# Firewall Modules and Modular Firewalls

H. B. Acharya
University of Texas at Austin
acharya@cs.utexas.edu

Aditya Joshi
University of Texas at Austin
adityaj@cs.utexas.edu

M. G. Gouda
National Science Foundation
mgouda@nsf.gov

***Abstract*—A firewall is a packet filter placed at an entry point of a network in the Internet. Each packet that goes through this entry point is checked by the firewall to determine whether to accept or discard the packet. The firewall makes this determination based on a specified sequence of overlapping rules. The firewall uses the first-match criterion to determine which rule in the sequence should be applied to which packet. Thus, to compute the set of packets to which a rule is applied, the firewall designer needs to consider all the rules that precede this rule in the sequence. This "rule dependency" complicates the task of designing firewalls (especially those with thousands of rules), and makes firewalls hard to understand. In this paper, we present a metric, called the dependency metric, for measuring the complexity of firewalls. This metric, though accurate, does not seem to suggest ways to design firewalls whose dependency metrics are small. Thus, we present another metric, called the inversion metric, and develop methods for designing firewalls with small inversion metrics. We show that the dependency metric and the inversion metric are correlated for some classes of firewalls. So by aiming to design firewalls with small inversion metrics, the designer may end up with firewalls whose dependency metrics are small as well. We present a method for designing modular firewalls whose inversion metrics are very small. Each modular firewall consists of several components, called firewall modules. The inversion metric of each firewall module is very small - in fact, 1 or 2. Thus, we conclude that modular firewalls are easy to design and easy to understand.**

## I. Introduction

A firewall is a packet filter that is placed at an entry point of a network in the Internet. The function of a firewall is to check each packet that goes through the entry point (at which the firewall is located) and determine whether to accept the packet and allow it to proceed on its way or to discard the packet.

The firewall perform its function based on a specified sequence of rules. Each rule is of the form

$$< predicate > \rightarrow < decision >$$

where $< predicate >$ is a function that assigns to each packet a boolean value, true or false, and $< decision >$ is either "accept" or "discard". When a packet $p$ reaches a firewall $F$, $F$ performs two steps:

1) $F$ identifies the first rule $r$ (in its sequence of rules) whose $< predicate >$ assigns the value true to packet $p$.
2) If the $< decision >$ of rule $r$ is accept (or discard, respectively) then $F$ accepts (or discards, respectively) packet $p$.

Note that $F$ employs a "first-match" criterion to determine which rule (in its sequence of rules) should be applied to which packet. This first-match criterion allows the rules in the rule sequence to be "overlapping". This can be both advantageous and disadvantageous.

The advantage of making the rules in the rule sequence overlapping is that it reduces the number of rules in the rule sequence, sometimes dramatically.

The disadvantage of making the rules in the rule sequence overlapping is that it creates many dependencies between the rules in the rule sequence. This, in turn, complicates the task of designing and understanding the rule sequence. For instance, if the firewall designer needs to compute the set of packets to which a rule $r$ (in the rule sequence) applies, then the designer needs to consider not only rule $r$ but also all the rules that precede $r$ in the rule sequence.

In this paper, we introduce a metric, called the "dependency metric", that measures the complexity of firewalls. The more the value of the metric for a given firewall, the more complex the firewall is and the harder it is to design and understand.

Unfortunately, the dependency metric, though accurate, does not seem to suggest methods for designing firewalls for which the values of the metric are small. Thus, we introduce another complexity metric, called the "inversion metric", for measuring the complexity of firewalls.

We show, below, that the dependency metric and the inversion metric are correlated (at least for a rich class of firewalls called "uniform firewalls"). This result allows us to use the inversion metric as a good approximation of the dependency metric.

Then, we identify three classes of firewalls, namely "simple firewalls", "partitioned firewalls", and "modular firewalls", for which the values of the inversion metric are small. (This implies that these classes of firewalls are easier to design and understand.) We also describe methods for designing firewalls in these three classes.

Of particular interest is the class of modular firewalls. Each modular firewall consists of simple firewall components, called "firewall modules". The value of the inversion metric for each firewall module is 1 or 2. This causes the value of the inversion metric for the full firewall to be 1 or 2. (Note that the smallest possible value of the inversion metric is 1.)

We present an algorithm that takes as input any firewall $F$ whose inversion metric is large and computes as output an equivalent modular firewall $MF$ whose inversion metric is (by definition) 1 or 2. The complexity of this algorithm is $O(n^2)$

where $n$ is the number of rules in the input firewall $F$. The existence of this algorithm indicates that designing a modular firewall is not harder than designing an equivalent non-modular firewall. Our simulation results, reported below, show that the cost and performance of this algorithm are attractive.

## II. Fields, Packets, Rules, and Firewalls

In this section, we define the main terms in this paper - fields, packets, rules, and firewalls.

A field is a variable, whose value is taken from an interval of non-negative integers. Examples of fields are source IP address, destination IP address, transport protocol, source port number, and destination port number. The domain of values of the source IP address field, for example, is the interval $[0, 2^{32} - 1]$.

In this paper, we consider $d$ fields, denoted $f_1$, .. , and $f_d$. The domain of values of each field $f_j$, denoted $D(f_j)$, is an interval of non-negative integers.

A *packet* $p$ is a $d$-tuple $(p.f_1, .., p.f_d)$, where each $p.f_j$ is an element from the domain $D(f_j)$ of field $f_j$.

A *rule* $r$ is of the form:

$$f_1 \in R_1 \wedge ... \wedge f_d \in R_d \rightarrow < r.decision >$$

where each $R_j$ is a non-empty interval of non-negative integers taken from the domain $D(f_j)$ of field $f_j$, and the $< r.decision >$ is either accept or discard. A rule whose decision is accept is called an *accept rule*, and a rule whose decision is discard is called a *discard rule*.

A packet $(p.f_1, .., p.f_d)$ is said to *match* a rule $r$ of the form:

$$f_1 \in R_1 \wedge ... \wedge f_d \in R_d \rightarrow < r.decision >$$

iff the predicate $(p.f_1 \in R_1 \wedge ... \wedge p.f_d \in R_d)$ holds.

A rule of the form

$$f_1 \in D(f_1) \wedge ... \wedge f_d \in D(f_d) \rightarrow accept$$

is called an *accept-all rule*, and a rule of the form

$$f_1 \in D(f_1) \wedge ... \wedge f_d \in D(f_d) \rightarrow discard$$

is called a *discard-all* rule.

A *firewall* $F$ is a nonempty sequence of rules, where the last rule is either an accept-all rule or a discard all rule.

A packet $(p.f_1, .., p.f_d)$ is said to be *accepted* by a firewall $F$ iff $F$ has an accept rule $r$ such that the following two conditions hold.

1) $(p.f_1, .., p.f_d)$ matches $r$.
2) $(p.f_1, .., p.f_d)$ does not match any rule that precedes $r$ in $F$.

A packet $(p.f_1, .., p.f_d)$ is said to be *discarded* by a firewall $F$ iff $F$ has a discard rule $r$ such that the following two conditions hold.

1) $(p.f_1, .., p.f_d)$ matches $r$.
2) $(p.f_1, .., p.f_d)$ does not match any rule that precedes $r$ in $F$.

Because the last rule in a firewall is either an accept-all rule or a discard-all rule, it is straightforward to show that for every packet and every firewall $F$, either the packet is accepted by $F$ or the packet is discarded by $F$.

Two firewalls $F$ and $G$ are said to be *equivalent* iff $F$ and $G$ accept the same set of packets (and discard the same set of packets).

## III. The Dependency Metric of Firewalls

In this section we define a metric that can be used to measure the complexity of a firewall. If the value of this metric is large for one firewall, then this firewall is relatively ”hard to understand”. And if the value of this metric is small for another firewall, then this firewall is relatively ”easy to understand”. We refer to this metric as the dependency metric. But before we can define the dependency metric, we first need to introduce several definitions.

A *band* of a firewall $F$ is a maximal sequence of consecutive rules that have the same decision, whether accept or discard, in $F$. If (all) the rules in a band have accept decisions, then the band is called an *accept band*. Similarly, if (all) the rules in a band have discard decisions, then the band is called a *discard band*.

**Theorem 1.** *If the rules in a band in a firewall F are reordered in any way, then the resulting firewall is equivalent to F.*

*Proof:* Assume that the rules in a band $B$ in $F$ are reordered in any way. Let $p$ be a packet that is resolved by a rule $r$ in $B$ before the reorder. And assume that packet $p$ is resolved by another rule $s$ after the reorder. Thus, rule $s$ belongs to band $B$, and has moved ahead of rule $r$ as a result of the reorder.

Because both rules $r$ and $s$ belong to the same band $B$, they have the same decision. Therefore, rule $s$ will resolve packet $p$ after the reorder in the same way that rule $r$ has resolved packet $p$ before the reorder.

Every packet that is accepted before the reorder is also accepted after the reorder, and every packet that is discarded before the reorder is also discarded after the reorder. Hence the firewall that results from the reorder is equivalent to the original firewall $F$ before the reorder. ■

If all the rules in a firewall have the same decision, then this firewall consists of only one band. But such a firewall is not very useful in practice. Thus, from now on, we consider only firewalls that consist of two or more bands.

A packet $p$ is said to be resolved by a rule $r$ in a firewall $F$ iff the following two conditions hold:

1) $p$ matches rule $r$.
2) $p$ does not match any rule $s$, where $s$ precedes $r$ in $F$ and $r$ and $s$ occur in different bands in $F$.

The *dependency set* of a rule $r$ in a firewall $F$ is the set containing every rule $s$, where $s$ precedes $r$ in $F$, and $r$ and $s$ occur in different bands in $F$.

From the last two definitions, we conclude that to determine whether a packet $p$ is resolved by a rule $r$ in a firewall $F$, one needs to test packet $p$ against rule $r$ and against every rule in the dependency set of $r$. Clearly, the complexity of these tests are proportional to the number of rules in the dependency set of $r$. If the cardinality of the dependency set of $r$ is large, then determining whether a given packet is resolved by $r$ is relatively hard. And one can claim, in this case, that rule $r$ is hard to understand. On the other hand, if the cardinality of the dependency set of $r$ is small, then determining whether a given packet is resolved by $r$ is relatively easy. And one can claim, in this case, that rule $r$ is easy to understand.

It follows from this discussion that the complexity of understanding a rule $r$ in a firewall $F$ can be measured by the cardinality of the dependency set of $r$ in $F$. Therefore, the complexity of understanding firewall $F$ can be measured by the average cardinality of a dependency set of a rule in $F$.

The *dependency metric* of a firewall $F$ is the average cardinality of a dependency set of a rule in $F$.

**Theorem 2.** *Let $F$ be any firewall that has $n$ rules.*

1) *The smallest possible value of the dependency metric of $F$ is $\frac{(n-1)}{n}$.*
2) *The largest possible value of the dependency metric of $F$ is $\frac{(n-1)}{2}$.*

*Proof:*

1) The dependency metric of $F$ has its smallest value when $F$ consists of only two bands. The first band consists of the top $n-1$ rules in $F$, and the second band consists of the last rule in $F$. In this case, the dependency set of each one of the top $n-1$ rules is empty, and the dependency set of the last rule has $n-1$ rules. Thus, the average cardinality of a dependency set of a rule in $F$ is $\frac{n-1}{n}$.
2) The dependency metric of $F$ has its largest value when $F$ consists of $n$ bands. And each band consists of only one rule. In this case,
the dependency set of the first rule in $F$ has $0$ rules,
the dependency set of the second rule in $F$ has $1$ rule,
...,
the dependency set of the $n$-th rule in $F$ has $n-1$ rules.
Thus, the average cardinality of a dependency set of a rule in $F$ is $\frac{n-1}{2}$.

■

The problem of the dependency metric is that this metric does not seem to suggest methods for designing firewalls whose dependency metrics are small.

This problem compels us to look for another complexity metric of firewalls. This new complexity metric needs to satisfy two requirements. First, this new metric needs to be correlated to the dependency metric (at least for some classes of firewalls). Second, it should be easy to design firewalls for which the new metric has a small value. We present such a metric in the next section.

## IV. The Inversion Metric of Firewalls

In this section we introduce a second metric that can be used to measure the complexity of firewalls. We refer to this metric as the inversion metric. We show that the inversion metric satisfies two nice properties. First, we show, in this section, that the value of the inversion metric of a firewall is correlated to the value of the dependency metric of the same firewall (when the firewall is uniform). This result allows us to use the inversion metric as a good approximation of the dependency metric. Second, we demonstrate, in Section 7 below, that one can develop methods for designing firewalls whose inversion metrics are very small. In particular, we give an algorithm that takes as input any firewall, whose inversion metric value is large, and produces an equivalent firewall, whose inversion metric value is no more than 2, a small value.

The *inversion metric* of a firewall $F$ is the number of pairs of adjacent rules that have different decisions in $F$.

**Theorem 3.** *Let $F$ be a firewall that has $n$ rules.*

1) *The smallest possible value of the inversion metric of $F$ is $1$.*
2) *The largest possible value of the inversion metric of $F$ is $n-1$.*

*Proof:* Because, as mentioned in Section 3, we consider only firewalls that have two or more bands, the smallest possible value of the inversion metric of a firewall is $1$. Also, for a firewall that has $n$ rules, the largest possible value of the inversion metric is $n-1$. ■

A firewall $F$ is called *uniform* iff each band in $F$ has the same number of rules.

Thus, if a uniform firewall $F$ has $n$ rules and $k$ bands, then each band in $F$ has $\frac{n}{k}$ rules.

**Theorem 4.** *Let $F$ be a uniform firewall that has $n$ rules. Also, let $dm$ be the value of the dependency metric of $F$, and $im$ be the value of the inversion metric of $F$.*

$$dm = \frac{n * im}{2 * (im + 1)}$$

*Proof:* Since $im$ is the inversion metric of firewall $F$, $F$ has $im+1$ bands, and because $F$ is uniform, each band in $F$ has $\frac{n}{im+1}$ rules.

The cardinality of the dependency set of each rule in the $i$-th band in $F$, where $i$ is in the range $1..(im+1)$, is $\frac{(i-1)*n}{(im+1)}$.

Thus, the average cardinality $dm$ of the dependency set of a rule in $F$ can be computed as follows:

$$\begin{aligned} dm &= \sum_{i=1}^{im+1} \frac{\frac{n}{im+1} * \frac{(i-1)*n}{im+1}}{n} \\ &= \frac{n}{(im+1)^2} * \sum_{i=1}^{im+1} i - 1 \\ &= \frac{n}{(im+1)^2} * \sum_{i=0}^{im} i \\ &= \frac{n}{(im+1)^2} * \frac{im * (im+1)}{2} \\ &= \frac{n * im}{2 * (im+1)} \end{aligned}$$

■

This theorem shows that when the value of the inversion metric $im$ (of a uniform firewall) is $n-1$, the value of the dependency metric $dm$ (of the same firewall) is $(n-1)/2$. Both these values are the largest possible values for their metrics. Also, when the value of the inversion metric $im$ is reduced to 1, the value of the dependency metric is reduced to $n/4$. Both these values are small values for their metrics. In other words, there is some correlation between the value of the inversion metric $im$ and the value of the dependency metric $dm$. Thus one can use the inversion metric (which is easy to deal with) as a good approximation of the dependency metric (which is hard to deal with).

In the next two sections, we present two classes of firewalls, namely simple firewalls and partitioned firewalls, whose inversion metrics are small.

## V. Simple Firewalls

A firewall $F$ is called *simple* iff $F$ is a sequence of three bands, $B_0$ followed by $B_1$ followed by $B_2$, such that the following three conditions are satisfied:

1) Band $B_0$ consists of zero or more discard rules. (Note that if $B_0$ has zero discard rules, then band $B_0$ does not exist in $F$ and, in this case, $F$ is a sequence of only two bands, $B_1$ followed by $B_2$.)
2) Band $B_1$ consists of one or more accept rules.
3) Band $B_2$ consists of only one discard-all rule.

Simple firewalls are interesting because the values of their inversion metrics are small (and so they are easy to understand) as follows. If band $B_0$ exists in a simple firewall $F$, then the inversion metric of $F$ is 2. Otherwise, the inversion metric of $F$ is 1.

Below we describe how to identify "irrelevant rules" in any simple firewall $F$ and argue that removing these rules from $F$ yields a firewall $G$ that is both equivalent to $F$ and simple. But first we need to present some definitions.

Let $F$ be a simple firewall and let $r$ and $s$ be two distinct rules in $F$ where

$$r : f_1 \in R_1 \wedge .. \wedge f_d \in R_d \rightarrow < r.decision >$$
$$s : f_1 \in S_1 \wedge .. \wedge f_d \in S_d \rightarrow < s.decision >$$

Rule $r$ is said to *cover* rule $s$ iff every interval $R_j$ in $r$ contains the corresponding interval $S_j$ in $s$.

Rule $r$ is said to *overlap* rule $s$ iff every intersection of an interval $R_j$ in $r$ with the corresponding interval $S_j$ in $s$ is nonempty.

Rule $s$ is called *irrelevant* in the simple firewall $F$ iff $s$ satisfies the following three conditions (Recall that, since $F$ is simple, $F$ is a sequence of three bands, $B_0$ followed by $B_1$, followed by $B_2$):

1) Rule $s$ is in band $B_0$ and there is another rule $r$ in $B_0$ where $r$ covers $s$.
2) Rule $s$ is in band $B_0$ and there is no rule $r$ in $B_1$ where $r$ overlaps $s$.
3) Rule $s$ is in band $B_1$ and there is another rule $r$ in $B_1$ where $r$ covers $s$.

Now we argue that if an irrelevant rule $s$ is removed from its simple firewall $F$, then any packet that could have been resolved (i.e., accepted or discarded) by rule $s$ can still be resolved in the same way after $s$ is removed. Because the removed rule $s$ is irrelevant, rule $s$ must have satisfied one of three conditions 1, 2, or 3 (in the above definition), before it is removed.

First, if $s$ satisfied condition 1 before it is removed, then any packet that is discarded by $s$, before $s$ is removed, will still be discarded at least by rule $r$, after $s$ is removed.

Second, if $s$ satisfied condition 2 before it is removed, then any packet that is discarded by $s$, before $s$ is removed, will still be discarded at least by the discard-all rule in $F$, after $s$ is removed.

Third, if $s$ satisfied condition 3 before it is removed, then any packet that is accepted by $s$, before $s$ is removed, will still be accepted at least by rule $r$, after $s$ is removed.

The algorithm for removing irrelevant rules from any simple firewall is detailed in Algorithm 1. Note that the time complexity for executing Algorithm 1 is $O(n^2)$, where $n$ is the number of rules in the input firewall $F$.

**Algorithm 1** Removing Irrelevant Rules

```
Input: A simple firewall $F$ that is a sequence of three bands
  $B_0$ followed by $B_1$ followed by $B_2$
Output: A simple firewall $G$ that is equivalent to $F$ and has
  no irrelevant rules
  for every rule $r$ in $B_0$ do
    if there is another rule $s$ in $B_0$ such that $r$ covers $s$ or
    there is no rule $s$ in $B_1$ such that $r$ overlaps $s$ then
      Remove rule $r$ from $B_0$
    end if
  end for
  for every rule $r$ in $B_1$ do
    if there is another rule $s$ in $B_1$ such that $r$ covers $s$ then
      then remove rule $r$ from $B_1$
    end if
  end for
  The remaining firewall is $G$
```

## VI. Partitioned Firewalls

A *partitioned firewall* $PF$ is a nonempty set $\{PF_1, .., PF_r\}$ of firewalls, such that the following *oneness condition* holds. Every packet is accepted by at most one firewall, say $PF_k$, in $PF$.

If a packet is accepted by one (and so only one) firewall in a partitioned firewall $PF$, then this packet is said to be *accepted* by $PF$. Otherwise, the packet is discarded by every firewall in $PF$ and, in this case, the packet is said to be *discarded* by $PF$.

If a partitioned firewall $PF$ is the set $\{PF_1, .., PF_r\}$, then each firewall $PF_k$ in this set is called a *component* of the partitioned firewall $PF$.

Note that one can view a monolithic firewall $F$ as a partitioned firewall that consists of only one component $F$.

A monolithic firewall $F$ and a partitioned firewall $PF$ are said to be *equivalent* iff $F$ and $PF$ accept the same set of packets (and discard the same set of packets).

There are three advantages of partitioned firewalls over monolithic ones:

(a) Parallel processing of packets
(b) Ease of design and update
(c) Small inversion metrics

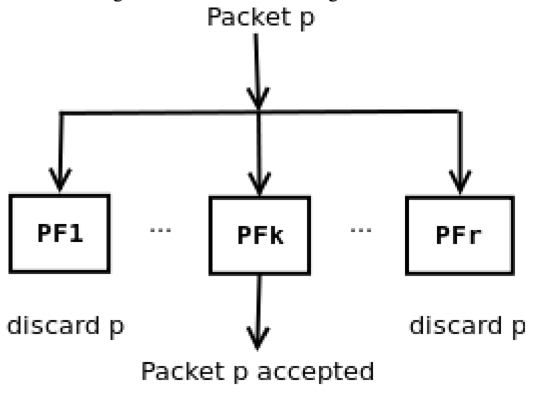
We discuss these three advantages , one by one, in order.

## A. Parallel Processing of Packets

Each component $PF_k$ of a partitioned firewall $PF$ can be implemented as a distinct thread [1] that is executed on a distinct core in a multicore architecture [2].

When a packet $p$ arrives at the multicore architecture hosting the partitioned firewall $PF$, a copy of $p$ is forwarded to each core, as shown in Figure 1. Each core then proceeds independently to determine whether or not to accept packet $p$ and allow it to proceed.

Fig. 1. Parallel Processing of Packets



Note that each core makes its determination (of whether or not to accept its copy of $p$) independently from the determinations made by the other cores. In other words, the cores do not need to synchronize in any way, and yet, thanks to oneness condition, at most one copy of packet $p$ is accepted and allowed to proceed by one core while all the other copies of $p$ are discarded by the other cores.

As shown in our experimental results below, this multicore architecture of a partitioned firewall can process up to 2.5 times as many packets per second as the traditional one core architecture of a monolithic firewall.

## B. Ease of Design and Update

A partitioned firewall $\{PF_1, .., PF_r\}$ can be designed in two steps as follows.

1) The set of all packets is partitioned into $r$ non-overlapping classes: $PC_1, .., PC_r$.
2) Each component $PF_k$ in the partitioned firewall is designed to accept some (or all) of the packets that belong to the packet class $PC_k$.

As an example, assume that we wish to design a partitioned firewall with five components $PF_1$ through $PF_5$. First, we partition the set of all packets into the five overlapping classes $PC_1$ through $PC_5$:

- $PC_1$: All outgoing packets
- $PC_2$: All incoming, TCP, email packets
- $PC_3$: All incoming, TCP, web packets
- $PC_4$: All incoming, TCP packets that are neither email nor web.
- $PC_5$: All incoming, non-TCP packets

Second, each firewall component $PF_k$ is designed to accept only some (or all) of the packets that belong to the corresponding packet class $PC_k$. For instance, $PF_1$ is designed to accept only some (or all) of the outgoing packets, and so on.

In other words, once the packet classes are all identified, the firewall components can be designed independently of one another. This makes the design of a partitioned firewall easier than that of a monolithic firewall.

Moreover, because each firewall component $PF_k$ is designed to accept only some (or all) of the packets that belong to the packet class $PC_k$, only component $PF_k$ needs to be updated whenever the set of accepted packets, that belong to the packet class $PC_k$, needs to be updated.

In other words, any update of a partitioned firewall can be realized by updating only one component in the firewall. This makes the update of a partitioned firewall easier than that of a monolithic one.

## C. Small Inversion Metric

The *inversion metric* of a partitioned firewall $\{PF_1, .., PF_r\}$ is the value
(MAX over $k$, $k$ is in the range $1..r$, $im.k$)
where each $im.k$ denotes the inversion metric of the firewall component $PF_k$.

Because the inversion metric of a partitioned firewall is the maximum, rather than say the sum, of the inversion metrics of the firewall components, the inversion metric of a partitioned firewall tends to be smaller than the inversion metric of an equivalent monolithic firewall. In other words, understanding a partitioned firewall tends to be easier than understanding an equivalent monolithic firewall.

We end this section by stating (and verifying) a sufficient condition for ensuring that two monolithic firewalls can be components in the same partitioned firewall.

**Theorem 5.** *Let $F$ and $G$ be two (monolithic) firewalls. If for every accept rule $r$ in $F$ and every accept rule $s$ in $G$, $r$ does not overlap $s$, then $F$ and $G$ can be components in the same partitioned firewall.*

*Proof:* Assume that for every accept rule $r$ in $F$ and every accept rule $s$ in $G$, $r$ does not overlap $s$. Thus, for every accept rule $r$ in $F$ and every accept rule $s$ in $G$, there is no packet that matches both $r$ and $s$. In other words, the set of packets that match accept rules in $F$ is disjoint from the set of packets that match accept rules in $G$. Moreover, because the set of packets that are accepted by a firewall is a subset of the set of packets that match accept rules in the firewall, we conclude that the set of packets that are accepted by $F$ is disjoint from the set of packets that are accepted by $G$. Therefore $F$ and $G$ satisfy the oneness condition and they can be firewall components in the same partitioned firewall. ■

Note that any two components of a partitioned firewall, that is designed using the method outlined at the beginning of this section, do satisfy the sufficient condition in Theorem 4.

## VII. Modular Firewalls

In the previous two sections, we presented two classes of firewalls, namely simple firewalls and partitioned firewalls, whose inversion metrics are small. In this section, we present a class of firewalls, called modular firewalls, that have similar characteristics to those of simple and partitioned firewalls. Therefore, the inversion metrics of modular firewalls are also small.

A *modular firewall* $MF$ is a partitioned firewall $\{MF_1, .., MF_r\}$ where each component $MF_k$, called a *firewall module*, is a simple firewall. It follows that the inversion metric of each firewall module $MF_k$ is 1 or 2 and the inversion metric of the modular firewall $MF$ is 1 or 2.

A modular firewall $\{MF_1, .., MF_r\}$ can be designed in two steps as follows.

1) The set of all packets is partitioned into $r$ non-overlapping classes: $PC_1, .., PC_r$.
2) Each module $MF_k$ in the modular firewall is designed to accept some (or all) of the packets that belong to the packet class $PC_k$ under the restriction that $MF_k$, being a simple firewall, must consist of three bands: a discard band $B_0$, followed by a accept band $B_1$, followed by a band $B_2$ that consists of a discard-all rule

The main thesis of this paper is that designing a modular firewall is easier than designing an equivalent monolithic firewall. To give some evidence to this thesis, we discuss next an algorithm that can take, as input, a monolithic firewall $F$ and produce, as output, an equivalent modular firewall $MF$. Because the time complexity of this algorithm is small $O(n^2)$, where $n$ is the number of rules in the input firewall $F$, one concludes that designing a modular firewall is not harder than designing an equivalent monolithic firewall.

The algorithm for modularizing a monolithic firewall is shown in Algorithm 2.

The correctness of Algorithm 2 follows from the following two theorems.

**Theorem 6.** *Assume that Algorithm 2 is applied to a monolithic firewall $F$ and produced the simple firewalls $\{MF_1, .., MF_r\}$. Then no two distinct firewalls $MF_i$ and $MF_k$ accept the same packet (indicating that the produced simple firewalls satisfy the oneness condition).*

*Proof:* Without loss of generality, assume that $i$ is less than $k$. This means that the accept rules in band $B_1$ of firewall $MF_i$ occur as discard rules in band $B_0$ of firewall $MF_k$. Thus, each packet that is accepted by (band $B_1$ in) firewall $MF_i$ is discarded by (band $B_0$ in) firewall $MF_k$. Also, each packet that is accepted by (band $B_1$ in) firewall $MF_k$ is discarded by (band $B_2$ in) firewall $MF_i$. In other words, no packet is accepted by both $MF_i$ and $MF_k$. ■

**Algorithm 2** Modularizing Monolithic Firewalls

```
Input: A monolithic firewall $F$ with $r$ accept bands ($r$ is at least 1)
Output: A modular firewall $MF$ with $r$ modules $\{MF_1, .., MF_r\}$ such that $F$ and $MF$ are equivalent.
  Let the $r$ accept bands of firewall $F$ be $AB_1, .., AB_r$ in order.
  for every accept band $AB_k$ in $F$ do
    Design the three bands $B_0$, $B_1$, and $B_2$ of module $MF_k$ as follows.
    • $B_0$ is the sequence of all rules that precedes $AB_k$ in $F$ after modifying their decisions to become "discard"
    • $B_1$ is the sequence of all (accept) rules in $AB_k$
    • $B_2$ is the discard-all rule;
    Apply Algorithm 1 to remove the irrelevant rules from $MF_k$
  end for
```

**Theorem 7.** *Assume that Algorithm 2 is applied to a monolithic firewall $F$ and produced a modular firewall $MF$ that consists of the modules $\{MF_1, .., MF_k\}$.*

1) *Each packet, that is accepted by $F$, is also accepted by $MF$*
2) *Each packet, that is accepted by $MF$, is also accepted by $F$*

*(These two statements indicate that $F$ and $MF$ are equivalent.)*

*Proof:*

1) Assume that a packet $p$ is accepted by $F$. Thus $p$ is resolved by a rule in some accept band $AB_k$ of $F$. This indicates that $p$ is also resolved by a rule in the accept band $B_1$ in module $MF_k$ in $MF$. Therefore $p$ is accepted by $MF$.
2) Assume that a packet $p$ is accepted by a module $MF_k$ in $MF$. Thus $p$ is resolved by a rule in band $B_1$ of module $MF_k$. This indicates that $p$ is also resolved by a rule in the accept band $AB_k$ in firewall $F$. Therefore $p$ is accepted by $F$.

■

## VIII. Simulation Results

In this paper, we presented two algorithms: Algorithm 1 for removing irrelevant rules from simple firewalls, and Algorithm 2 for modularizing monolithic firewalls. In fact, the important role of Algorithm 1 is to be invoked from within Algorithm 2 to remove the irrelevant rules from the firewall modules in the computed modular firewall. In this section, we report the results of several simulations that we carried out to measure the cost and performance of Algorithm 2. (The cost and performance of Algorithm 1 contribute to those of Algorithm 2.)

Figure 2 shows the execution time of Algorithm 2, when applied to modularize a monolithic firewall $F$, as a function of the number of rules in $F$. From this figure, the execution time of Algorithm 2 is very small, less than half a second, even when the firewall being modularized has up to 2000 rules.

Figure 3 shows the average number of firewall modules, that result from applying Algorithm 2 to modularize a monolithic firewall $F$, as a function of the number of rules in $F$. From this figure, a monolithic firewall that has 2000 rules can be converted into a modular firewall with about 22 modules on average.

Figure 4 shows the average number of rules in a firewall module, that results from applying Algorithm 2 to modularize a monolithic firewall $F$, as a function of the number of rules in $F$. From this figure, a monolithic firewall that has 2000 rules can be converted into a modular firewall where a firewall module has 800 rules on average.

Consider the case where Algorithm 2 is applied to a monolithic firewall $F$ to produce an equivalent modular firewall $MF$. As discussed in Section 6, $F$ can be implemented as a single thread on a single core architecture, whereas the firewall modules in $MF$ can be implemented on a multicore architecture. Let $RF$ denote the rate (in packets per second) of processing packets by the single core architecture, and $RMF$ denote the rate (in packets per second) of processing packets by the multicore architecture. Then $RMF/RF$ is called the speed-up ratio. Figure 5 shows the speed-up ratio as a function of the number of rules in $F$. From this figure, the speed-up ranges from 1.7 (when the number of rules in $F$ is small) to 2.6 (when the number of rules in $F$ is large).

## IX. Related Work

Firewalls are a critical line of defence in cybersecurity, but tend to be very hard to understand. As firewall correctness is a hard but important problem, there has been extensive research in the field, following four main approaches:

1) Firewall Testing: To test a given firewall $F$ , one generates many packets for which the "expected" decisions of $F$ , accept or discard, are known a priori. The generated packets are then sent to $F$, and the actual decisions of $F$ for these packets are observed. If the expected decision for each generated packet is the same as the actual decision for the packet, one concludes that the given firewall $F$ is correct. Otherwise, the given firewall $F$ has errors. Different methods of firewall testing differ in how the testing packets are generated. For instance, the test packets can be hand-generated by domain experts to target specific vulnerabilities in the given firewall $F$ , or generated from the formal specifications of the security policy of the given firewall $F$ , as in [3]. A scheme for targeting test packets for better fault coverage is given in [4] and [5]. Blowtorch [6] is a framework to generate packets for testing.
2) Firewall Analysis: To analyze a given firewall $F$, one applies an algorithm to identify (some or all of the) vulnerabilities, conflicts, anomalies, and redundancies in the given firewall $F$ . A systematic method for analyzing firewalls is presented in [7]. The concept of conflicts between rules in a firewall is due to [8] and [9]. A classification of anomalies, as well as algorithms to detect them, may be found in [10] and [11]. (This analysis works for verifying the security policies in IPsec and VPN as well [12].) A framework for understanding the vulnerabilities in a single firewall is outlined in [13], and an analysis of these vulnerabilities presented in [14]. [15] is a quantitative study of configuration errors for a firewall. An example of an efficient firewall analysis algorithm is given in FIREMAN [16].
3) Firewall Verification: To verify a given firewall $F$ against a given property $R$, one applies an algorithm to verify whether or not $F$ satisfies $R$. The question of how to query a given firewall and obtain the answer (whether or not it satisfies a given property) is discussed in [17] and [18]. The time and space complexity of these algorithms are proved to be $O(n^d)$ in [19]. In [20], a probabilistic verification algorithm is provided and shown to have a time and space complexity of $O(nd)$. In [21], we provide an elegant algorithm for firewall verification whose space complexity is $O(nd)$, and whose time complexity is order $O(n^d)$.
4) Firewall Design: To ensure a firewall does not have vulnerabilities or other problems, it can be designed from the outset using structured algorithms. Such algorithms, that can generate a firewall from its specification, are provided in [22].

In this paper, we present two new metrics for the complexity of a firewall, and show that these metrics are related. Further, we give a new algorithm for implementing firewalls such that the inversion metric of the firewall is small; this algorithm can be considered a firewall design algorithm to produce easy-to-understand firewalls. Our algorithm has the advantage that it need not be applied at the outset when designing a firewall; any pre-existing firewall may be converted to a modular firewall in $O(n^2)$ time.

The advantage of a modular firewall is the cleanness of the design; the low inversion metric makes such firewalls relatively easy to understand, and permits modification with no unexpected side effects. A side benefit is that modular firewalls, being inherently parallel, also process packets faster than equivalent conventional firewalls.

In this paper, we have dealt with modular firewalls located at a single interface between two computer networks. However, we do not see any reason why modular firewalls cannot be used for distributed firewalls, where firewall policies are distributed across many systems located at multiple points in the network [11], [10], [19]. We plan to study the possibility of developing modular distributed firewalls in future work.

Fig. 2. Execution Time to Modularize a Monolithic Firewall

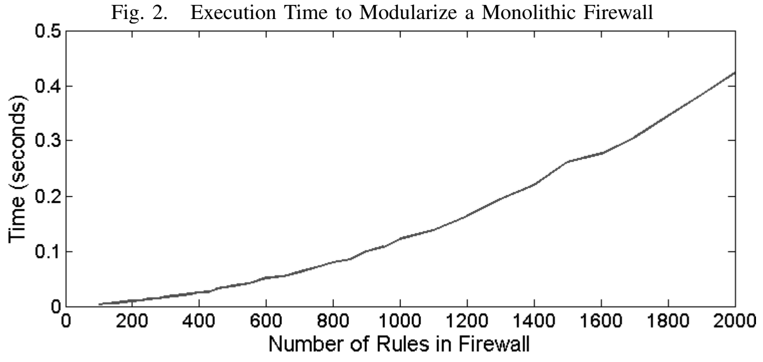


Fig. 3. Average Number of Modules Produced

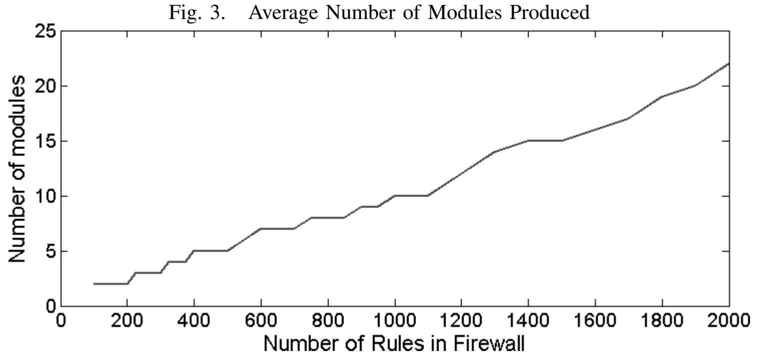


Fig. 4. Average Number of Rules per Module

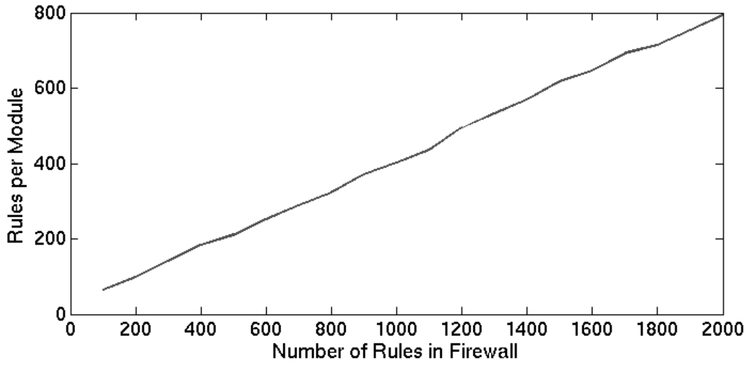
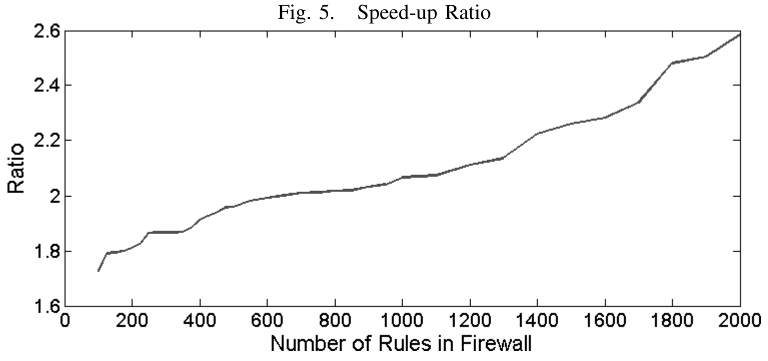


Fig. 5. Speed-up Ratio



## X. Concluding Remarks

Firewalls are a very important component of system security, but, unfortunately, current firewalls are mostly designed and modified ad hoc; this makes them very difficult to understand, so it is not uncommon for a large firewall with thousands of rules to have many vulnerabilities. In this paper, we make three important contributions to the theory of firewalls and firewall complexity. Our first contribution is that we define two metrics for the complexity of a firewall, called the dependency metric and the inversion metric. We also demonstrate that the two are correlated, so designing a firewall with a small value of inversion metric is likely to yield a firewall with a small value of dependency metric as well. For our second contribution, we present several classes of firewalls with a small inversion metric, as well as a method for designing such firewalls. Our final contribution is that we show that the class of modular firewalls, which have a low inversion metric $(1-2)$, is sufficiently powerful to describe any firewall. Algorithm 2, presented in this paper, can take as input any firewall and convert it into an equivalent modular firewall.

It may be noted that this paper introduces two separate concepts, which are interesting in their own right. The first concept is, of course, firewall metrics - we introduce the concept of dependency and inversion metrics, and develop a method to design firewalls that are "easy to understand" by these measures. The second, independent concept is that of partitioned firewalls; we show how to decompose any firewall into multiple simpler firewalls, that together are equivalent to the original firewall. By combining the concepts of simple firewalls (which have low inversion metrics) and partitioned firewalls, we develop the concept of modular firewalls.

Our work naturally suggests several rich problems for further study. The dependency metric and the inversion metric are not the only possible metrics for the complexity of a firewall; it would be an interesting problem to identify other such metrics, show how they are related, and possibly develop further algorithms to minimize the complexity of a firewall. The development of alternate algorithms to partition and modularize firewalls is another area for further research. By varying the algorithm, it is possible to produce modular firewalls with different properties, such as size, performance, length of modules, and so on. For our own future work, we note that the method for constructing a partitioned firewall involves dividing the packet space into partitions and constructing a (simpler) firewall to classify the packets of each partition independently. By clearly specifying how to partition the packet space, and when to stop partitioning and construct a firewall, we aim to develop a recursive new algorithm for firewall design.

## XI. Acknowledgements

The authors are grateful to Dr. Ehab Al-Shaer for his help in preparing the final version of this paper.

## References

[1] D. M. Tullsen, S. J. Eggers, and H. M. Levy, "Simultaneous multithreading: maximizing on-chip parallelism," in *ISCA '95: Proceedings of the 22nd annual International Symposium on Computer architecture*, 1995, pp. 392–403.

[2] J. E. Savage and M. Zubair, "A unified model for multicore architectures," in *IFMT '08: Proceedings of the 1st international forum on Next-generation multicore/manycore technologies*, 2008, pp. 1–12.

[3] J. Jürjens and G. Wimmel, "Specification-based testing of firewalls," in *Revised Papers from the 4th International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, 2001, pp. 308–316.

[4] A. El-Atawy, K. Ibrahim, H. Hamed, and E. S. Al-Shaer, "Policy segmentation for intelligent firewall testing," *Secure Network Protocols, 2005. (NPSec). 1st IEEE ICNP Workshop on*, pp. 67–72, Nov. 2005.

[5] E. Al-Shaer, A. El-Atawy, and T. Samak, "Automated pseudo-live testing of firewall configuration enforcement," *IEEE Journal on Selected Areas in Communication*, vol. 27, no. 3, pp. 302 –314, 2009.

[6] D. Hoffman and K. Yoo, "Blowtorch: a framework for firewall test automation," in *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, 2005, pp. 96–103.

[7] A. J. Mayer, A. Wool, and E. Ziskind, "Fang: A firewall analysis engine," in *IEEE Symposium on Security and Privacy*, 2000, pp. 177–187.

[8] H. Adiseshu, S. Suri, and G. M. Parulkar, "Detecting and resolving packet filter conflicts," in *INFOCOM*, 2000, pp. 1203–1212.

[9] D. Eppstein and S. Muthukrishnan, "Internet packet filter management and rectangle geometry," in *SODA*, 2001, pp. 827–835.

[10] E. S. Al-Shaer and H. H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *INFOCOM*, 2004.

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

[12] H. H. Hamed, E. S. Al-Shaer, and W. Marrero, "Modeling and verification of ipsec and vpn security policies," in *ICNP*, 2005, pp. 259–278.

[13] M. Frantzen, F. Kerschbaum, E. E. Schultz, and S. Fahmy, "A framework for understanding vulnerabilities in firewalls using a dataflow model of firewall internals," *Computers & Security*, vol. 20, no. 3, pp. 263–270, 2001.

[14] S. Kamara, S. Fahmy, E. E. Schultz, F. Kerschbaum, and M. Frantzen, "Analysis of vulnerabilities in internet firewalls," *Computers & Security*, vol. 22, no. 3, pp. 214–232, 2003.

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

[16] L. Yuan, J. Mai, Z. Su, H. Chen, C.-N. Chuah, and P. Mohapatra, "Fireman: A toolkit for firewall modeling and analysis," *Security and Privacy, IEEE Symposium on*, vol. 0, pp. 199–213, 2006.

[17] A. X. Liu and M. G. Gouda, "Diverse firewall design," *IEEE Transactions on Parallel and Distributed Systems*, vol. 19, no. 9, pp. 1237–1251, 2008.

[18] ——, "Firewall policy queries," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 6, pp. 766–777, 2009.

[19] M. G. Gouda, A. X. Liu, and M. Jafry, "Verification of distributed firewalls," in *Proceedings of the IEEE Global Communications Conference (GLOBECOM)*, 2008.

[20] H. B. Acharya and M. G. Gouda, "Linear-time verification of firewalls," in *Proceedings of the International Conference on Network Protocols*, 2009.

[21] ——, "Projection and division: Linear-space verification of firewalls," *Distributed Computing Systems, International Conference on*, pp. 736–743, 2010.

[22] M. G. Gouda and A. X. Liu, "Strucured firewall design," *Computer Networks*, vol. 51, pp. 1106–1120, 2007.