Fast Algorithms for Local Inconsistency Detection in Firewall ACL Updates

S. Pozo, R. Ceballos, R. M. Gasca, A. J. Varela-Vaca
Department of Computer Languages and Systems, ETS Ingeniería Informática,
University of Seville, Avda. Reina Mercedes S/N, 41012 Sevilla, Spain
{sergiopozo,ceball,gasca}@us.es
www.lsi.us.es/~quivir

Abstract

Filtering is a very important issue in next generation networks. These networks consist of a relatively high number of resource constrained devices with very special features, such as managing frequent topology changes. At each topology change, the access control policy of all nodes of the network must be automatically modified. In order to manage these access control requirements, Firewalls have been proposed by several researchers. However, many of the problems of traditional firewalls are aggravated due to these networks particularities.

In this paper we deeply analyze the local consistency problem in firewall rule sets, with special focus on automatic frequent rule set updates, which is the case of the dynamic nature of next generation networks. We propose a rule order independent local inconsistency detection algorithm to prevent automatic rule updates that can cause inconsistencies. The proposed algorithms have very low computational complexity as experimental results will show, and can be used in real time environments.

1. Introduction

In the next generation of communication networks, there will be a need for the quick deployment of independent mobile users. An ad hoc network is a network connection method which is most often associated with wireless devices. A wireless ad hoc network is a collection of autonomous nodes that communicate with each other by forming a multihop radio network and maintaining connectivity in a decentralized manner. Each node in a wireless ad hoc network functions as both a host and a router, and the control of the network is distributed among the nodes. In next generation networks, the network topology is in general dynamic, because the connectivity among the nodes may vary with time due to node departures, new node arrivals, and the possibility of having mobile nodes.

In next generation networks, authentication and access control between nodes is very important. However, prior and after the authentication step, there are attacks that can be performed with the aim of degrading network performance, for example to exploit services that should be available. In traditional networks, firewalls reduce the impact of these attacks, since they enforce at network perimeters an access control policy. However, as the concept of perimeter is not well defined in next generation networks, the firewall concept must be adapted [17, 18, 2].

A firewall is a network element that controls the traversal of packets across different network segments [1], thus it is a mechanism to enforce an access control policy, represented as an Access Control List (ACL). An ACL is in general a list of linearly ordered (total order) condition/action rules. The *condition* part of the rule is a set of condition attributes or selectors, where k is the number of selectors. The condition set is typically composed of five elements, which correspond to five fields of a packet header. Rule sets are usually composed of a number of rules ranging from tens to five thousand [3].

One of the main problems when adapting the concept of firewalls to next generation networks is that in general, nodes will have very limited resources (memory, processing power, and bandwidth), but a complete firewall software must be executed at each node. In addition, problems regarding matching algorithm complexity [3], rule set consistency [4, 5, 6], and rule set conformity [7] not only also exist, but are aggravated due to the particularities of the environment.

As can be seen from the example in Table 1, selectors of rules may overlap. There is a local inconsistency when two or more rules of the same rule set which have different actions overlap, since a packet can be matched with any of the overlapping rules. In next generation networks, when a rule is inserted,

Table 1. Example rule set

Priority/ID	Protocol	Source IP	Src Port	Destination IP	Dst Port	Action
R1	tcp	192.168.1.5	any	*.*.*.*	80	deny
R2	tcp	192.168.1.*	any	*.*.*.*	80	allow
R3	tcp	*.*.*.*	any	172.0.1.10	80	allow
R4	tcp	192.168.1.*	any	172.0.1.10	80	deny
R5	tcp	192.168.1.60	any	*.*.*.*	21	deny
R6	tcp	192.168.1.*	any	*.*.*.*	21	allow
R7	tcp	192.168.1.*	any	172.0.1.10	21	allow
R8	tcp	*.*.*.*	any	*.*.*.*	any	deny
R9	udp	192.168.1.*	any	172.0.1.10	53	allow
R10	udp	*.*.*.*	any	172.0.1.10	53	allow
R11	udp	192.168.2.*	any	172.0.2.*	any	allow
R12	udp	*.*.*.*	any	*.*.*.*	any	deny

removed or modified in a rule set, it can produce a local inconsistency. An inconsistent rule set may accept traffic that should be denied or vice versa, causing an important security problem.

In this paper we formally define what is a local inconsistency between an arbitrary number of rules in a firewall rule set. We have deeply analyzed the local consistency problem in firewall rule sets, with special focus on automatic rule insertions, removal and modifications. We propose a fast polynomial local inconsistency detection algorithm in firewall rule sets to prevent rule updates that can cause inconsistencies. The algorithm is in $O(n \cdot m)$ time complexity in the average case with the number of rules in the rule set, n, and with the number of inserted or modified rules, m, and in $O(m+n)$ space complexity. The proposed algorithm is capable of handling full ranges in all selectors. A Java tool has been implemented and it is available under request. To the best of our knowledge, this is the first time a polynomial local inconsistency detection algorithm for rule set updates has been proposed and tested with success in resource constrained devices. Note that inconsistency detection is only a part of the inconsistency management problem, where inconsistencies should be automatically detected, identified, characterized and possibly repaired [6]. In this paper we only focus in fast algorithms for inconsistency detection.

This paper is structured as follows. In Section 2 the special needs for firewalls in next generation networks are explained and related works are presented. In Section 3 firewall rule set local consistency management is explained, the problem is dissected and a solution is proposed for different rule set operations (insertion, removal, and modification). Section 4 presents the algorithms and a theoretical complexity analysis. Section 5 presents the experimental results of the algorithms that prove its feasibility in resource constrained devices. Finally, Section 6 presents the concluding remarks and some insights for future work.

2. Firewalls in Next Generation Networks. Related Works

There are two major types of next generation networks.

- Mobile Ad hoc Networks (MANet). A MANet is an autonomous collection of mobile nodes that communicate over relatively bandwidth constrained wireless links. Since the nodes are mobile, the network topology may change quickly and unpredictably over time. The network is decentralized, where all network activity, including discovering the topology and delivering messages must be executed by the nodes themselves, i.e., routing functionality will be incorporated into mobile nodes.
- Wireless Sensor Networks (WSN). A wireless ad hoc sensor network consists of a number of sensors spread across a geographical area. Each sensor has wireless communication capability and some level of intelligence for signal processing and networking of the data.

2.1. Firewalls in next generation networks

Filtering is a very important issue in any kind of next generation network, and especially in MANet due to its highly dynamic nature. The typical scenario results when a node enters or leaves a network. In this case, as nodes should or not communicate with the new one, the access control policy of all nodes of the network must be updated accordingly.

In traditional networks, firewalls are deployed in the perimeter routers that connect networks with different access control requirements (and not in

systems themselves). However, in next generation networks, filtering must me implemented at each node of the network. There are several important problems associated to the design and deployment of firewalls in next generation networks. Some problems are presented bellow, but a more extensive analysis was presented in [18]:

- Real time frequent rule set updates. As next generation networks and especially MANet are highly dynamic, rule set modification may be a frequent task performed in real time. As with traditional firewalls rule modification could cause inconsistencies in the rule set. However, in next generation networks, these inconsistencies must be detected and managed very fast.
- The nodes of next generation networks usually are battery powered embedded devices (cellular phones, cameras, PDA, smart cards, laptops, etc.) The problem of CPU and memory consumption is of extreme importance.

2.2. Related works

The closest works to ours come from the firewall diagnosis field. The seminal paper presented by Hari [11] identified this topic. They base their proposal on the assumption that there could not be a partial overlap between any selectors, which is not true for firewall rule sets. The proposed diagnosis algorithm is $O(n)$ for rules with two selectors. Authors explain that the application of their algorithm to rules with more selectors could exceed computational and memory limits of most systems. Eppstein [12] proposed a new definition of inconsistency based on rule priorities, and provided worst case $O(n^{1.5})$ algorithm to diagnose inconsistencies in rules with only two selectors (again, this is not the case of firewalls). Baboescu et al. improved both works [13] and provided diagnosis algorithms to diagnose inconsistencies in router filters that are 40 times faster than $O(n^2)$ ones for the general case of k selectors. However, they preprocess the rule set and convert integers and ranges to prefixes. This pre-process technique imposes the implicit assumption that a range can only express a single interval, which is true for all firewall selectors that can express ranges in the most common used firewall languages [14]. However, the range to prefix conversion technique could need to split a range in several prefixes (for a range $[0, 2^i]$, it could be needed 2^i prefixes in the worst case), and thus the final number of rules could increase over the original rule set [15] (which is not good for resource constrained devices). Recently, we proposed a novel algorithm for k selectors [6] based on a theoretical analysis of the problem. The algorithms are worst case $O(n^2)$ time complexity and $O(n)$ space complexity with the number of rules in the rule set. It is capable of handling full ranges in all selectors without any pre-process. However, none of these works propose real time rule set modification algorithms.

Other researchers complemented the diagnosis process with a characterization of the faults against an established taxonomy (inconsistencies are called anomalies in these works). Al-Shaer et al. [4] provided an order-dependent diagnosis plus characterization of different kinds of inconsistencies by pairs of rules and provide an algorithm. They use rule decorrelation techniques as a pre-process in order to decompose the rule set in a new one with non overlapping rules. Since the proposed rule decorrelation algorithms [8] have worst case exponential time and space complexity, the worst case complexity of their process is also exponential. Their proposal also contains an algorithm to insert, remove and modify rules in a consistent way. As this algorithm call the previous one, worst case time complexity is also exponential. A modification to their algorithm was provided by García-Alfaro et al. [5], where they integrate the decorrelation and consistency diagnosis plus characterization algorithms of Al-Shaer, and generate a decorrelated and consistent rule set. Due to the use of the same decorrelation techniques, this proposal has the same complexity of Al-Shaer one. Others have tried to address the consistency problem using OBDDs [9]. However, complexity of OBDDs depends on the optimal ordering of its nodes which is a NP-Complete problem [10]. Due to the high computational complexity, none of these algorithms are suitable for resource constrained devices.

3. Firewall Rule Set Consistency Management in Updates

The consistency management process can be divided in three sequential phases [6]: detection (finding the rules that are inconsistent with other rules), identification (finding the rules that cause all the inconsistencies among the detected inconsistent rules), and characterization (naming the identified inconsistent rules among a taxonomy of faults using rule relations) of inconsistent rules. The combination of detection and identification is the diagnosis.

In this paper, we are only interested in the detection of possible inconsistencies that can be caused when a rule or a collection of rules is updated in a consistent rule set. Note that identification is a naïve process in

this case, as all inconsistencies are caused by the newly inserted rule.

3.1. One to one local consistency

In [6] we showed that all inconsistencies characterized in the bibliography [16] (Fig. 1) can be detected with a single higher level one to one inconsistency definition. We showed that all characterized inconsistencies are special cases of correlation. So, the correlation inconsistency can be redefined as the superset of all inconsistencies, representing the most general case.We propose to use that one to one inconsistency definition (Definition 3.1), which ignores identification and characterization in order to simplify the detection process.

Definition 3.1. Local inconsistency. Two rules R_x, R_y from a rule set *RS* are *locally inconsistent* if and only if the intersection of *each of all* of its selectors *R[k]*, $\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$ is not empty, and they have different actions, *independently of their priorities*. The inconsistency between two rules expresses the *possibility* of a non desirable effect in the *semantics* of the rule set.

$$Inconsistent(R_i, R_j, RS), 1 \le i, j \le n, i \ne j \Leftrightarrow$$

$$R_x[k] \cap R_y[k] \ne \varnothing \wedge R_x[Action] \ne R_y[Action]$$

$$\forall k \in \{protocol, src\_ip, src\_prt, dst\_ip, dst\_prt\}$$

Attending to Definition 3.1, all characterized cases are of the same kind, and are called *inconsistencies* without any particular characterization. Note that, attending to Defintion 3.1, rule priority is not needed for inconsistency detection, and is only needed if inconsistencies are going to be characterized.

3.2. One to many local consistency

With the given definition, some questions may arise about how two or more rules can relate themselves in order to cause an inconsistency. We showed that all inconsistencies between pairs of rules can be detected by pairs of two with Definition 3.1, but more complicated situations must also be analyzed in order to illustrate this definition. In this section we show that no extension is needed to Definition 3.1, since the case of *one to many* rule inconsistency can be decomposed in several independent two-rule inconsistencies.

All base situations are presented in Fig. 1. This figure is a simplification to three inconsistent rules, but can easily be extended to more rules that can be composed in several ways.

Fig. 1(a1) represents an inconsistency where the union of two independent rules R_x, R_y overlaps with another one, R_z. As R_x is inconsistent with R_z, and R_y is also inconsistent with R_z, both in an independent manner, this situation can be decomposed in two independent inconsistencies, and can easily be detected. Note that, although R_x and R_y overlap, they do not cause an inconsistency, since they have the same action. Fig. 1(a2) presents a similar situation, where R_x overlaps with the union of R_y and R_z. This situation is also decomposable in two independent inconsistencies: R_x inconsistent with R_y, and R_x with R_z. Take into consideration that, in order to detect inconsistencies, the priority of the rules is not necessary. The situations presented in Fig. 1(b1) and Fig. 1(b2) are the inverse of the two previous ones respect to the action. Thus, the detection process is analogous. Finally, Fig. 1(c) represents a relation with three overlapping rules. This situation can also be decomposed in two independent ones: R_x inconsistent with R_y, and R_y with R_z.

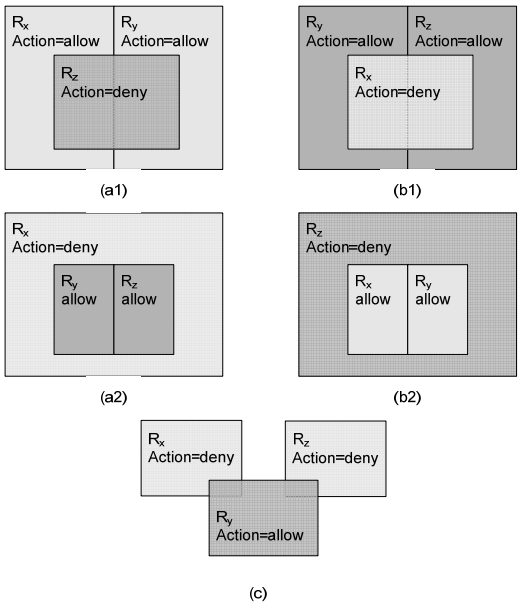

Figure 1. Graphical diagram of inconsistencies between three rules

In conclusion, it is possible to detect inconsistencies between an arbitrary number of rules with Definition 3.1, because all the presented situations can be decomposed in independent two by two relations. These examples are easily extendable to more than three rules.

3.3. Rule set update operations

There are three basic update operations: insertion, removal or modification of one or more rules. Insert operation usually occur when a new node enters the networks, removal when a node leaves the network, and modifications when the access control requirements between nodes change. Each of these three operations needs an analysis in order to know if they can cause an inconsistency. In this paper, consistency of the node rule set is maintained (1) not allowing the insertion of an inconsistent group of rules and (2) only inserting rules that are consistent with the node rule set. Thus, the algorithm only inserts rules that guarantee that the resulting rule set is consistent. Rule set distribution and rule deployment is considered in this paper, and it is a topic for future research.

3.3.1. Rule insertion. One or more rules can be inserted at the same time in a rule set, for example when one or more nodes enter the network. We assume in this paper that the target rule set is consistent. Initial consistency diagnosis can be run offline, as it is understood that this case only happens prior to the first deployment of the rule set. Several algorithms to diagnose inconsistencies in rule sets have been reviewed in the related works section.

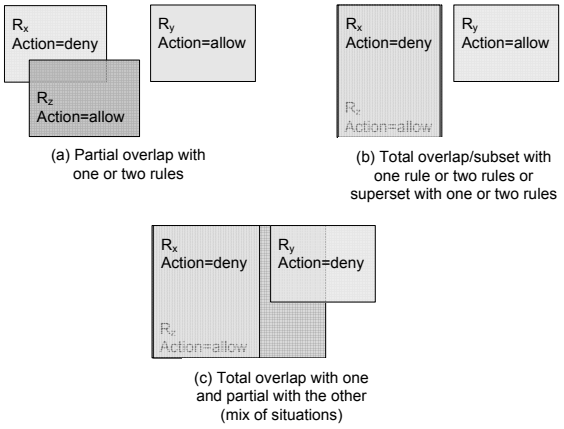

Figure 2. Graphical diagram a new inconsistent rule introduced in a consistent rule set

If only one rule is inserted, then it could be consistent with all rules of the rule set, or may be inconsistent with one or many of them. As has been explained in the previous subsection, this case can be reduced to checking the consistency between pairs of rules. These pairs of rules will always be composed of the new rule to be inserted and each of the rules in the rule set. Note that, as Defintion 3.1 is order independent, the new rule can be inserted anywhere in the rule set. Fig. 2 presents an example where a new rule R_z is going to be added to a consistent rule set consisting of two rules. More complex cases can be derived from this one.

Fig. 2(a) represents a partial overlap between the new rule, R_z, and one of the independent rules, R_x. Note that this case can be extended to a partial overlap with both R_x and R_y, deriving in the situation of Fig. 1(c). Fig. 2(b) represents the case of a total overlap between the new rule R_z, and one of the independent ones, R_x. This case can be extended to a subset overlap of R_z in R_x, and also to a superset overlap (it is the inverse). Fig. 2(c) represents an example of a combined case of the two previous situations, where the new rule R_z totally overlaps with R_x and partially with R_y. This situation can also be extended to fully overlap with R_x and R_y, or even a subset or superset of both.

If more than one rule is being inserted, then there are three possible approximations. Note that the relative order between the collection of rules being inserted and their final position on the rule set to be inserted in is not relevant, since inconsistencies are order independent.

- The first one is to insert all rules and then check the consistency of the full rule set with an on-line algorithm. To the best of our knowledge, the fastest algorithm to solve the inconsistency diagnosis problem [6] is worst case $O(n^2)$ time complexity with the number of rules in the rule set. However, for big rule sets the consistency diagnosis of the full rule set could be unfeasible in time and space in resource constrained devices.
- The second one is to iterate over the collection of rules to be inserted and check the consistency of the rule set for each insertion operation, inserting it if consistent. This case is the same as the one explained above but repeated as many times as rules are in the collection. In addition, if the collection is inconsistent, then it is not detected until two inconsistent rules of the collection are inserted in the rule set.
- The third one is to check the consistency of the collection of rules being inserted using one of the reviewed offline algorithms and then use the second approximation. In next generation networks, the collection of new rules may be in general very small, since these networks (especially MANet) suffer from small but frequent topology updates. In this case, the $O(n^2)$ consistency diagnosis algorithm [6] can be applied

to the collection of rules being inserted. If they are consistent, then they can be added to the rule set using the second approximation; if not, the process is stopped here. This approximation is fastest than the first one and more secure than the second one, since if the collection of new rules are inconsistent, no rule is going to be added to the rule set. This approximation is the one that is going to be used in the algorithms presented in this paper.

3.3.2. Rule removal. One or more rules can be removed at the same time in a rule set, for example when one or more nodes leave the network. If the rule set has been deployed in a node, then it is consistent by definition, as has been explained in the previous subsection. Note that, contrary to the insertion operation, the cases of removing one or more rules are the same with regard to consistency, since the rules to be removed are by definition, consistent (they are part of a consistent rule set). No special care must be taken.

3.3.3. Rule modification. Rule modification is a similar operation to a sequential removal and insertion, and represents the case when access control requirements change, or when a node leaves the network and another enters it. Since rule removal cannot cause any inconsistency, then this case is reduced to rule insertion with regard to consistency. If several rules are going to be modified, this can again be reduced to the removal of several rules and the insertion of the new ones. Consistency checking is necessary in the new collection of rules being updated.

4. Local Inconsistency Detection Algorithms

As was explained below, upon network creation nodes must have a rule set that implement the initial access control requirements. However, in order be as secure as possible, this rule sets must be consistent and, if possible, not redundant. The removal of redundancies is not required, since redundancies can be used to improve the performance of matching algorithms. Any of the reviewed offline algorithms like [4, 5, 6] can be used in this step in order to diagnose inconsistencies prior to first deployment. In this section, two polynomial algorithms to detect inconsistencies in rule sets upon rule insertion are presented; the second one is an improvement in computational complexity over the first one. These two algorithms implement Definition 3.1 and the process explained in Section 3.3. Recall Note that in this paper, consistency of the node rule set is maintained (1) not allowing the insertion of an inconsistent group of rules and (2) only inserting rules that are consistent with the node rule set. Thus, the algorithm only inserts rules that guarantee that the resulting rule set is consistent.

4.1. Local inconsistency detection algorithm

The algorithm presented in Fig. 3 receives a firewall rule set and one or more rules to be inserted. The result of the algorithm is a boolean matrix called *Inconsistency Matrix* (Definition 4.1).

Definition 4.1. Inconsistency Matrix, IM. It is a matrix of size $n \cdot m$, where n is the number of rules in the rule set of the node, and m is the size of the collection of rules being inserted. Upon instantiation, the matrix elements are set to false. An element *im[n,m]* is set to *true* when an inconsistency is detected between the rules in the row n and column m.

Firstly, if there is more than one rule to be inserted, consistency of the collection is checked. If the collection is inconsistent, the algorithm finishes with no rules inserted. The consistency diagnosis algorithm used in this first step has been taken from [6], and returns a *boolean* indicating if the collection of rules to be inserted is consistent or not. Then, the algorithm enters the main loop. For each rule in the collection (if there are more than one rule), it checks if there is an inconsistency with any of the rule set. Note that the detection process, like Definition 3.1, is order independent. If there is no inconsistency with the rule considered in that iteration, it is inserted. However, if there is inconsistency, the corresponding element of the IM is set to *true* and the rule is not inserted. The algorithm finishes with a consistent rule set, because if inconsistent rules are detected, they are not inserted. Inconsistencies are given to the user in the Inconsistency Matrix.

Time complexity of Algorithm 1 is bounded by the diagnosis algorithm of line 7 applied to the collection of rules being inserted, $O(m^2)$, and the two nested loops of lines 9 and 11. Note that the general case in next generation networks is $m << n$. Complexity of the algorithm depends on the size of the list that contains the rules to be inserted (which will usually be very small in next generation networks). The *inconsistency()* operation of line 23 (called in line 13) implements Definition 3.1. It is composed of an iteration. In typical firewall rule sets the iteration runs 5 times. Anyway, the iteration is bounded by the number of selectors, which is a constant, k. In addition, inside the iteration there is an intersection between each selector. The typical 5 selectors of firewall rule

Algorithm 1. Inconsistency Detection

```
1  Func detection(in List: ruleSet, List: insertCollection; out
2  Array[][]: im)
3  Var
4     Rule ri, rj
5  Alg
6     im=new Boolean[ruleSet.size()][insertCollection.size()]
7     if (insertGroup.size()>1)
8        res = consistencyDiagnosis(insertCollection)
9     if (res) { // If consistent
10       for each j=1..insertCollection.size() {
11          rj=insertCollection.get(j)
12          for each i=1..ruleSet.size() { // Main loop
13             ri = ruleSet.get(i)
14             if (inconsistency(ri, rj))
15                im[ri.getID()][rj.getID()] = true
16             else
17                insertRule(ruleSet, insertCollection.get(j))
18          }
19       }
20    }
21 End Alg
22
23 // This function implements the Inconsistency Definition
24 Func inconsistency(in Rule: rx, ry; out Boolean: b)
25 Var
26    Integer i
27 Alg
28    b = false
29    if (rx.action != ry.action) {
30       b = true
31       i = 1
32       while (i<=x.selectors.size() AND b)
33          b = b AND intersection(rx.getSelector(i),
34 ry.getSelector(i))
35    }
36 End Alg
```

Figure 3. Inconsistency Detection for rule insertion algorithm

Algorithm 2. Improved Inconsistency Detection

```
1  Func detection(in List: ruleSetAllow, List: ruleSetDeny ,
2  List: insertCollection; out Array[][]: im)
3  Var
4     Rule ri, rj
5  Alg
6    im=new Boolean[ruleSet.size()][insertCollection.size()]
7    if (insertCollection.size()>1)
8       res = consistencyDiagnosis(insertCollection)
9    if (res) {
10      for each j=1..insertCollection.size() { // Main loop
11            rj=insertCollection.get(j)
12            if (rj.action == allow) {
13               for each i=1..ruleSetDeny.size() {
14                  ri = ruleSetDeny.get(i)
15                  if (inconsistency(ri, rj))
16                     im[ri.getID()][rj.getID()] = true
17                  else
18                   insertRule(ruleSet, insertCollection.get(j))
19               }
20            }
21            else { // Check with allow rule set
22               for each i=1..ruleSetAllow.size() {
23                  ri = ruleSetAllow.get(i)
24                  if (inconsistency(ri, rj))
25                    im[ri.getPriority()][rj.getPriority()]= true
26                  else
27                   insertRule(ruleSet, insertCollection.get(j))
28               }
29            }
30       }
31    }
32 End Alg
33
34 // This function is the same than in Algorithm 1
35 Func inconsistency(in Rule: rx, ry; out Boolean: b)
```

Figure 4. Improved Inconsistency Detection for rule insertion algorithm

sets are source and destination IP, source and destination ports and protocol. All these selectors are integers or ranges of them except IP address. Knowing if two ranges of integers intersect can be done in constant time with a naïve algorithm which compares the limits of the intervals. Knowing if two IP addresses intersect can also be easily done in constant time by comparing their network addresses by using their netmasks.

Thus, the best case for the algorithm is achieved when only one rule is going to be inserted, $m=1$. In this case, time complexity is in $O(n)$. Worst case is achieved when the number of rules to be inserted is very near to the number of rules in the rule set. In this case time complexity is very *near to* $O(n^2)$. However, note that the worst case is very rare in next generation networks. Average case is in $O(m^2+n \cdot m)$, $m<<n \rightarrow O(n \cdot m)$. Complexity also depends on the selectors of the rules being inserted, as is derived from the implementation of the *inconsistency()* operation (it uses lazy evaluation). Worst case complexity is achieved if rules are too open (ranges in selectors or even wildcards), which is not the usual case in next generation networks (as was explained in the first two sections of the paper).

Space complexity of Algorithm 1 is bounded by the size of the rule set and the number of rules being inserted, which in the best case is $O(n)$, in the worst case is *close to* $O(n^2)$, and in the average case is $O(n+m)$.

4.2. Improved algorithm

An optimization can be introduced in Algorithm 1 in order to improve performance. Note that Definition 3.1 is based on the fact that two rules can only be inconsistent if they have different actions. To improve the algorithm, it is possible to divide the original rule set in two, one with *allow* rules and the other with *deny* rules. This modification can be done because the possible inconsistencies are independent of rule priority (Definition 3.1) The modifications needed over Algorithm 1 are presented in Fig. 4 (lines 9-30, the main loop). Algorithm 2 receives three lists, one with *allow* rules, other with *deny* rules, and finally the list of rules to be inserted. The division of the original rule set in two can be done while parsing with no additional cost. As in Algorithm 1, the first operation is to check the consistency of the collection. Then, and here is the difference, for each rule in the collection (if there are more than one rule), it checks its action. If it is *allow*, the rule is checked for inconsistency with all of the *deny* rule set; if it is *deny*, it is checked for consistency with all *allow* rules. The rest of the algorithm is equal to Algorithm 1.

Time complexity of Algorithm 2 (Table 2) is again bounded by the diagnosis algorithm of line 7 applied to the collection of rules being inserted, $O(m^2)$, and the two nested loops of lines 9 and 12 or 21.

With the improvement, the best case for the algorithm is achieved when only one rule is going to be inserted, $m=1$, and there are no rules in the opposite action rule set. In this case, time complexity is a *constant.* Worst case is achieved when the number of rules to be inserted is very near to the number of rules in the rule set, and most of them have the opposite action of the smaller rule set. In this case time complexity is very *near to* $O(n^2)$ and there is no real improvement over Algorithm 1. However, note that the worst case is very rare in next generation networks. Average case is in $O(m^2+n/2 \cdot m)$, $m<<n \rightarrow O(n/2 \cdot m)$. However, as in next generation networks most rules will be *allow* and there will be few exceptions (*deny* rules), experimental results should show a huge improvement in real life scenarios. As in Algorithm 1, complexity also depends on the selectors of the rules being inserted, as is derived from the implementation of the *inconsistency()* operation (it uses lazy evaluation). Space complexity of Algorithm 2 is equal to Algorithm 1. Thus, in most real life scenarios the improvement should show a huge reduction in running time over Algorithm 1.

Recall that rule removal does not need a special algorithm, since it cannot cause inconsistencies. In addition, rule update is equivalent to sequential rule removal and insertion in terms of the possibility of causing inconsistencies.

Table 2. Computational complexity of Algorithm 2

Number of inserted rules	Best case	Worst case	Average case
1	*constant*	$O(n)$	$O(n/2)$
m	$O(m)$	*near* $O(n^2)$	$O(n/2 \cdot m)$

5. Experimental Results

As real rule sets have been used in the experiments, results represent an average case in the number of deny and allow rules for each rule set. Table 3 presents the results of the conducted tests in one node of the network. The first column represents the size of the rule set. The second and third ones represent the running time of Algorithm 1 with only one rule to be inserted. The average case (AC) is achieved when the rule to be inserted is an arbitrary rule (in experiments, a rule with *allow* action). The worst case (WC) is achieved when the rule to be inserted is the wildcard rule *deny all.* Fourth and fifth columns represent the same but for Algorithm 2. Note that the only difference in the presented average and worst case is a multiplicative constant, as the type of the rule only changes the time needed to execute the *inconsistency()* method in both algorithms by a constant factor.

If more rules are going to be inserted, it is necessary to multiply the desired results by the number of rules, and add the needed time for the initial diagnosis of the collection of rules. In order to test the feasibility of this diagnosis, we have used the fastest known heuristic

Table 3. Experimental results

Rule Set size	%Deny rules	Algorithm 1 AC (ms)	Algorithm 1 WC (ms)	Algorithm 2 AC (ms)	Algorithm 2 WC (ms)
50	28,21	2	4	1	3
144	30,91	9	13	3	11
238	66,43	16	19	9	14
450	34,73	27	41	11	33
900	14,8	48	88	13	77

algorithm [6] and used the rule set of size 50, obtaining a result of 64ms, which is quite acceptable. However, in real life scenarios of next generation networks, insertions will be of few rules and time needed for the diagnosis of the new rules may be negligible. Experiments were performed on a Java implementation with SableVM Java Virtual Machine implementation 1.13, and on an isolated machine with Intel XScale IXP420@266MHz processor with 32Mb of RAM, running Debian Etch GNU/Linux Distribution. Each test was conducted 5000 times in order to get accurate results (the presented results are the average). Note that SableVM is not a just in time virtual machine (JIT VM). If a JIT capable VM is used, results could improve by a factor or 10 or more, depending on the particular JIT implementation.

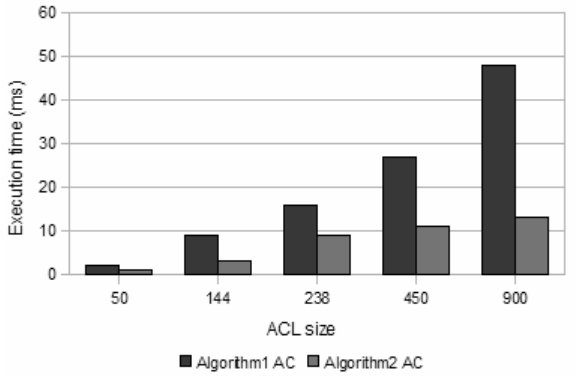

Figure 5(a). Running time of the algorithms. Average case

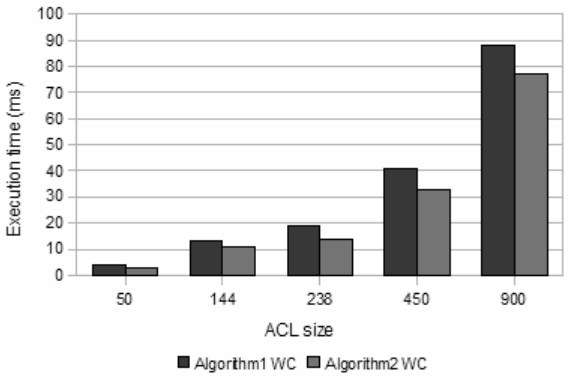

Figure 5(b). Running time of the algorithms. Worst case

The first thing that should be noted is the difference between the two algorithms in the insertion of one rule (in average and worst cases). As expected, there is a huge reduction in the execution time of the average case in a real life scenario for all rule set sizes (Fig. 5(a)). Even for very large rule sets in next generation networks (900 rules) execution time is about 13ms with the improved algorithm (a 5x speedup). This time is very reasonable taking into account that next generation networks may have rule sets smaller than 900 rules. However, if a very conflictive rule is going to be added, execution time increases a lot (Fig. 5(b)). With 900 rules, it can be eight times slower (Fig. 6). Even in this case, the execution time is very reasonable, about 80ms). Note how in Fig. 5(b) the difference between worst cases of Algorithms 1 and 2 is not as big as was for the average case. The reason is that the used rule for insertion *(deny all)* will cause an inconsistency with most rules of the rule set (in Algorithm 2 the rule is compared with the bigger of the two lists, the one that contains *allow* rules).

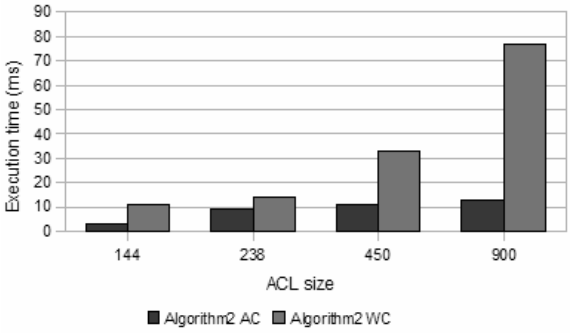

Figure 6. Algorithm 2 running time

In conclusion, theoretical and empirical results show that the algorithms can be used in resource constrained devices where a quick response time is needed, as a complement for actual distributed and ubiquitous firewall proposals.

6. Conclusions and Future Works

This paper proposes a real time approach to detect inconsistencies in firewall rule sets when inserting, removing or modifying its rules. We showed that filtering is a very important topic in next generation networks. We showed that, to the best of our knowledge, no algorithm to date is applicable in real time in resource constrained devices. For these reasons, firewall problems must be revisited. We analyzed the consistency diagnosis problem in firewall rule sets when inserting, removing and modifying the rule set, and showed that inconsistencies can only be caused in rule insertions. This analysis is based on a previous more general theoretical one for the general problem of firewall consistency management. Due to the division of the consistency management process in three steps, we realized that local inconsistency

detection does not depend on rule priorities, identification is a naïve step and characterization is not necessary for rule modifications. Inconsistency detection is only a part of the inconsistency management problem, where inconsistencies should be automatically detected, identified, characterized and possibly repaired [6]. In this paper we only focus in fast algorithms for inconsistency detection.

We explained that inconsistencies can only be caused by rule insertions, proposed three approximations to the solution and selected one for implementation, giving two algorithms. In this paper, consistency of the node rule set is maintained (1) not allowing the insertion of an inconsistent group of rules and (2) only inserting rules that are consistent with the node rule set.

The average case time complexity of the process in $O(n \cdot m)$ with the number of rules in the rule set, n, and with the number of updated rules, m. Space complexity is in $O(n/2+m)$. The proposed process is capable of handling full ranges in all selectors without pre-processing them. The algorithms must be run at each node of the topology which needs a modification of its rule set. We provide experimental results using resource constrained devices. Results were provided. A Java tool has been implemented and it is available under request. To the best of our knowledge, this is the first time a polynomial algorithm has been proposed to automatically address this problem in resource constrained devices. This work represents a complement to the distributed and ubiquitous firewall proposals.

However, our approach has some limitations that give us opportunities for improvement in future works. The most important one is that our process can detect inconsistent rules but not redundancies Although redundancies do not change the semantics of the rule set, they can increase its size.

References

[1] D. Chapman and E. Zwicky. Building Internet Firewalls, Second Edition, O'Reilly & Associates, Inc., 2000.

[2] Nicolas Prigent and Christophe Bidan. "Securing Devices Communities in Spontaneous Networks." International Scientific Journal of Computing, Vol. 4 Issue 2, 2005.

[3] David E. Taylor. "Survey and taxonomy of packet classification techniques." ACM Computing Surveys, Vol. 37, No. 3, 2005. Pages 238 – 275.

[4] E. Al-Shaer, Hazem H. Hamed. "Modeling and Management of Firewall Policies". IEEE eTransactions on Network and Service Management (eTNSM) Vol.1, No.1, 2004.

[5] J. García-Alfaro, N. Boulahia-Cuppens, F. Cuppens. "Complete Analysis of Configuration Rules to Guarantee Reliable Network Security Policies." Springer-Verlag International Journal of Information Security (Online) (2007) 1615-5262.

[6] S. Pozo, R. Ceballos, R.M. Gasca. "Improving Computational Complexity of the Inconsistency Characterization Problem in Firewall Rule Sets". International Conference on Security and Cryptography (SECRYPT). Porto, Portugal. INSTICC Press, 2008.

[7] S. Pozo, R. Ceballos, R. M. Gasca. "CSP-based Rule Set Diagnosis using Security Policies." International Symposium on Frontiers in Availability, Reliability and Security (FARES), in International Conference on Availability, Reliability and Security (ARES), Vienna, Austria. IEEE Computer Society Press, April 2007.

[8] M. Condell, L. Sánchez. "On the Deterministic Enforcement of Unordered Security Policies." BBN Technical Memorandum No. 1346, April 2004.

[9] L. Yuan, J. Mai, Z. Su, H. Chen, C. Chuah, P. Mohapatra. "FIREMAN: A Toolkit for FIREwall Modelling and Analysis." IEEE Symposium on Security and Privacy (S&P). Oakland, CA, USA. May 2006.

[10] B. Bollig, I. Wegener. "Improving the Variable Ordering of OBDDs is NP-Complete". IEEE Transactions on Computers, Vol.45 No.9, September 1996.

[11] B. Hari, S. Suri, G. Parulkar. "Detecting and Resolving Packet Filter Conflicts." Proceedings of IEEE INFOCOM, March 2000.

[12] D. Eppstein, S. Muthukrishnan. "Internet Packet Filter Management and Rectangle Geometry." Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), January 2001.

[13] F. Baboescu, G. Varguese. "Fast and Scalable Conflict Detection for Packet Classifiers." Elsevier Computers Networks (42-6) (2003) 717-735.

[14] S. Pozo, R. Ceballos, R.M. Gasca. "AFPL, An Abstract Language Model for Firewall ACLs". 8th International Conference on Computational Science and Its Applications (ICCSA). Perugia, Italy. Springer-Verlag, 2008.

[15] V. Srinivasan, G. Varguese, S. Suri, M. Waldvogel. "Fast and Scalable Layer Four Switching." Proceedings of the ACM SIGCOMM conference on Applications, Technologies, Architectures and Protocols for Computer Communication, Vancouver, British Columbia, Canada, ACM Press, 1998.

[16] H. Hamed, E. Al-Shaer "Taxonomy of Conflicts in Network Security Policies." IEEE Communications Magazine Vol.44, No.3, 2006.

[17] S. M. Bellovin, "Distributed Firewalls." ;login Magazine:, November 1999, pp. 39-47.

[18] R. Fantacci, L. Maccari, P. Neira, R. M. Gasca. "Efficient Packet Filtering in Wireless Ad Hoc Networks." IEEE Communications Magazine Vol.46, No.2, 2008.

Acknowledgements

This work has been partially funded by Spanish *Ministry of Science and Education project* under grant DPI2006-15476-C02-01, and by FEDER (under ERDF Program).