Verification and Synthesis of Firewalls Using SAT and QBF

Shuyuan Zhang, Abdulrahman Mahmoud, Sharad Malik
Princeton University
{shuyuanz, mahmoud, sharad}@princeton.edu

Sanjai Narain
Applied Communication Sciences
snarain@appcomsci.com

Abstract—**Firewalls are widely deployed to safeguard the security of networks and it is critical for enterprise networks to have firewalls to prevent malicious attacks and to guarantee the normal functioning of the network. Firewalls prevent dangerous packets from entering the inner network by looking up the Access Control List (ACL) to permit or drop certain packets. However, ACLs often suffer from redundancy problems, which can degrade the performance of firewalls and the network. The contribution of this paper is threefold: 1) we present a Boolean Satisfiability (SAT) based technique that can compare the equivalence and inclusion relationship between two firewalls, which is very valuable for the testing between a given firewall and an optimized one, 2) we present a technique to discover redundancies within a firewall, and 3) we formulate the ACL optimization problem as a *Quantified Boolean Formula problem (QBF)* and explore its practical application using a QBF solver.**

Acknowledgments

This work has been funded by Air Force Research Laboratory under contract FA8750-11-C-0012.

I. Introduction

Firewalls are very common in modern networks and most private networks utilize a firewall to protect and monitor incoming and outgoing traffic, using a set of rules developed by a network administrator. Firewalls help filter contact between a network and the rest of the Internet, dropping packets that do not satisfy the priority rules outlined by an administrator. These rules are designed to examine certain fields of a packet header, and subsequently either forward the packet if it is on the "permit" list, or drop the packet if it is not included in the set of acceptable rules.

However, one issue that plagues firewalls is that these rules are not always optimized, with many firewalls containing redundancies in the rule set [1]. Additionally, firewalls often contain policy errors which are difficult to detect due to the lack of available policy verification tools, thus compromising the security of the network [2]. Firewall verification entails using formal analysis for ensuring that a set of firewall policy rules are consistent with a specification, and includes checking for redundancies within a firewall and equivalence checking between firewalls. Firewall verification can assist in discovering superfluous policy rules that can slow down the travel time of packets to and from a network. Verification can also help ensure that even if a rule set does not contain any conflicts, it is prioritized in the correct sequence to avoid undesired packet denial. Additionally, one can use *inclusion checking* to determine if one firewall has a stricter policy than another firewall by permitting only a subset of packets allowed by the second firewall.

In this paper, we present a SAT based method of firewall verification for equivalence checking between two firewall rule sets to check if they have the identical behavior for every incoming packet. We also use our SAT-based verification tool for inclusion checking between two firewalls, which is very helpful in checking if one firewall has a stricter policy than another firewall. We also demonstrate the application of verification in detecting redundancy in a firewall. We additionally present a firewall synthesis method based on a Quantified Boolean Formula (QBF) formulation and use a QBF solver to synthesize an optimal firewall. This paper is organized as follows: Section II presents the background of this paper, i.e. an overview of the firewalls that we are targeting. Section III and section IV discuss the technical details on encoding the firewalls and verification properties as logic formulas. Following the discussion of the encoding, we present experimental data that tests our encoding and formulation in section V. Next, we discuss the related work and finally draw conclusions in section VI and section VII.

II. Background

A. Firewalls

The firewalls that we are studying in this paper are abstracted as a one-input one-output device that can model both software-based and hardware-based firewalls. A firewall takes packets from the network as an input and returns the decision about whether the packets should be dropped or permitted.

The main component of a firewall is a matching table, which is composed of *strictly prioritized* matching rules. By strictly prioritized, we mean that there do not exist two rules that have the same priority. If a packet matches a rule, that rule will automatically dominate all the rules with lower priorities. It also means that the packet does not match any rules with higher priorities.

Each matching rule has two fields, one matching field and one action field. The matching field determines which of the packets should be processed by this rule and it is usually in the packet header information. Packet payload is not considered although there are firewalls that require application-layer information to make a decision. For simplicity, we do not consider this case here. The matching field can be quite flexible. For example, we can include source/destination IP prefixes and source/destination TCP/UDP port number in the matching field. The matching field is represented using a flat ternary array. A '1' or a '0' entry match a '1' or a '0' in the corresponding header, respectively. We use the wildcard 'x' to represent a match to both '0' and '1'. For example, consider a matching field that has two eight-bit prefix addresses, 224/3 and 120/5. 224 is 11100000 and 120 is 01111000 so 224/3 is 111xxxxx and 120/5 is 01111xxx. Then, the matching field is (111xxxxx01111xxx). The action of a firewall is simply "drop" or "permit", which specifies whether the packet should be dropped or not.

The matching table is considered to be completely static during verification. Although stateful firewalls usually change state during operation, they can be regarded as static between two matching rule updates. Hence our firewall is a snapshot of stateful firewalls at a single instance of time.

B. Firewall Verification and Synthesis

1) Firewall Equivalence Checking: One of the major properties that is studied in this paper is firewall equivalence. Given two firewalls, we determine if they have identical behavior, i.e. they drop/permit exactly the same set of packets. This is an important property because it can be used in firewall optimization to compare whether a given firewall and an optimized firewall are identical. The fundamental principle in firewall equivalence checking is trying to find the difference between two firewalls. If no differences exist between them, the two are equivalent.

2) Firewall Inclusion Checking: Another relationship between two firewalls is inclusion. If firewall A permits only a subset of the packets permitted by firewall B, then we say firewall A is stricter than firewall B since it drops more packets. Often, when we want to replace one firewall with another, we have to make sure that the new firewall does not sacrifice the security requirements of the old one and is at least as strict as the replaced firewall.

3) Firewall Rule Redundancy Checking: A firewall may also have redundant rules within its own matching table [3]. We define a redundant rule as one that once removed from the rule set produces a matching table that is logically equivalent to the original ruleset. For example, one rule may accept packets destined to 10.0.0.0/32 while another rule accepts all packets destined to 10.0.0.0/24. In this case, if the first rule is removed, all packets destined to 10.0.0.0/32 are still accepted by the latter rule, rendering the first rule redundant.

4) Firewall Synthesis: Firewall synthesis is concerned with synthesizing a firewall with exactly the same behavior as a given firewall such that the synthesized firewall's specifications have the smallest number of rules installed. Here, we use a *Symbolic Firewall*, which is parameterized in the number of the rules. The advantage of a symbolic firewall is that we can use variables to program the firewall and an optimal synthesis determines the values of the variables and thus the rules. We leave all the technical details for the following sections.

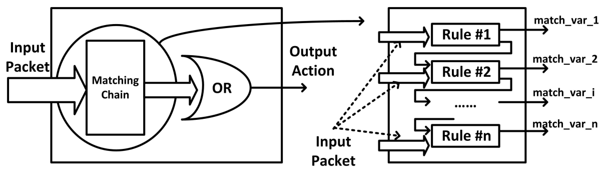

Fig. 1. The Overview of Firewall Encoding

III. Firewall Equivalence, Inclusion, and Redundancy Checking

In this section, we discuss the details of how we check the equivalence and inclusion between two firewalls, and redundancy checking within a firewall. We will first present how to encode a fixed firewall, in which all the rules are given and fixed and then we will present how to use the encoding of a fixed firewall to do firewall equivalence, inclusion, and redundancy checking.

A. The Encoding of Fixed Firewalls

Figure 1 depicts the basic structure of our firewall encoding. As mentioned in section II, the firewall is a one-input one-output device. The input is a packet, which is modeled as a set of Boolean variables. We use $b_i, i \in [1, N_{input}]$ to represent the Boolean variable for the input bit i and N_{input} is the total number of input bits needed for the packet. The input packet is fed to the matching table. Since every rule in the firewall is prioritized, the matching table is essentially a matching rule chain. Every rule has two outputs, one $match\_var$ that indicates if the packet does not match any of the previous rules and it matches the current rule, and one $total\_match\_var$ that tells the next rule if the packet has matched previous rules or the current rule. This $total\_match\_var$ is fed to the next rule in the matching table.

We use m_i to represent the $match\_var$ for rule #i, where $i \in [1, N_{rule}]$ and N_{rule} is the total number of rules, and use p_i to represent the $total\_match\_var$ for rule #i, where $i \in [2, N_{rule}]$ and p_i is the input of rule i and the output of the rule $i-1$. As described in section II, the matching field in every rule is composed of '0', '1', and 'x'. We use $r_{i,j}$ to represent the value of the matching field bit for rule #i and bit j and we use $k_{i,j}$ to represent the formula of that bit. Then

$$k_{i,j} = \begin{cases} b_j & \text{if } r_{i,j} \text{ is } 1 \\ \neg b_j & \text{if } r_{i,j} \text{ is } 0 \\ \text{TRUE} & \text{if } r_{i,j} \text{ is x} \end{cases} \tag{1}$$

If $r_{i,j}$ is 1, it specifies that the matching bit matches 1, which is the positive phase of the input bit b_j. Similarly, if it is 0, it matches the negative phase of the input bit. If it is an 'x', it matches both cases, which is always true. The match bit can be calculated as:

$$m_i = \left(\bigwedge_{j=1}^{N_{input}} k_{i,j} \right) \wedge (\neg p_i) \text{ if } i \geq 1 \tag{2}$$

$$p_i = \begin{cases} FALSE & \text{if } i == 1 \\ m_{i-1} \vee p_{i-1} & \text{if } i \geq 2 \end{cases} \tag{3}$$

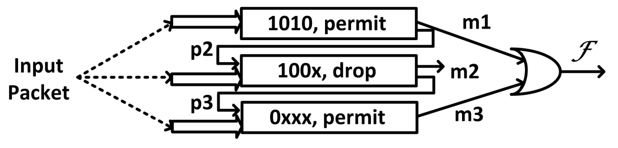

Fig. 2. Encoding Example

Then the output of the firewall, $\mathcal{F}(B)$ is

$$\mathcal{F}(B) = \bigvee_{i \in S_{permit}} m_i \tag{4}$$

where $B = \{b_1, ..., b_{N_{input}}\}$ and S_{permit} is the set of rules with an action of 'permit'. If the formula evaluates to 1, the firewall permits the packet; otherwise, it drops the packet. We make the assumption that the default rule is a 'drop' rule, which means that all the packets which do not match any rules specified in the firewall will be dropped by default.

Consider the simple example depicted in Figure 2. Suppose we have a firewall with 3 rules and each rule has a matching field of 4 bits. The rules are in the order (1010, permit), (100x, drop), and (0xxx, permit). Thus, $p_1 = FALSE$, $m_1 = b_1 \wedge (\neg b_2) \wedge b_3 \wedge (\neg b_4) \wedge (\neg p_1)$, $p_2 = m_1 \vee p_1$, $m_2 = b_1 \wedge (\neg b_2) \wedge (\neg b_3) \wedge (\neg p_1)$, $p_3 = m_2 \vee p_2$, $m_3 = (\neg b_1) \wedge (\neg p_3)$, and $\mathcal{F}(b_1, b_2, b_3, b_3) = m_1 \vee m_3$.

To encode a firewall with N_{rule} rules in it, the variables required are simply m_i, p_i, and b_i and hence, the total number of variables are about the sum of the total number of the three variables. Each rule has one m_i and p_i and we have N_{rule} rules. Each input bit has one b_i and we have N_{input} input bits. Thus, the total number of variables required is $2 \times N_{rule} + N_{input}$. We use the *Conjunctive Normal Form (CNF)* formula representation and the *Tseitin Transformation* [4] to convert the expression in formula 4 into a CNF formula. The number of CNF clauses required depends on how many wildcards, i.e. 'x', appear in each rule. If on average there are $N_{wildcard}$ wildcards in each rule, the total number of CNF clauses is about $(N_{input} - N_{wildcard}) \times N_{rule}$, since if it is a wildcard the clause is always true, which is a trivial case. The reason is that most of the operations in the encoding are AND and OR and the number of CNF clauses used to encode an AND or OR operation is the number of input variables plus 1.

B. Firewall Equivalence and Inclusion Checking

To check if two firewalls are equivalent, we can use an XOR operation to connect the output of the two firewalls and see if it is different. For example, given two firewalls $\mathcal{F}_1(B)$ and $\mathcal{F}_2(B)$, if $\mathcal{F}_1(B) \oplus \mathcal{F}_2(B)$ is satisfiable, it means there exists at least one input packet B, such that the two firewalls differ in their actions; otherwise, there is no such packet that exists.

To check inclusion, we need a formula that can only return false if the stricter firewall returns true but the other firewall returns false. Thus, $\neg(\mathcal{F}_1(B) \rightarrow \mathcal{F}_2(B))$ checks if firewall 2 includes firewall 1. If this formula is unsatisfiable, this means that firewall 1 can never permit a packet that is blocked by firewall 2, i.e. firewall 1 permits a subset of packets permitted by firewall 2 and it is stricter than firewall 2. If it is satisfiable, the satisfying assignment to the input packet bits serves as a counterexample for the inclusion property.

C. Firewall Rule Redundancy Checking

To check if a firewall contains redundancies amongst its rules, we need to check whether a matching table created by the removal of a rule (or multiple rules) is equivalent to that of the original unmodified matching table. We add a *control bit* for each rule so that we can control whether that rule is present in the ruleset or not. We use o_i to represent the control bit and the formula for m_i is

$$m_i = \left(\bigwedge_{j=1}^{N_{input}} k_{i,j} \right) \wedge (\neg p_i) \wedge (\neg o_i) \text{ if } i \geq 1 \tag{5}$$

If o_i is 0, the rule behaves just like a normal rule but if it is 1, that rule will never match any packets and can be regarded as a discarded rule.

Then we can use the array of the control bits to do redundancy removal. We start from the first rule and make its control bit to be 1 and compare the equivalence between the new firewall and the original firewall. If they are equivalent, it means the first rule is redundant and can be removed. Then we keep the control bit for the first rule as 1 and do the same procedure for the second rule and so on until we find all the assignments to the control bits. This results in a minimal set of rules, i.e there is no subset of the final set of rules that is equivalent to the original firewall. However, this may not be a minimum subset, i.e subset of the least cardinality, as the rules were removed in a specific order.

IV. Firewall Synthesis

For the firewall synthesis problem, we use a symbolic firewall with rules represented as symbols instead of being fixed and compare it against the given firewall, which serves as a specification. If we can find an assignment for all symbols such that we can make the two firewalls equivalent, then these symbol values serve as the rules definition of the new firewall.

A. The Encoding of Symbolic Firewalls

The difference between a fixed firewall and a symbolic firewall is that the symbolic firewall has symbols instead of fixed rules. Thus, the symbolic firewall has two sets of inputs: one packet which is the same as the fixed firewall, and one set of input symbols which is used to program the firewall. The output is a single binary bit to indicate the action of the firewall.

The structure of the matching table of a symbolic firewall is the same as that of a fixed firewall but the number of matching rules is parameterized, which means that N_{rule} becomes a variable, and we use n to represent it. The encoding of matching rules is completely changed and as a result the formulas for m_i and p_i do not change but the formula for $k_{i,j}$ becomes

$$k_{i,j} = (v^1_{i,j} \wedge b_j) \vee (\neg v^1_{i,j} \wedge \neg b_j) \vee v^2_{i,j} \tag{6}$$

where $v^1_{i,j}$ and $v^2_{i,j}$ are the two variables to program the bit at rule #i and bit j. It captures that if $v^2_{i,j}$ is 1, $k_{i,j}$ evaluates to TRUE to capture the 'x' case. If $v^1_{i,j}$ is 0, $k_{i,j}$ equals to $\neg b_j$; otherwise, it is b_j to capture the '1' case.

The action of each rule is made symbolic using variable v^3_i. For every rule, the output action is $v^3_i \wedge m_i$. It means it is a permit rule as long as v^3_i evaluates to TRUE. Therefore, the output of the symbolic firewall $\mathcal{F}^s(B, n, V)$ is

$$\mathcal{F}^s(B, n, V) = \bigvee_{i \in S_{all}} v^3_i \wedge m_i \qquad (7)$$

where V is the set of all variables v^1, v^2, and v^3 and S_{all} is the set of all rules.

The total number of Boolean variables required to encode a symbolic firewall with n rules is about $2 \times N_{input} \times n + 3 \times n + N_{input}$. Since we need 2 variables to represent each matching bit, in total we have $N_{input} \times n$ such bits. For each rule, we need another variable to encode the action and together with m_i and p_i, we have an extra 3 variables for each rule.

B. Firewall Synthesis

We encode a symbolic firewall with n rules and find V such that the symbolic firewall is equivalent to the given fixed firewall, which serves as the specification. Here we use a *Quantified Boolean Formula (QBF)* formulation to determine the assignments for V.

Given a firewall $\mathcal{F}(B)$ and a symbolic firewall with n rules installed $\mathcal{F}^s(B, n, V)$, we need to find an assignment for all variables in V such that for every input packet B, $\neg(\mathcal{F}(B) \oplus \mathcal{F}^s(B, n, V))$ is satisfiable, and there exists an equivalent firewall to the given one with n rules installed. Expressed as a QBF formula, it becomes

$$\exists V \forall B : \neg(\mathcal{F}(B) \oplus \mathcal{F}^s(B, n, V)) \qquad (8)$$

If the QBF solver returns satisfiable for this formula, we can build an equivalent firewall with n rules installed and the assignment to the V variables that makes the formula true defines the rules; otherwise, there does not exist such firewall. To find a firewall with the smallest number of rules installed, we do a binary search on the number of rules n.

V. Experiments

We used Minisat [5] as our SAT engine to check for equivalence and inclusion between two generated firewalls and we use BDepQBF [6] as our QBF solver for firewall synthesis. These are considered amongst the fastest SAT and QBF solvers respectively based on results of tool competitions [7], [8]. All experiments are run on Gentoo Linux with kernel 3.4.4. We used an AMD Phenom 3.3 GHz processor with 8 gigabytes of RAM for these experiments.

To construct different firewalls to test, we used the ClassBench benchmark generator [9]. The ClassBench program allows the user to input several different parameters for priority rule generation. For our experiments, we used the firewall parameter file input to simulate real firewall rule sets, available with the downloadable benchmarks. Additionally, we were provided with three input parameters to set, which expanded upon the parameter file: the address scope, the application scope, and the smoothness of the generated rule set. The address scope adjusts the bias of how specific address prefixes are, indicating whether longer or shorter prefixes are desired. The application scope affects the protocol specifications of the rule, and the smoothness value determines the distribution of prefixes across the protocol. For simplicity, we kept all three parameters at their default value: uniform address scope, uniform application scope, and default smoothness as set by the parameter input file.

The matching rules generated by ClassBench have 6 fields, which are source/destination IP addresses, source/destination ports, protocol, and flags. In total, they account for 136 matching bits, i.e, N_{input} is 136.

A. Equivalence Checking

In order to test our SAT based equivalence checker, we wanted to generate two firewalls which were very similar to one another, but with subtle differences. This represents the difficult care for equivalence checking.We first used the ClassBench program to create rulesets of size 50 to 26000. We made a copy of each of these rulesets, but with a small mutation. We tested for three different types of mutations:

1) Flipping a random bit in the firewall.
2) Deleting one random rule in the firewall.
3) Swapping two random rules in the firewall.

For the first mutation, we chose a random bit in the entire ruleset, and depending on its value, flipped it to one of the other two bits. If the random bit was an 'x' , we demoted it to either a 0 or a 1, with equal probability. If the random bit was either a zero or a one, we promoted it to an x.

For the second mutation, we uniformly chose one random rule, and removed it from the ruleset. This mutation was chosen to see how well the equivalence checker could pick up a slight difference between two rulesets that are very similar.

For the third and final mutation, we chose two random rules uniformly, and swapped their positions. Since the original ruleset is prioritized, the swapping of two rules could potentially cause a portion of the ruleset to be totally ignored, as a higher prioritized rule is assumed to be more specific than any rule below it in priority.

Figure 3, 4, and 5 show the results. As we can see, the execution time increases as the total number of rules grows since the formula size becomes larger. SAT cases (not equivalent) are generally faster than UNSAT cases (equivalent) since for SAT it often only explores part of the search tree. Another thing to note is that all the data points scatter around the plane and it is because the benefit of the heuristics built inside the SAT engine can differ from case to case. Sometimes when we are lucky, it is even faster to check a larger test case. The one that executes the longest is the UNSAT case for mutation 3 with about 26000 rules. It takes about 48 minutes to finish. The SAT instance has a total of 103843 Boolean variables and 5.2 million CNF clauses.

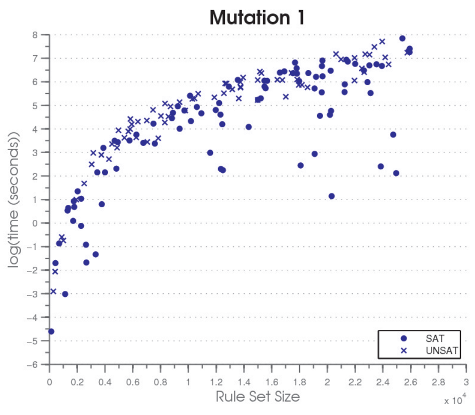

Fig. 3. Mutation 1: Flipping a random bit.

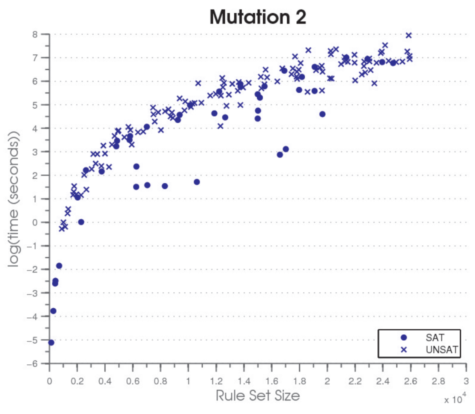

Fig. 4. Mutation 2: Deleting one random rule.

B. Inclusion Checking

To check for inclusion of one firewall ruleset within another ruleset, we again began by developing a set of differently sized benchmarks. We then created a new firewall for each benchmark by removing the lowest priority 10% of the rules. Since it permits fewer packets to go through, our inclusion checking tool can only return UNSAT (i.e. the inclusion check will pass). Figure 6 plots the execution time.

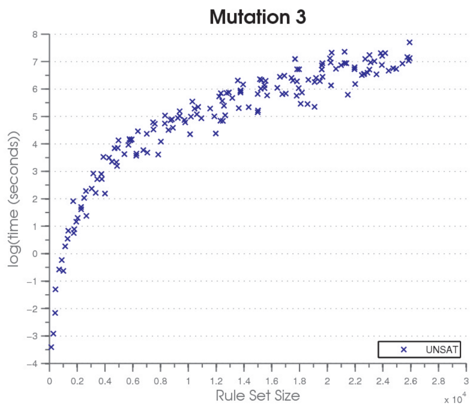

Fig. 5. Mutation 3: Swapping two random rules.

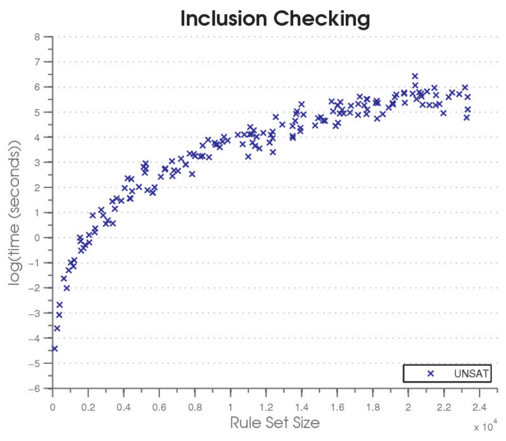

Fig. 6. Inclusion Checking

Total Rules	Redundant Rules	Time (Sec)	Percent Redundancy
130	29	2.228	22.3%
286	180	15.357	62.9%
438	268	66.245	61.2%
702	447	237.368	63.7%
887	627	479.874	70.7%
1007	773	743.023	76.8%
1135	757	953.684	66.7%
1355	870	2080.754	64.2%
1753	1167	4938.189	66.6%
1932	1371	5004.529	71.0%

TABLE I
REDUNDANCY REMOVAL RESULTS

C. Rule Redundancy Checking

To check for redundancies within a set of firewall rules, we took our original ClassBench files and attempted to remove rules one at a time to reduce the size of the matching table. We attempted to remove the largest cumulative set of rules while maintaining equivalence to the original ruleset. This process was relatively slow, as shown in Table I. For a matching table with almost 2000 rules, the process took about 83 minutes and found approximately 1400 redundancies. This means that were we to remove all redundancies found, the matching table would behave identically to the full set, albeit much smaller in size. Another interesting thing to notice is that the percent of redundant rules is very high and the reason is that the default action for a packet that does not match any rules is “drop”. Therefore, we can remove all of the rules that have an action of “drop” and have no overlap with other “permit” rules.

D. Firewall Synthesis

The firewall synthesis is prohibitively slow and the speed slows down dramatically as the size of the input packet grows. This is because in our QBF formula we have a universal quantification for the input packet bits. It can manage the firewall size of 20 input packet variables and 3 rules ($N_{input} = 20$, $n = 3$, and the size of V is 123) in about 10 minutes but it times out for a size of 25 variables. Thus, while the synthesis problem can be encoded as a QBF problem, even simple instances are beyond the reach of the best current QBF solvers.

VI. Related Work

In the past few years, there has been some work in firewall analysis and optimization. [10] proposed a method to minimize the number of rules in *Ternary Content-Addressable Memories (TCAM)*, which are used for packet classification. Similar to our method, they express the output of a TCAM using a Boolean formula. They use *Disjunctive Normal Form (DNF)* and two-level DNF optimization to minimize the number of rules required. They implement a heuristic solver because the size of the problem is too large to get an exact solution. [3], [11], [12] propose to remove redundancies in a firewall based on a decision tree but these techniques are unable to compare firewalls and do not guarantee that they are minimized. [13], [14] systematically summarize different kinds of conflicts for both standalone and distributed firewalls. There are also some older works that target optimizing IP routing tables [15], [16]. Since an IP routing table can have multiple routing decisions instead of only two decisions for a firewall, IP routing table optimization is a harder problem than firewall optimization in terms of problem size. However, they use longest-prefix matching and it is difficult for them to adapt to priority-based matching and they also did not propose how to compare the relationship between two routing tables. In a broader context, [17], [18] discussed about the verification of general network properties, including reachability and forwarding loops. [19], [20] have discussed how to use SMT for firewall equivalence and inclusion checking and we plan to compare the performance of the two encodings in the near future.

VII. Conclusions and Future Work

In this paper, we present a SAT based method that can check the equivalence and inclusion relationship between two firewalls and how to use SAT to remove redundancy in a firewall. These techniques are shown to scale well for practical sized firewalls using state-of-the-art SAT solvers. We also proposed using a QBF formulation to solve the firewall synthesis problem. Our formulation, even with the fastest QBF solvers, is very slow. Our ultimate goal is to develop a run-time firewall checking tool that can do firewall verification upon firewall configuration changes but this will require significant tool speedups. We are exploring the use of incremental verification techniques for this purpose. Firewalls are a simple case of a network middlebox and our technique can be applied to network middlebox verification and synthesis. Since switches and routers usually have several forwarding ports and routers, in particular, are capable of modifying the packet header, a more complex encoding is needed to model the middleboxes. However, the fundamental principles remain the same.

References

[1] H. Acharya and M. Gouda, "Firewall verification and redundancy checking are equivalent," in *INFOCOM, 2011 Proceedings IEEE*, april 2011, pp. 2123 –2128.

[2] A. Wool, "A quantitative study of firewall configuration errors," *Computer*, vol. 37, no. 6, pp. 62 – 67, june 2004.

[3] A. X. Liu, C. R. Meiners, and Y. Zhou, "All-match based complete redundancy removal for packet classifiers in TCAMs," in *Proceedings of the 27th Annual IEEE Conference on Computer Communications (Infocom)*, Phoenix, Arizona, April 2008.

[4] T. G.S., "On the complexity of proof in prepositional calculus," vol. 8, pp. 234–259, 1968.

[5] N. Een and N. Sorensson, "An extensible sat-solver," in *Theory and Applications of Satisfiability Testing*, ser. Lecture Notes in Computer Science, E. Giunchiglia and A. Tacchella, Eds. Springer Berlin / Heidelberg, 2004, vol. 2919, pp. 333–336.

[6] F. Lonsing and A. Biere, "Depqbf: A dependancy-aware qbf solver," *Journal on Satisfiability, Boolean Modeling and Computation*, vol. 7, 2010.

[7] D. Le Berre and L. Simon, "The sat 2005 competition," SAT Competition 2005. [Online]. Available: http://www.satcompetition.org/2005/

[8] N. Massimo, "Fifth qbf solvers competition," Fifth Competitive Evaluation of QBF Solvers. [Online]. Available: http://www.qbflib.org/

[9] D. Taylor and J. Turner, "Classbench: a packet classification benchmark," in *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, vol. 3, march 2005, pp. 2068 – 2079 vol. 3.

[10] R. McGeer and P. Yalagandula, "Minimizing rulesets for tcam implementation," in *INFOCOM 2009, IEEE*, april 2009, pp. 1314 –1322.

[11] A. Liu, E. Torng, and C. Meiners, "Firewall compressor: An algorithm for minimizing firewall policies," in *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, april 2008, pp. 176 –180.

[12] C. R. Meiners, A. X. Liu, and E. Torng, "Tcam razor: A systematic approach towards minimizing packet classifiers in tcams," in *Proceedings of the 15th IEEE International Conference on Network Protocols (ICNP)*, Beijing, China, October 2007.

[13] E. Al-Shaer and H. Hamed, "Modeling and management of firewall policies," *Network and Service Management, IEEE Transactions on*, vol. 1, no. 1, pp. 2 –10, april 2004.

[14] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan, "Conflict classification and analysis of distributed firewall policies," *Selected Areas in Communications, IEEE Journal on*, vol. 23, no. 10, pp. 2069 – 2084, oct. 2005.

[15] R. Draves, C. King, S. Venkatachary, and B. Zill, "Constructing optimal ip routing tables," in *INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 1, mar 1999, pp. 88 –97 vol.1.

[16] S. Suri, T. Sandholm, and P. Warkhede, "Compressing two-dimensional routing tables," *Algorithmica*, vol. 35, pp. 287–300, 2003, 10.1007/s00453-002-1000-7.

[17] P. Kazemian, G. Varghese, and N. McKeown, "Header space analysis: static checking for networks," in *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, ser. NSDI'12. Berkeley, CA, USA: USENIX Association, 2012, pp. 9–9. [Online]. Available: http://dl.acm.org/citation.cfm?id=2228298.2228311

[18] M. Reitblatt, N. Foster, J. Rexford, C. Schlesinger, and W. David, "Abstractions for network update," *SIGCOMM Comput. Commun. Rev.*, Aug. 2012.

[19] S. Narain, R. Talpade, and G. Levin, *Network Configuration Validation*, ser. Guide to Reliable Internet Services and Applications. Springer Verlag, 2010.

[20] S. Narain, "Motivating constraint solving for networking," Formal Methods in Networking, Princeton University Computer Science Seminar Course. [Online]. Available: http://www.cs.princeton.edu/courses/archive/spring10/cos598D/FMINLecture3.pdf