficient as server-to-server traffic traverse a much longer path than necessary consuming network bandwidth. The communication also has higher latency.

Proposals for next generation enterprise networks and data center networks [8]–[11] advocate distributed enforcement of security policies. Observing that in a data center network, virtually all switches are managed switches that are capable of doing packet classifications very efficiently for a limited number rules, one can make use of these capability to realize distributed sub-rulesets enforcement (caching) without the need of massively deploying full featured firewalls. A few catch-all firewalls can be used for misses to ensure all the rules are covered.

A prerequisite for a scalable and cost-efficient distributed enforcement is the availability of an efficient and correct sub-ruleset selection algorithm. Here we would like to highlight two observations: (i) Some rules in a ruleset are matched to a larger portion of traffic (i.e. higher hit-rate) than the others [12], [13]; (ii) The rules that are most relevant to a particular network location in the network are related to the traffic pattern, which changes from time to time and differs from location to location. In this paper, we explore the idea of dynamically selecting a subset of N rules (Top-N) with the highest hit-rates based on traffic pattern.

Such a selection algorithm has the following requirements: **R1**: Accepts rulesets with dependencies; **R2**: Suitable for online computation; **R3**: Imposes only light burden on traffic monitoring; and **R4**: Dynamically adapts to traffic changes. Our main goal in this paper is to develop an online algorithm for obtaining a Top-N subset of rules that satisfies the above requirements. Optimally dynamic reordering of rules with dependencies have been shown to be NP-Complete [6], [12]. Section II reviews related work in the literature. Section III elaborates on the background of the problem and a formal definition. Section IV details our algorithm and Section V presents simulation results. We conclude our paper in Section VI.

II. Related work

There is a rich literature in packet classification algorithms and data structures such as HiCut, HyperCut, Shunting, etc. to name a few [1], [7], [14]–[17]. While they greatly advance the classification speed of a firewall, it is generally true that a smaller (sub-)ruleset is desirable. It lowers storage requirement and memory footprint of a data structure, allows high-speed on-chip memory to be used, reduces the size and power consumption of some hardware components such as TCAM, etc.

Another direction of research aims at optimizing ruleset for smaller sizes [2]–[5]. While these optimization can be very useful for individual firewalls, they do not allow efficient sub-ruleset selection.

Traffic-aware firewall optimization and reordering [6], [7], [12] all face the challenge of dependencies among rules. A common tactic in solving the dependencies is to pre-process the originally ruleset to obtain a totally disjoint ruleset (one that has no rule overlapping with each other in their matching space) and then merge the rules after the proposed optimization. While this approach satisfies **R1**, the resulting disjoint ruleset will be so big that it will either burden traffic monitoring by requiring hit-counts on a large number of rules (fails **R3**) or it can only partially satisfy **R4** by

978-1-4244-9920-5/11/$26.00 ©2011 IEEE

adapting to historically measured traffic patterns through off-line computation (fails **R2**).

Yan et al. [18] proposed to optimize rule distribution among distributed firewalls. They first distribute rules to distributed firewalls and then optimize the rules in those firewalls. Their aim is to reduce the highest normalized workload among all firewalls. However, the algorithm is not suitable for online computation (fails **R2** and **R4**).

Fu and Zhang [19] presented an online adaptive firewall allocation scheme that dynamically load-balances firewall workloads among a farm of firewalls. The number of firewalls is dynamically adjusted according to the current load. While this is an online algorithm, the firewalls are centralized in a firewall farm location with all firewall configured with an identical set of global rules.

III. Problem Overview

A firewall rule specifies a matching space in the 5-tuple of source and destination IP addresses, source and destination ports and the protocol and is associated with an action. Three of the simplest actions commonly used in firewall are `Allow`, `Deny` and `Drop`.

The set of rules (ruleset) used in a firewall is specified as a sequence of rules. The matching spaces of rules can overlap with each other. When different actions exist among overlapping rules, there are conflicts among the rules. Rules earlier in the sequence have precedence over the later rules.

To provide traffic-aware optimization one will need to reorder the rules according to their *hit-rates*. However, reordering rules that conflict with each other changes the action assigned to packets that fall within the overlapping matching space. When a lower priority rule r_2 (partially) conflicts with a higher priority rule r_1, we say that r_2 *depends* on r_1. Any reordering shall only results in a ruleset that are equivalent to the original one by resolving the dependencies of reordered rules.

The idea is to optimize for rules with higher hit-rates so that an overall higher packet classification throughput can be achieved. A top-N selection is to select the N rules that maximize the total hit-rate achieved by the resulting sub-ruleset. As an example, consider an OpenFlow-like deployment where the limited packet classification capability in switches is used as caches of the whole ruleset and a controller is used as a catch-all classifier. A top-N sub-ruleset allows operators to have a small number of top-N rules to be classified at managed switches while minimizing the miss-traffic converge at the controller.

When only a subset of rules (sub-ruleset) is extracted for packet classification, it is possible that a packet that would be a match in the original ruleset may get no match from the sub-ruleset. However, it is incorrect for a packet to get an action from the sub-ruleset that is different from one that it would get from the original ruleset. Given a ruleset, we would like to obtain a sub-ruleset of N rules by modifying some of the rules to resolve their conflicts.

The key concepts, strategy and process of the proposed Top-N selection are presented next followed by the formal problem statement in Section III-E.

A. Top-N target list

By sorting the hit-rate table in descending order of hit-rates, the first N rules are the *target rules* and together we call them the *target set*. Since some of the target rules may depend on rules that are not in the target set, one cannot simply select the N target rules as the top-N rules.

B. Dependency graph

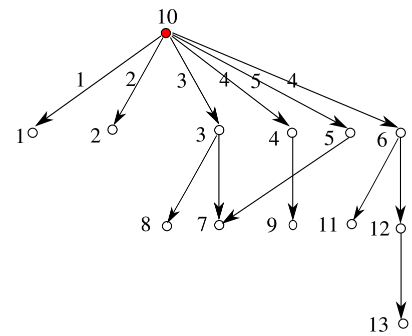

Fig. 1. Dependency graph for example. Each number on the edge represents the number of derived rules

A dependency graph is used to represent the dependency relationships among rules in a ruleset. A dependency graph is a directed graph with a vertex v_i representing the rule r_i for each rule in the ruleset and an edge $e_{c,d}$ that points from a vertex with a larger ID c to another vertex with a smaller ID d where r_c depends on r_d. Moreover, the graph is acyclic because, by definition, only lower priority rules can depend on higher priority rules.

Algorithm 1 Dependency Graph Construction

Input:
- A ruleset with priority

Output:
- A set of dependency graph $G = (V, E)$

procedure Dependency Graph Construction
    Initialize a vertice v_i $\forall r_i$
    for each pair of rules (v_i, v_j) **do**
        if *overlap* $((v_i, v_j))$ and $p(r_i) \neq p(r_j)$ **then**
            add an edge e_{ij} from v_j to v_i
            $w(e_{ij}) \leftarrow$ no. of derived rules

To construct a dependency graph, the matching spaces of rules are pair-wisely compared. Two rules that have overlapping matching spaces and differ in actions are connected by an edge. The weight of the edge $w(e_{ij})$ represents the minimum number of derived rules of r_i that results after disjointing r_i and r_j. Algorithm 1 is used to construct the graph.

C. Top-N selection

Top-N selection is a process to select up to N rules in order to maximize the overall *hit-rate*. The hit-rate of a rule is the fraction of packet classification queries that the rule has provided an action for. Hit-rates are easily obtained using hit-counts, which are available in most managed switches and is part of OpenFlow specification.

D. Selection strategy

We examine each target rule in order to decide whether to include it in the top-N list and what position to place it in the list. Rules that are independent of others (i.e. no conflict), are safely included. Rules that only depends on other target rules can also be included, as long as their relative order is preserved in the resulting list. For target rules that depend on at least one non-target rule, we need to either resolve the conflicts with the dependent rules or include the dependent rules to the top-N list, retaining their relative priorities.

Conflicts can be resolved by splitting the target rule concerned into *smaller* derived rules that are disjoint with the dependent rules. It is noteworthy that in either way, some target rules, starting from bottom of the target list, have to be excluded from the sub-ruleset because we can only have N rules in the sub-ruleset. This unavoidably lowers the overall hit-rate provided by the resulting sub-ruleset.

The proper choice between the two options mainly depends on: 1) The number of derived rules that are required to resolve the dependency; 2) The total hit-rate offered by the dependent rule(s); and 3) The total hit-rate of the target rules that would be excluded in each options.

E. Formal problem statement

The top-N selection constructs a new graph $G' = (V', E')$ with up to N vertices from the original dependency graph $G = (V, E)$. Let $h(v_j)$ to be the hit-rate of the rule r_j. The total hit-rate, given by $\sum_j h(v'_j), v'_j \in V'$, should be maximized.

The essential part of our solution towards the problem is the partition operation. Equation 1 depicts the operation, where r_c is a target rule and $G_c = (V_c, E_c)$ is a dependency sub-graph rooted at r_c in G.

$$(V_c^P, E_c^R) = \text{partition}(v_c, G_c) \tag{1}$$

Specifically, given a target rule r_c, for each of its dependency r_d, the partition operation either: 1) includes the dependent rule r_d and all the rules in the dependency sub-graph rooted at r_d, or 2) partitions the target rule r_c into k derived rules $r_{c,1\dots k}$ such that the derived rules are disjoint with r_d. The first case leads to vertices as elements in set V_c^P that represents the set of derived rules to be included, while the second leads to edges as elements in set E_c^R that represents the dependencies (and hence the dependent rules) that are retained. Therefore the partition operation results in a change from G_c to $G'_c = (V_c^P, E_c^R)$. From the original dependency graph G, one can extract the set of original edges E_c from the dependency sub-graph G_c. By subtracting E_c^R from E_c, we obtains the set of edges E_c^B representing resolved dependencies. Similarly, we can obtain the set of retained vertices $V_c^R = V_c - V_c^P$.

A top-N selection problem can be described as an optimization problem shown below:

$$\begin{aligned}
\max \quad & \sum_j h(v'_j) && (2)\\
\text{s.t.} \quad & V' = V^P \cup V^R && (3)\\
& |V'| \le N && (4)\\
& (V^P, E^R) = \bigcup_i \text{partition}(v_i, G_i) && (5)\\
& V^R = \bigcup_i V_i^R && (6)\\
& E = \bigcup_i (E_i^B \cup E_i^R) && (7)\\
& E' = \bigcup_i E_i^R && (8)\\
\text{where} \quad & v_i \in V \text{ and } v'_j \in V' \\
& i = 1, \cdots, m, m \le |V| \\
& j = 1, \cdots, n, n \le N
\end{aligned}$$

In the optimization, it maximizes total hit-rates in G', which represents the portion of the network traffic that the top-N sub-ruleset can cover. The set of constrains connects the newly constructed graph $G' = V', E'$ to the original one $G = V, E$. That is, the vertex set V' in G' contains two sets: V^P representing derived rule set and V^R representing retained dependent rule set. The E' in equation 8 are edges connecting vertices of target rules to their retained dependent rules. The total number of rules included in the top-N list shall not exceed N, as indicated in constrain 4.

The best case would be the rules in the original ruleset have no dependencies (i.e. $E = \emptyset$, or G has no edges).The optimization problem then becames:

$$\begin{aligned}
\max \quad & \sum_i h(v_i) && (9)\\
\text{s.t.} \quad & |V| \le N && (10)\\
\text{where} \quad & v_i \in V, i = 1, \cdots, m, m \le |V|
\end{aligned}$$

Its corresponding solution is simply sorting all the vertices $v_i \in V$ in descending order of $h(v_i)$ and the first N vertices form the required top-N list.

When dependencies exist, $i.e. E \neq \emptyset$, the optimization relies on a partition operation to produce the new graph G'. Recall that for each dependency $e_{i,j}$, the partition operation needs to decide either to resolve the dependency or to include all rules in the dependency sub-graph rooted at v_j. One can observe that a brute-force optimization algorithm will lead to an exponential complexity, because the partition operation needs to consider all combinations of partition decision. The exponential complexity is evident when several dependent rules overlap with each other. A rule with n dependent rules, has 2^n combinations to take into account.

Let us consider an illustrative example with a dependency graph as depicted in Figure 1. Suppose that the partition operation has decided to partition the target rule $r_1 0$ to resolve the conflict with the dependent rule r_3. After performing the partition, the resulting derived rules may or may not overlap with the remaining dependent rules the same ways as the original rule. In order to make decisions on whether to partition for subsequent dependent rules, such as r_5, we can not make the decision based on the edge weights of r_{10} and r_5 in the original dependency graph. Instead a new dependency graph or at least part of it needs to be reconstructed and weights recalculated, such that it reflects the derived rules. Any algorithm working in this manner has exponential complexity.

IV. Top-N approximation algorithm

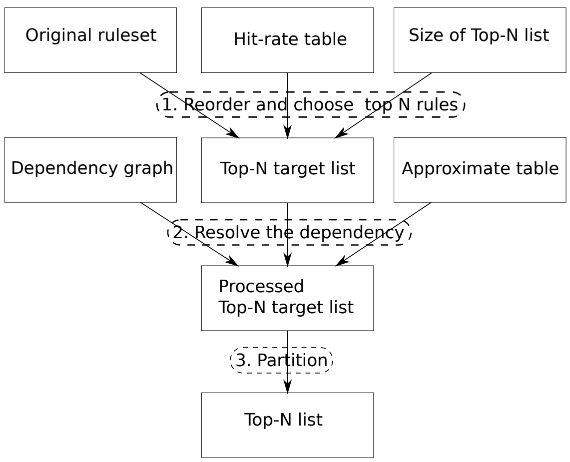

Fig. 2. Top-N selection overview.

Given the complexity of Top-N selection problem, we proposed a heuristic algorithm to solve it efficiently. The proposed approximation Top-N approximation algorithm has the following steps, illustrated in Figure 2: 1) The Top-N target list is constructed by choosing the N rules with highest hit-rates followed by reordering them in descending order of priority. 2) A dependency sub-graph rooted at the first rule in the target list is obtained. Starting from the root, the `partition_decision` algorithm (Algorithm 3) makes a decision of either partitioning the target rule or including the dependent rule in the Top-N list, for each dependent rule in the sub-graph.

After the above two steps, each target rule has a list of dependent rules R^P to resolve conflicts and another list of dependent rules R^D to be included together in the top-N list. Step 3, initializes an empty top-N list and iteratively processes the target rules in descending order of the hit-rates. Using conflict resolution algorithms such as [20], each target rule is partitioned to resolve conflicts with R^P, if any. The resulting derived rules and R_D are added to the top-N list. The iteration terminates when there are more than N rules in the top-N list or all target rules have been processed. This step is represented in Algorithm 2.

A. Approximation table

The approximation table is a pre-computed table that represents all the possible combinations of scenarios where two overlapped rules r_1 and r_2, both depended on by a target rule r_0, and the associated estimated number of additional derived rules of r_0 to disjoint with r_2.

Table I shows the 2D approximation table, which includes 6 combinations.

To understand how the table is computed for each dimension, let us first look at how it represents the overlap relations between two rules. There are three types of overlap relations between two rules in each tuple as shown in Figure 3.

More specifically, given a target rule r_0 that depends on rules r_1 and r_2, and the partition decision for r_1, we can obtain the number of additional derived rules thorough following

Algorithm 2 Top-N Selection

procedure Top-N Selection
  select N target rules with highest reference
  sort them to descending priority
  for each target rule r_i **do**
    T = *partition_decision*(r_i, G)
    if Top-N list is full **then**
      Stop and output the Top-N list
    else
      put T into Top-N list

Algorithm 3 Partition_decision

procedure PARTITION_DECISION
  for each rule r_j that r_i depends on **do**
    if r_j disjoints with all the other rules that r_i depends on and *size*(sub tree rooted at r_j) $\geq v(e_{ij})$ **then**
      decision $\leftarrow$ PARTITION
  for each group **do**
    $S \leftarrow$ all rules in this group
    $s_D \leftarrow 0$ ▷ total number of derived rules
    $r_k \leftarrow$ *getfirst*(S)
    $s_D \leftarrow s_D + v(e_{ik})$
    if *size*$(r_k) \geq s_D$ **then**
      decision $\leftarrow$ PARTITION
      $T \leftarrow$ derived rules
    else
      decision $\leftarrow$ KEEP
      $T \leftarrow$ all rules in the dependency graph root at r_k
    while S is not empty **do**
      $Q \leftarrow$ all rules in S and overlap with r_k
      for each rule $r_k \in Q$ **do**
        $s_l \leftarrow lookup(r_k, r_l)$ ▷ lookup the approximation table
        if $size(r_k) \geq s_l$ **then**
          decision $\leftarrow$ PARTITION
          $T \leftarrow$ derived rules
        else
          decision $\leftarrow$ KEEP
          $T \leftarrow$ all rules root at r_kFrep
  return T

No.	Combination	Additional rules after partition
1	11	3
2	22	0
3	33	0
4	12	2
5	13	4
6	23	4

TABLE I
THE 2D-APPROXIMATION TABLE. NUMBERS IN THE COMBINATION COLUMN REPRESENTS THE TYPE OF OVERLAP.

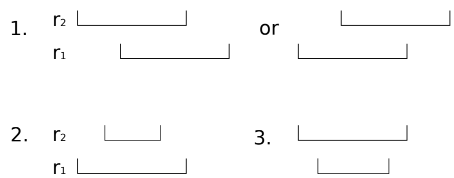

Fig. 3. Three types of overlap relations between two rules in 1-dimension.

steps: (1) Generate all derived rules to disjoint r_0 with r_1. (2) For each derived rule, generate a new set of derived rules to disjoint it with r_2. (3) Permute all the derived rules, and merge those only differing in one dimension (Optional). (4) Count the number of the total derived rules. The 3^{rd} step makes the approximation more tight. However, without it, we can still get a good estimate, which is shown in the section V.

V. EVALUATION

Our simulation experiments shows the proposed algorithm achieves good approximation efficiently.

Both the brute-force optimal algorithm and the approximation algorithm are written in Perl. We synthesized rulesets of 100 to 6500 rules in 800 rules increments as follows. For each rule's source and destination IP addresses, random 32 bits unsigned integers are generated as the upper and lower bounds of each IP address range. Similar, ranges of 16 bits unsigned integers are used for source and destination ports. A single random value is used for protocol. The random numbers are generated with uniform distribution. Random hit-rates are assigned to each rules following three different distributions as shall be discussed in Section V-A. The experiments are conducted on a 64bit Linux machine with Intel Core i7 950, quad core, 3.07 Ghz CPU with 6GB memory. Although a machine with large amount of memory is used, we observe that our program only used 60MBs at peak while processing 1000 rules.

A. Cumulated hit-rates in Top-N

To exam how close the approximate algorithm is to the optimal one, we compared the cumulated hit-rate of both algorithms. The cumulated value is the summation of the hit rate of all the rules in either ruleset or Top-N list. We use a ruleset of 100 rules and varies the size of Top-N from 10 to 100 in steps of 10 rules. Hit-rates of the 100 rules are randomly assigned a value between 0 and 1 with uniform distribution, exponential distribution and normal distribution. In uniform distribution case, an upper-bound of 0.01 is set for all the rules expect one rule who will take the remaining value. Experiments under each distribution are repeated on 10 set of random hit-rates. We ran the brute-force optimal algorithm and our approximation algorithm on the rulesets for each N. For each distribution, the average of the sum of hit-rates in the resulting 10 top-N rulesets are presented in Figure 4, 5 and 6, respectively. The original ruleset is also included as a baseline by simply selecting the first N rules according to the original rule order.

The differences between the approximate and optimal algorithm are shown in Figure 7. The three polylines indicate

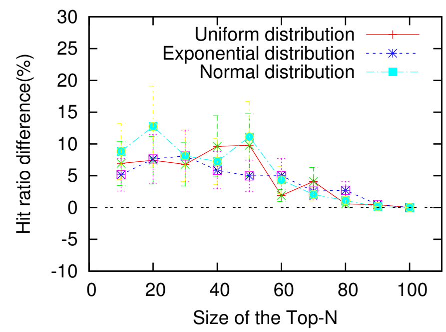

Fig. 7. Difference between the approximate and the optimal algorithm

that the Top-N list computed by the approximate algorithm are around 10% worse than the optimal one. This difference between the two algorithms decreases as the size of Top-N grows. Especially, when the size of Top-N list and the ruleset is equal, both algorithms give an identical Top-N list, since in such case, the job for them is just to reorder the ruleset in priority descendent order.

B. Running time

We also recorded the running time of our approximate algorithm on different size of ruleset, ranging from 100 to 6500. The approximation algorithm is used to calculate 25%, 50%, 75% and 100% of Top-N rules in ruleset of different sizes. The result is represented in figure 8.

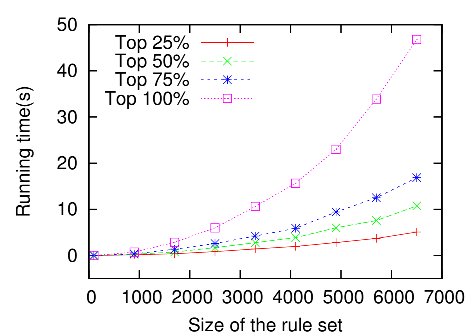

Fig. 8. The running time of program implementing the approximate algorithm.

The experiment on the running time reflects two trends of the approximate algorithm. First, it takes longer, as the size of the original ruleset increases; Second, for a specific size of original ruleset, the running time of the approximate algorithm

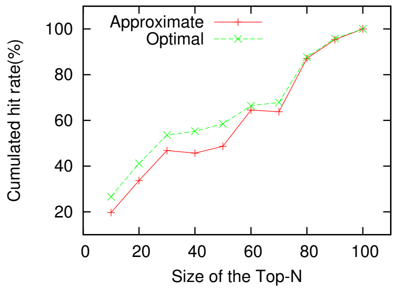

Fig. 4. Cumulative hit-rate with uniform distributed hit-rate dataset

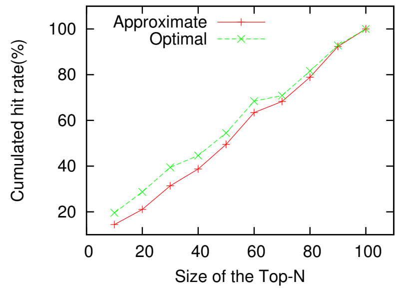

Fig. 5. Cumulative hit-rate with exponential distribution hit-rate dataset

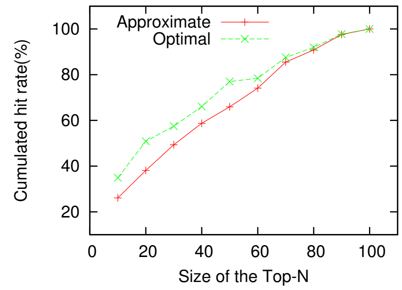

Fig. 6. Cumulative hit-rate with normal distribution dataset

also increases with the size of Top-N list. Note that no result is obtained from the optimal algorithm for rulesets of 200 or more rules because it failed to complete in a reasonable time (one day in iour experiments). Our approximation algorithm is able to finish computing with at most 20 seconds for selecting a 75% top-N rules out of 6500 rules. For most settings, it takes only seconds to finish.

VI. Conclusion

This paper suggested four basic requirements to a top-N sub-ruleset selection problem. An optimization framework has been proposed and the challenges of the problem are identified. A greedy-like heuristic algorithm is proposed to choose rules among those with the highest hit-rates, their associated dependencies and derived rules that have their dependencies resolved. The algorithm does not require conflict-free ruleset to be pre-computed. The required hit-rate statistics are readily available with little overhead from most managed switches, routers and firewalls.

The simulations show that the top-N approximation algorithm achieves cumulative hit-rate is reasonably close to the optimal. The running time is in the order of seconds and thus is able to respond to dynamic changes in traffic pattern.

As future work we intend to improve the performance of constructing dependency graph by partially modifying existing graph to reflect the change of the rules or the order. We would also enhance the precision of approximation table to make the Top-N approximation algorithm closer to the optimal solution.

References

[1] A. Kennedy, X. Wang, and B. Liu, "Energy efficient packet classification hardware accelerator," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–8.

[2] A. X. Liu, E. Torng, and C. R. Meiners, "Firewall compressor: An algorithm for minimizing firewall policies," in *INFOCOM '08: Proc. of the 27th Conf. on Comp. Comm.*, Phoenix, AZ, USA, 2008.

[3] H. Kaplan, E. Molad, and R. E. Tarjan, "Dynamic rectangular intersection with priorities," in *STOC '03: Proc. of the thirty-fifth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 2003.

[4] D. A. Applegate, G. Calinescu, D. S. Johnson, H. Karloff, K. Ligett, and J. Wang, "Compressing rectilinear pictures and minimizing access control lists," in *SODA '07: Proc. of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Philadelphia, PA, USA, 2007.

[5] C. R. Meiners, A. X. Liu, and E. Torng, "Topological transformation approaches to optimizing tcam-based packet classification systems," in *SIGMETRICS '09: Proc. of the 11th intl joint conf. on Measurement and modeling of comp. sys.*, New York, NY, USA, 2009.

[6] H. Hamed and E. Al-Shaer, "Dynamic rule-ordering optimization for high-speed firewall filtering," in *ASIACCS '06: Proc. of the 2006 ACM Symp. on Inform., comput. and commun. security*. New York, NY, USA: ACM, 2006, pp. 332–342.

[7] S. Acharya, M. Abliz, B. Mills, T. F. Znati, J. Wang, Z. Ge, and A. Greenberg, "OPTWALL: A hierarchical traffic-aware firewall," in *NDSS '05: Proc. of the 12th Annual Network and Distributed System Security Symposium*, San Diego, CA, USA, 2005.

[8] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: taking control of the enterprise," in *SIGCOMM '07: Proc. of the 2007 Conf. on Applicat., technol., architectures, and protocols for comput. commun.* New York, NY, USA: ACM, 2007.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *SIGCOMM Comput. Commun. Rev.*, 2008.

[10] N. Gude, T. Koponen, J. Pettit, B. Pfaff, M. Casado, N. McKeown, and S. Shenker, "Nox: towards an operating system for networks," *SIGCOMM Comput. Commun. Rev.*, 2008.

[11] A. Greenberg, G. Hjalmtysson, D. A. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "A clean slate 4D approach to network control and management," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 5, 2005.

[12] S. Acharya, J. Wang, Z. Ge, T. Znati, and A. Greenberg, "Traffic-aware firewall optimization strategies," in *ICC '06. IEEE International Conference on Communications, 2006.*, vol. 5, June 2006.

[13] J. Wallerich, H. Dreger, A. Feldmann, B. Krishnamurthy, and W. Willinger, "A methodology for studying persistency aspects of internet flows," *SIGCOMM Comput. Commun. Rev.*, vol. 35, no. 2, pp. 23–36, 2005.

[14] P. Gupta, , P. Gupta, and N. Mckeown, "Packet classification using hierarchical intelligent cuttings," in *in Hot Interconnects VII*, 1999, pp. 34–41.

[15] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 213–224.

[16] J. M. Gonzalez, V. Paxson, and N. Weaver, "Shunting: a hardware/software architecture for flexible, high-performance network intrusion prevention," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2007, pp. 139–149.

[17] E. Cohen and C. Lund, "Packet classification in large isps: design and evaluation of decision tree classifiers," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 73–84, 2005.

[18] G. Yan, S. Chen, and S. Eidenbenz, "Dynamic balancing of packet filtering workloads on distributed firewalls," in *Quality of Service, 2008. IWQoS 2008. 16th International Workshop on*, June 2008, pp. 209–218.

[19] H. Fu and M. Zhang, "Online adaptive firewall allocation in internet data center," *Computer Communications*, vol. 29, no. 10, pp. 1858 – 1867, 2006, monitoring and Measurements of IP Networks. [Online]. Available: http://www.sciencedirect.com/science/article/B6TYP-4GFV2B9-1/2/869517e9a153d81d923c43aae2e28195

[20] M. G. Gouda and A. X. Liu, "Structured firewall design," *Comput. Netw.*, vol. 51, no. 4, pp. 1106–1120, 2007.