# Computer Sciences Department

**Towards Robust Firewalls Using Approximate Packet Classification**

Qunfeng Dong
Dheeraj Agrawal
Zihui Ge
Jia Wang
Jianming Wu
Suman Banerjee

Technical Report #1589

January 2007

UNIVERSITY OF WISCONSIN MADISON

# Towards Robust Firewalls Using Approximate Packet Classification

Qunfeng Dong
University of Wisconsin
Madison, WI 53706
qunfeng@cs.wisc.edu

Dheeraj Agrawal
University of Wisconsin
Madison, WI 53706
dheeraj@cs.wisc.edu

Zihui Ge
AT&T Labs - Research
Florham Park, NJ 07932
gezihui@research.att.com

Jia Wang
AT&T Labs - Research
Florham Park, NJ 07932
jiawang@research.att.com

Jianming Wu
University of Wisconsin
Madison, WI 53706
jianming@cs.wisc.edu

Suman Banerjee
University of Wisconsin
Madison, WI 53706
suman@cs.wisc.edu

## ABSTRACT

During the past decade or two, the Internet has witnessed an ever escalating demand for protection against unwanted traffic, including those carrying out malicious attacks. Packet filtering has been universally deployed in firewalls to serve as the first defense frontier against such unwanted traffic. Thus far in practice, packet filtering in firewalls has followed the conventional paradigm of *exact* packet classification, in which every packet has to be classified exactly conforming to the complete set of defined rules. However, under heavy traffic load due to unusually large traffic bursts or malicious attacks, performing exact packet classification sometimes incurs load that far exceeds the firewall's capacity. It is not rare for firewalls to crash under such circumstances, causing considerable loss of important data and extended periods of service disruption. In this paper, we propose the first robust scheme for *approximate packet classification*, which dynamically adjusts the rules to be evaluated at runtime as a function of system load, so as to reduce the drop rate and delay of legitimate packets at the firewall while still being conservative enough in filtering all unwanted packets. Through extensive simulations based on firewall rule sets and traffic logs managed by a large tier-1 ISP, we demonstrate that our proposed solution can reduce legitimate packet drop rate by as much as an order of magnitude and hence significantly improve the robustness of the firewall, especially under high traffic loads.

## 1. INTRODUCTION

During the past decade, the Internet has witnessed an escalating demand for protection against unwanted traffic, including those carrying out malicious attacks. To guard against such attacks, enterprises and networks typically construct multiple levels of defense layers consisting of both stateless and stateful components. Stateless approaches — approaches in which the decision to `permit` or `deny` a packet depends on the packet itself and no other packet — to traffic filtering are relatively simple, can operate at much higher speeds, but are not as sophisticated in detecting all unwanted traffic. Stateful approaches, though better at detecting sophisticated attacks, cannot match the speeds of stateless filtering. Both these techniques are complementary in their use. In general, we can view stateless firewalls to be the first layer of a network's defense perimeter. All traffic permitted by a stateless firewall may subsequently be inspected by more stateful approaches. The role of the stateless firewalls is, thus, to reduce the volume of traffic that stateful components have to further inspect and perform complex operations on.

The focus of this paper is on this first layer of a network's defense mechanism, i.e., on the design of stateless firewalls. Stateless firewalls perform packet filtering operations which match each incoming packet against a *rule set*, i.e., a set of rules defined over the entire packet content. Even though, the operations of a stateless firewall are relatively simple, the rules themselves might be a quite complex function of the entire packet, which evaluates each incoming packet to either `permit` or `deny` for packet filtering. For example, a rule might specify a large number of value ranges that will be matched to different components of the packet content. Dependencies might exist between different rules in a rule set such that a packet may match more than one rule. In such cases, there is a strict ordering among the rules and the goal is to find the highest priority matching rule.

For a simple illustrating example, consider the rule set in Table 1 and an incoming UDP packet, which originates from the source IP address `192.168.224.18`, targeting UDP port 1433. This packet matches both the first rule and the second (default deny) rule. (It matches the first rule because it comes from the network `192.168.224.0` whose network mask is `255.255.255.0`.) Applying the two rules in their given order in Table 1, the first rule determines the fate of the packet and hence the packet is accepted. However, if we reverse the ordering of these two rules, the default deny rule will determine the fate of the packet and hence the packet will

```
:rule (
        :source (
                : host_192.168.221.97
                : host_192.168.27.8
                : network_192.168.224.0_255.255.255.0
        )
        :destination (
                : Any
        )
        :services (
                : udp-1433-1434
                : traceroute
                : echo-request
                : ping-replies
        )
        :action (
                : permit
        )
)
:rule (
        :source (
                : Any
        )
        :destination (
                : Any
        )
        :services (
                : Any
        )
        :action (
                : deny
        )
)
```

**Table 1: Some example rules used by the firewalls.**

be dropped instead. Such and other complexities in the matching process implies that a firewall's packet filtering operations need to be implemented in software.

As transmission speeds continue to increase at a faster rate than memory access speeds [16], software-based classification systems are not always able to match the potential rates at which traffic may arrive at the firewalls. Moreover, as more and more complex rules are used to handle increasingly sophisticated attacks [29], the classification process becomes even slower, thus further hindering the ability of firewalls to match such packets at wire speeds. Hence, it is likely that during an overwhelming burst of traffic, the incoming traffic load can exceed the classification capacity of such systems. In such a scenario, incoming packets will have to be delayed in the queue, for longer and longer periods of waiting time. Eventually, the firewall will run out of critical resources such as buffer space, and start *dropping even legitimate packets without getting an opportunity to classify them.* Indeed, this is a real problem faced by many networks under high traffic loads (including misbehaving users and DoS attacks), and our exploration in this domain was triggered by multiple instances of firewall failures due to such overload.

The goal of this paper is to design *robust* packet filtering strategies in a stateless firewall that minimize the total volume of legitimate traffic that is dropped by it.

## 1.1 Our approach to robustness

Typical firewalls attempt to perform *exact* packet classification — the software filtering process will permit or deny a packet only if it is the correct action according to its rule set. We call such packet classification *semantically-exact.* In contrast, in this paper we propose to use *semantically-inexact* packet classification — classification where the firewall's software process may sometimes violate the rule set semantics, and drop some legitimate traffic as well. (It will never permit unwanted traffic through the firewall, however.) Such inexact classification will be applied only when necessary, i.e., only when the exact classification process is unable to keep up with the incoming traffic volume, will the system switch to inexact classification. Since the classification is inexact, we also call this approach *approximate packet classification* in this paper. As discussed, the approximation is conservative — no unwanted traffic (as defined by the rule set) is permitted by the firewall, but some legitimate traffic may be dropped during high loads.

Based on this simple idea, the focus of this work is to structure the classification process to meet two goals. The most important goal is to minimize the total volume of legitimate traffic that is dropped by the firewall. The secondary goal is to minimize the classification latency for all traffic that is permitted by the firewall. Note that any exact classification system will also drop some legitimate traffic under heavy loads. This will happen only due to buffer overflows prior to classification. In our approximate approach, some legitimate traffic may be dropped due to the inexact nature of the classification process. Depending on the specific techniques applied, additional legitimate traffic may still be dropped due to buffer overflows. Minimally we desire that the aggregate of all such drops of legitimate traffic in our proposed inexact case is lower than the drops in the current model of exact classification.

## 1.2 A simple example

We illustrate this approach of approximate packet classification using a simple example. Consider the rule set shown in Table 2, which is also pictorially illustrated in Figure 1. The rule set checks two fields in incoming packets, denoted by $F_1$ and $F_2$. In the figure, the two fields, $F_1$ and $F_2$, are represented along $x$ and $y$ axes, respectively. The boxes correspond to different rules. In particular, the shaded boxes correspond to rules whose decision is `permit` whereas the white boxes correspond to rules whose decision is `deny`. In the scenario depicted in Figure 1, there are 8 flows observed by the firewall, each represented by a corresponding dot. (In this paper, a *flow* corresponds to a set of all packets with the same *projection*, and where the projection of a packet is defined as the $d$-tuple consisting of the values of the $d$ fields specified in the rule set.) Rules I, II, III and IV match 4, 2, 1 and 1 of these 8 flows, respectively.

| | | |
|---|---|---|
| Rule I | : | $(F_1 \in [10, 70]) \wedge (F_2 \in [40, 65]) \rightarrow permit$ |
| Rule II | : | $(F_1 \in [20, 85]) \wedge (F_2 \in [20, 60]) \rightarrow permit$ |
| Rule III | : | $(F_1 \in [25, 75]) \wedge (F_2 \in [55, 85]) \rightarrow permit$ |
| Rule IV | : | $(F_1 \in [0, 100]) \wedge (F_2 \in [0, 100]) \rightarrow deny$ |

**Table 2: A rule set of 4 rules. Rules ordered by priority.**

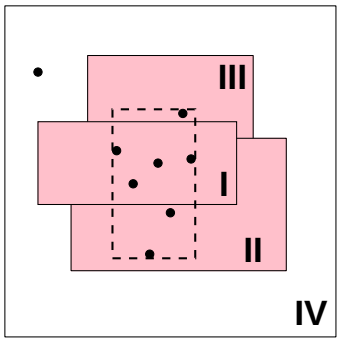


**Figure 1: Approximate packet classification based on the rule set in Table 2.**

Among these 8 flows, the 7 flows matched by Rules I, II and III are legitimate flows, while the other one should be denied.

To provide the basic intuition, we assume a naïve packet classification algorithm which compares each incoming packet with every individual rules in order[1]. Let us say the firewall is capable of comparing a rule with 100 units of traffic per second, and each flow contributes 10 units of traffic per second. Exact classification of the 8 flows requires a classification capacity of comparing a rule with $4\times1\times10+2\times2\times10+1\times3\times10+1\times4\times10 =$ 150 units of traffic per second. Consequently, incoming packets will get delayed in the queue and the firewall will end up dropping one third of incoming legitimate packets. Let us assume the queue can accommodate $L$ packets. For those legitimate packets that are not dropped, as the queue is always full, they have to wait for all the $L$ packets already in queue to be classified, before they can be classified. That represents a significant delay for those legitimate packets that were eventually permitted by the firewall.

The basic idea of approximate packet classification is quite simple: if we can somehow quickly approve a considerable percentage of legitimate packets to avoid accumulating packets in the queue, possibly at the cost of mistakenly denying a small percentage of legitimate packets, the *legitimate packet drop rate* may still be lower than that caused by exact packet classification, since the system will not have to drop (possibly a large pencentage of) packets due to buffer overflow. Moreover, the average delay on legitimate packets will hopefully be much lower than the delay incurred by exact packet classification. The system should also provide the additional guarantee that unwanted packets should always be denied.

We, first, consider a simple approximate packet classification scheme where the firewall only compares each incoming packet with the first $K$ rules and drops all packets that do not match them. Let us examine the case when $K = 2$. Such approximate classification of the 8 flows requires a classification capacity of comparing a rule with $4\times1\times10+2\times2\times10+1\times2\times10+1\times2\times10 =$ 120 units of traffic per second. As a result, the firewall will only need to drop $\frac{120-100}{120} = \frac{1}{6}$ of incoming (legitimate) packets, although the queue is still full and hence the long delay on approved packets remains there. It can be verified that $K = 1$ or $K = 3$ will lead to more drops of legitimate traffic than the $K = 2$ case.

However, this is by no means the best we can do. If we construct a new rule Rule $X$: $(F_1 \in [32, 55]) \wedge (F_2 \in [32, 68]) \rightarrow permit$ as illustrated by the dashed box in Figure 1, this single rule will match all 7 legitimate flows and executes the same action. In this case, we will need to simply compare each incoming packet with this single new rule $X$, which requires a classification capacity of comparing a rule with $4\times1\times10+2\times1\times10+1\times1\times10+$ $1\times1\times10 = 80$ units of traffic per second, which is within the firewall's classification capacity. Consequently, the firewall does not have to put packets in queue or drop them. The legitimate traffic drop rate and the delay of legitimate traffic both reach zero, in this example of approximate packet classification.

## 1.3 Design objectives and challenges

The above example illustrates that *under heavy load conditions* a careful design of semantically-inexact classification can actually be better than semantically-exact one and a poor design may hurt performance. Here, we identify the following to be key requirements for the design of inexact classification techniques:

- The inexact classification should not lead to unnecessary packet drops for legitimate traffic when the incoming volume of traffic is low. In particular, under low loads inexactness would not be useful.
- Under high loads, inexact classification should lead to lower drop rate and lower delay for legitimate traffic than exact classification.
- No unwanted traffic will be permitted even when inexact classification is in effect.

Our approximate packet classification system meets all of these requirements by answering the following specific questions.

1. When and how should we switch between exact and inexact classification schemes? While inexact classification can reduce legitimate packet drop rate under high loads, we do not want to use it

[1]Indeed, this simple naïve algorithm is implemented in many real systems. Moreover, the classification speed of other proposed packet classification algorithms depends on the number of used rules as well.

under low loads since it may unnecessarily drop legitimate packets due to its inexact nature.

2. How shall we obtain the new rules for further improving classification efficiency?
3. Which of the new rules and given rules should we use for approximate classification of incoming packets?
4. As incoming traffic pattern changes, how should we update the set of rules we use in the approximate classification scheme?
5. How to make sure that unwanted packets are never permitted by the firewall?

This paper represents a first exploratory step to solve these problems. In particular, we start with a recently proposed efficient technique for exact classification (proposed in the context of traffic classification in core routers using TCAMs) [33] which we adapt to improve firewall performance. This solution, however, still gets overloaded when large bursts of traffic arise. We then introduce a systematic approach to implementing inexact classification such that it achieves our desired performance objectives for firewalls. Through comparisons, using real traffic traces and real rule sets from a tier-1 ISP, we show that the inexact classification scheme leads to significant performance gains (both in terms of latency and drop rate for legitimate traffic) over the exact classification scheme, especially under high loads. In particular, our proposed solution can reduce legitimate packet drop rate by as much as an order of magnitude, and reduce packet delay by as much as a factor of 4. When the incoming traffic load is low, our proposed solution seamlessly converges to exact classification and hence avoids unnecessary drops of legitimate packets under low loads.

It is worth emphasizing that, while our work in this paper focuses on stateless packet classification only, the general idea of inexact packet classification is likely to apply beyond that. We expect to further explore that direction in our future work.

### 1.4 Road map

The rest of the paper is organized as follows. In Section 2, we present the design of our proposed approximate classfication scheme. We evaluate its performance through extensive simulations in Section 3. After reviewing related work in Section 4, we conclude the paper in Section 5.

## 2. DESIGN

In this section, we describe our design for approximate packet classification to improve robustness of stateless firewalls.

We start with the observation that most rule sets have significant redundancy in their rules [26, 32]. In particular, different firewall rules may get added at different points in time, possibly triggered by different sources of reported vulnerabilities. It is possible that a newly added rule is partially, or even completely, covered by other rules. Hence, the first step in designing our system is to eliminate such redundancies, by transforming a specified rule set into a new rule set that is semantically equivalent, i.e., the classification decision of the new rule set is identical to the original rule set [26, 32, 33]. The new rule set is just a more efficient version of the original rule set for exact packet classification.

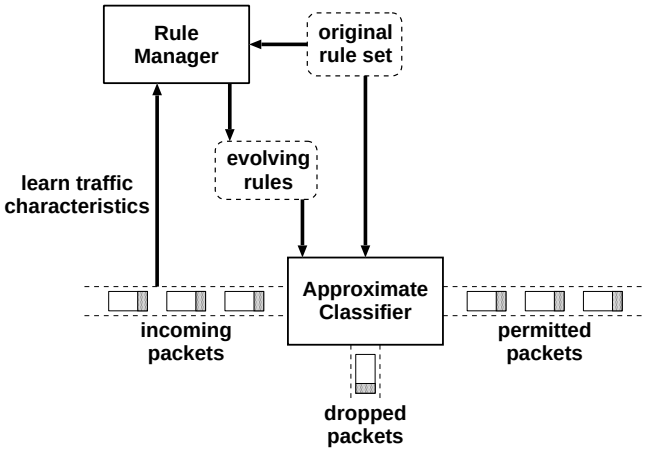


**Figure 2: Framework of our design.**

The focus of this paper is on the second step, in which we build an approximate classifier based on the new rule set, by carefully introducing inexactness during periods of high loads in lieu of faster classification speeds.

Our design of the approximate classification system, therefore, consists of two components — the *rule manager* and the *approximate classifier* as shown in Figure 2. The rule manager is responsible for the first step, while the approximate classifier is responsible for the second step. In particular, the approximate classifier tries to classify incoming packets in an adaptive and not necessarily exact manner, using a certain subset of the rules provided by the rule manager as well as the original rule set. It adapts its choice of this subset in response to changes in incoming traffic. In our proposed solution, we select a rule manager that guarantees exact classification selected from prior best known schemes in the literature [33], and we summarize its characteristics in Section 2.1. The design of the approximate classifier is at the heart of our proposed approximate classification scheme, and is the key contribution of this paper. We describe the design of our efficient approximate classifier in Section 2.2.

### 2.1 Rule manager

Prior work [33] describes a rule manager designed for exact packet classification using TCAMs in core routers. We briefly summarize the design of this rule manager in this section, which we adapt for our use in software-

based stateless firewalls. (We will also borrow terminology from this subsection in describing the approximate classification process.)

To build an efficient rule set, the rule manager continuously samples incoming traffic and computes specific statistics of this sampled traffic to learn its current characteristics. In particular, the rule manager calculates all distinct sampled flows and their frequency (which we will refer to as *weight*) in the sample. Based on this sampled information, the rule manager creates and maintains a small set of new rules that cover all sampled packets, and dynamically evolve these rules in response to traffic pattern changes. We call them *evolving rules*. The evolving rules created will (typically) match a significant portion of incoming traffic and hence can be effectively used later for improving the efficiency of both exact classification and inexact classification. These evolving rules possess the following key properties, which make our approximate packet classification a lot easier to implement.

- *Each evolving rule is semantically consistent with the original rule set.* Namely, if an evolving rule matches a packet, its decision (on that packet) is always the same as the decision specified by the original rule set.
- *The packets of each distinct sampled flow always get assigned to one evolving rule that matches it.* This ensures the evolving rules contain the entire sampled information. The *weight* of each evolving rule is defined to be the total weight of its assigned flows, i.e., the total number of assigned sample packets. After normalization, *the normalized weight of an evolving rule is an estimate of the percentage of incoming packets that will be matched by this rule*. The approximate classifier tries to adopt an appropriate classification strategy based on this estimation.
- *The evolving rules are structured in a way such that if two rules match the same packet, they must have the same decision.* This greatly simplifies the approximate packet classification, because this allows us to use the evolving rules in any order for approximate packet classification.

## 2.2 Approximate classifier

In this subsection, we describe our design of the approximate classifier, which classifies incoming packets in a way that adapts to incoming traffic. Suppose there are $l$ evolving rules, $R_1$, $R_2$, $\cdots$, $R_l$, provided by the rule manager. Let $w_1$, $w_2$, $\cdots$, $w_l$ denote their normalized weight, respectively. Our approximate classifier employs a combination of two classification schemes, as described below in Section 2.2.1 and Section 2.2.2, and carefully switches between these two schemes depending on the dynamics of incoming traffic load as discussed in Section 2.2.3.

### 2.2.1 Scheme I

This scheme matches each incoming packet against a certain small number (denoted by $m$) of evolving rules provided by the rule manager, using some packet classification algorithm $\mathcal{A}_0$. If a matching rule is not found, we then match the packet against the original rule set (which contains $n$ rules), using some other packet classification algorithm $\mathcal{A}$.

We intentionally use this two stage classification process based on the following observations. Typical rule sets in firewalls we consider have the order of $10^4 \sim 10^5$ rules. However, in normal operations, it has been reported that a large volume of the traffic often match just a few rules [28]. Employing a single stage classification process over the entire rule set to find such a match can therefore be much less efficient, even if the best known classification algorithm [4] is applied. Instead, if we can carefully select a small number (say, $m < 10$) of popular evolving rules, even a simple sequential search approach (used as algorithm $\mathcal{A}_0$) will deliver much higher performance. On failure, the packet can then be compared against the entire rule set using more sophisticated techniques applicable for large rule sets.

It is worth emphasizing that *we do not need to make any assumption about $\mathcal{A}_0$ and $\mathcal{A}$.* As an initial example for demonstrating the effectiveness and potential of approximate packet classification, let us simply take naïve sequential search as the algorithm we use as $\mathcal{A}_0$. Because the rule manager typically provides a very small number of evolving rules that are highly popular. Employing sophisticated classification algorithms using these evolving rules can only generate very marginal efficiency improvement. Moreover, as the evolving rules are frequently updated (i.e., evolved) by the rule manager, sophisticated algorithms typically have to re-compute sophisticated data structures upon every update by the rule manager. The added overhead by far exceeds the marginal performance gain.

To perform sequential search (algorithm $\mathcal{A}_0$) through the evolving rules, we sort the evolving rules into a list $\mathscr{L}$ (in some order we shall discuss shortly). Note that simply searching through the entire list of $l$ evolving rules does not necessarily lead to optimal performance, as we shall analyze below. Instead, we should carefully compare each incoming packet with the first $m$ evolving rules in $\mathscr{L}$, where $m \leq l$. In fact, this is an essential advantage of our Scheme I over the exact classification scheme using the rule manager proposed in [33], which simply uses a fixed number of evolving rules for improving classification efficiency.

To determine the optimal value of $m$, let us suppose the evolving rules are indexed based on their position in $\mathscr{L}$. The estimated workload when using the first $m$ evolving rules is equal to comparing each incoming

packet with an average of

$$N_1(m) = \left(\sum_{k=1}^{m} w_k k\right) + (m + W(n)) \left(1 - \sum_{k=1}^{m} w_i\right)$$

rules, where $W(n)$ denotes the average number of comparisons per packet incurred by the complete packet classification algorithm $\mathcal{A}$ using the original rule set (containing $n$ rules). Here, we do not need to make any assumption about $W(n)$. The firewall can use any complete classification algorithm $\mathcal{A}$ applicable for large rule sets. Note that if $m = 0$, Scheme I is reduced to the original single stage packet classification scheme used by the firewall.

Let us say the firewall's classification capacity enables it to perform an average of $C$ comparisons for each incoming packet. In general, there are two cases where an incoming packet may be dropped.

1. Before an incoming packet enters the queue, the packet may be directly dropped without classification, due to a full queue in the case of system overload. We refer to such drops as *pre-queuing drops*.
2. After an incoming packet enters the queue, the packet may be dropped according to a classification decision. We call such drops *post-queuing drops*.

In Scheme I, there is no post-queuing drop of legitimate packets, because queued packets are always correctly classified. However, incoming legitimate packets may be dropped due to system overload (i.e., pre-queuing drop). Therefore,

- If $N_1(m) \leq C$, the firewall is able to handle the incoming traffic load and hence does not have to drop (legitimate) packets.
- If $N_1(m) > C$, the firewall is only able to handle $\frac{C}{N_1(m)}$ of incoming traffic and hence the estimated pre-queuing packet drop rate is $1 - \frac{C}{N_1(m)}$. Since packets are dropped without classification here, we assume such pre-queuing drops are completely random. Thus, legitimate packets are dropped with the same probability $\rho = 1 - \frac{C}{N_1(m)}$.

*In both cases, we want to minimize $N_1(m)$ in order to minimize $\rho$.* Thus in $\mathscr{L}$, we should sort the evolving rules in non-increasing order of weight. (Because $m + W(n) > k$ for any $k \leq m$.) An optimal value of $m$ that minimizes $N_1(m)$ can be easily determined by checking all possible values of $m \in [0, l]$. The calculations can be done quite efficiently, especially given the fact that $l$ is (typically) very small.

### 2.2.2 Scheme II

Compared with the exact Scheme I, our inexact Scheme II is even more aggressive. In Scheme II, if a matching rule is not found among the evolving rules, we simply drop the packet without further classifying it using the original rule set, which introduces (slight and conservative) inexactness for (significantly) decreased workload and hence much better efficiency.

In this scheme, we also match incoming packets against the first $m$ evolving rules in $\mathscr{L}$, using sequential search. (However, the way we determine the list $\mathscr{L}$ and the value of $m$ is different from Scheme I.) The estimated workload is equal to comparing each incoming packet with an average of

$$N_2(m) = \left(\sum_{k=1}^{m} k w_k\right) + m \left(1 - \sum_{k=1}^{m} w_i\right)$$

rules. Compared with Scheme I, Scheme II is less likely to drop packets due to overload (i.e., pre-queuing drops), since the incurred workload is much lower. But it may drop packets due to mistaken classification decisions (i.e., post-queuing drops), due to its inexactness.

For ease of presentation, we also define the notion of *positive weight* (denoted by $w_i^+$) for each evolving rule $R_i$. If the decision of $R_i$ is `permit`, then $w_i^+ = w_i$ and we refer to $R_i$ as a *positive rule*; If the decision of $R_i$ is `deny`, then $w_i^+ = 0$ and we refer to $R_i$ as a *negative rule*.

- If $N_2(m) \leq C$, the firewall is able to handle the incoming traffic load and packets are only dropped as a classification decision. Thus, the estimated legitimate packet drop rate is given by

$$\rho = \frac{\sum_{k=m+1}^{l} w_k^+}{\sum_{k=1}^{l} w_k^+}.$$

- If $N_2(m) > C$, firewall is only able to handle $\frac{C}{N_2(m)}$ of incoming traffic. The estimated percentage of (legitimate) packets that are dropped before queuing is $\rho_1 = 1 - \frac{C}{N_2(m)}$, and the estimated percentage of legitimate packets that are dropped after queuing is $\rho_2 = \frac{C}{N_2(m)} \times \frac{\sum_{k=m+1}^{l} w_k^+}{\sum_{k=1}^{l} w_k^+}$. The aggregate

legitimate packet drop rate $\rho$ is thus given by

$$\rho = \left(1 - \frac{C}{N_2(m)}\right) + \frac{C}{N_2(m)} \times \frac{\sum_{k=m+1}^{l} w_k^+}{\sum_{k=1}^{l} w_k^+}$$

$$= 1 - \frac{C}{N_2(m)} \times \frac{\sum_{k=1}^{m} w_k^+}{\sum_{k=1}^{l} w_k^+}.$$

*In both cases, we want to maximize $\sum_{k=1}^{m} w_k^+$ and minimize $N_2(m)$, in order to minimize $\rho$.* Unlike the case in Scheme I, simply sorting the evolving rules in non-increasing order of weight may not minimize $\rho$ here. To determine the optimal list $\mathscr{L}$ of evolving rules to be used for approximate packet classification, we show there must exist an optimal list $\mathscr{L}$ that satisfies the following properties.

I. The first $m$ evolving rules in $\mathscr{L}$, regardless of their decision, are sorted in non-increasing order of weight. To see that, consider two evolving rules $R_i$ and $R_j$ that both appear in the first $m$ evolving rules of $\mathscr{L}$. Suppose $w_i < w_j$ and $R_i$ appears before $R_j$ in $\mathscr{L}$. If we switch $R_i$ and $R_j$, the value of $N_2(m)$ will decrease and the value of $\sum_{k=1}^{m} w_k^+$ will not change. The value of $\rho$ will decrease.

II. Positive rules in $\mathscr{L}$ are sorted in non-increasing order of weight. To see that, consider two positive rules $R_i$ and $R_j$ in $\mathscr{L}$. Suppose $w_i < w_j$ and $R_i$ appears before $R_j$ in $\mathscr{L}$. If we switch $R_i$ and $R_j$, the value of $N_2(m)$ will not increase and the value of $\sum_{k=1}^{m} w_k^+$ will not decrease. The value of $\rho$ will not increase.

III. Negative rules in $\mathscr{L}$ are also sorted in non-increasing order of weight. To see that, consider two negative rules $R_i$ and $R_j$ in $\mathscr{L}$. Suppose $w_i < w_j$ and $R_i$ appears before $R_j$ in $\mathscr{L}$. If we switch $R_i$ and $R_j$, the value of $N_2(m)$ will not increase and the value of $\sum_{k=1}^{m} w_k^+$ will not change. The value of $\rho$ will not increase.

IV. The $m$-th evolving rule in $\mathscr{L}$ should be a positive rule. Otherwise, there is no need to compare incoming packets with the $m$-th evolving rule, since the packets will be dropped anyway.

By property II, let us assume the $k$ highest-weight positive rules are in the first $m$ evolving rules of $\mathscr{L}$. By property III, the $m-k$ highest-weight negative rules are in the first $m$ evolving rules of $\mathscr{L}$. Then by property I, these first $m$ rules should be sorted in non-increasing order of their weight. Finally, we check if the $m$-th rule satisfies property IV. Thus, once $k$ and $m$ are given, an optimal list $\mathscr{L}$ can be determined, if such an optimal list $\mathscr{L}$ exists for the given $k$ and $m$ at all. That said, an optimal list $\mathscr{L}$ can be found after checking all possible values of $k \in [1, m]$. A pseudo code description of this simple algorithm is given in Table 3.

```
for (m = 0; m ≤ l; m++) {
  for (k = 0; k ≤ m; k++) {
    /* Based on Property II ...  */
    if there are less than k positive rules
      continue;
    pick the k highest weight positive rules;
    /* Based on Property III ...  */
    if there are less than m − k negative rules
      continue;
    pick the m − k highest weight negative rules;
    /* Based on Property I ...  */
    sort the m rules in non-increasing order of weight;
    /* Based on Property IV ...  */
    if the m-th rule is a negative rule;
      continue;

    compute ρ for this sorted list 𝓛;
    keep the optimal 𝓛 that minimizes ρ so far;
  }
}
```

**Table 3: Compute an $\mathscr{L}$ that minimizes $\rho$ in Scheme II.**

### 2.2.3 Approximate classification algorithm

Given the design and analysis of Scheme I and Scheme II, we now present a simple yet effective algorithm for approximate packet classification. Our algorithm dynamically switches between Scheme I (which is exact) and Scheme II (which is inexact), with preference being given to Scheme I if the packet drop rate is already quite low. Because ideally, if we ignore packet drops due to traffic bursts (much of which we try to handle by actively adapting to changes in incoming traffic pattern), Scheme I guarantees correct classification of every incoming packet. In contrast, Scheme II does not provide such a guarantee.

Initially, our algorithm starts in Scheme I. According to the analysis in Section 2.2.1, the optimal strategy is to choose a value of $m$ such that $N_1(m)$ is minimized. To effectively adapt to incoming traffic load, we continuously monitor the pre-queuing drop rate $\rho_0$ of recently received packets, and adapt our classification scheme accordingly.

Let us first consider the case where the current scheme in use is Scheme I. If $\rho_0$ does not exceed a threshold (e.g. 3% in our simulations), which is quite low, we continue to use Scheme I for classification. Because on one hand, the legitimate packet drop rate $\rho$ is equal to the pre-queuing drop rate $\rho_0$ in Scheme I. On the other hand, Scheme II always has a certain probability of mistakenly dropping legitimate packets, due to its

inexact classification of incoming packets. Therefore, continue using Scheme I is a conservative and acceptable choice, especially when the threshold is quite low.

To decide the optimal value of $m$, we can estimate the constant $C$ in our formula by $\rho_0 = 1 - \frac{C}{N_1(m)}$, which gives $C = (1 - \rho_0)N_1(m)$. [2] An apparent merit of this approach is that $C$ and $\rho_0$ are estimated in a real time manner, which provides us with a dynamic view of the system's currently available capacity and incoming traffic load. Using this estimation, explicit knowledge about currently available system capacity and incoming traffic load is not required, which greatly simplifies the design and implementation of our approximate classification scheme. However, if $\rho_0 = 0$, this may underestimate $C$. Therefore, in such cases we will not update our estimation of $C$. Using the estimated value of $C$, we can determine the optimal value of $m$ as described in Section 2.2.1.

If $\rho_0$ exceeds the threshold (and hence $\rho_0 > 0$), we need to decide which scheme to use and what the optimal value of $m$ should be. Again, we can estimate $C$ by $\rho_0 = 1 - \frac{C}{N_1(m)}$. After estimating $C$, we shall choose a value of $m$ and one of Scheme I and II that minimize the drop rate $\rho$ of legitimate packets. To minimize $\rho$ in Scheme II, we compute an optimal list $\mathscr{L}$ as well as an optimal value of $m$, as is described in Table 3. If this optimal estimated value of $\rho$ in Scheme II is lower than the optimal estimated value of $\rho$ in Scheme I, we will use Scheme II with that $m$ value for approximate packet classification. Otherwise, Scheme I will be used with its optimal $m$ value.

Now let us consider the case where the current scheme in use is Scheme II. Similarly, the constant $C$ can be estimated by $\rho_0 = 1 - \frac{C}{N_2(m)}$. If $\rho_0 = 0$, we will not update our estimation of $C$ to avoid possible underestimation. Using the estimated value of $C$, we shall similarly choose a value of $m$ and one of Scheme I and II that minimize the drop rate $\rho$ of legitimate packets, as described above.

A brief pseudo code description of this adaptation algorithm used by our approximate classifier is given in Table 4.

```
ChooseScheme;
Scheme I :
    if (ρ0 ≤ threshold) {
      choose Scheme I;
      if (ρ0 > 0)
        C = (1 − ρ0)N1(m);
      pick the optimal value of m;
      return;
    }
    if (ρ0 > threshold) {
      if (ρ0 > 0)
        C = (1 − ρ0)N1(m);
      pick the optimal scheme and value of m;
      return;
    }
Scheme II :
      if (ρ0 > 0)
        C = (1 − ρ0)N2(m);
      pick the optimal scheme and value of m;
      return;
```

**Table 4: The adaptation algorithm.**

## 3. EVALUATION

We evaluate the performance of our proposed solution for approximate packet classification using real rule sets and traffic logs obtained from firewalls managed by a large ISP. Each day, the firewalls dump a snapshot of their rule sets as well as a 24-hour-long traffic log of that day.

- Each rule set contains up to the order of $10^4$ rules. The decision of rules is either `permit` or `deny`.
- Each traffic log is essentially a list of flow records of incoming packets at the firewall for a single day. Each record contains a number of fields, including timestamp, source and destination IP addresses, source and destination port numbers, protocol type, etc. Each traffic log lasts one day. Since these firewalls only log traffic that have been successfully classified, packets that were dropped prior to classification (e.g., packet loss due to buffer overflow) do not have corresponding records in the traffic logs. This recording strategy at the firewalls would limit our ability to study behavior patterns during high loads. Hence, in our simulations, we generate heavier traffic loads by compressing each 24 hour long traffic log into a shorter time duration.

In our simulations, we try to classify each traffic log using the rule set snapshot obtained from the same firewall on the same day, using different classification schemes we are going to evaluate and compare.

### 3.1 Schemes and metrics

In simulations, we can possibly compare our proposed approximate classification scheme against the best possible classification algorithm as our reference point. As the authors showed in [4], for packet classification over $d > 3$ fields, the best known algorithm has $O(\log n)$ search time at the cost of $O(n^d)$ space, where $n$ is the number of rules in the rule set. Note that the bound provided above makes no assumption about the incoming traffic pattern.

Our Scheme I is also an exact classification algorithm that enhances any existing exact classification algorithm, $\mathcal{A}$, with another stage, a stage that first learns popu-

[2] We use pre-queuing drop rate $\rho_0$ instead of the drop rate $\rho$ of legitimate packets to estimate $C$, because calculating $\rho_0$ does not require knowing if a dropped packet is legitimate or not, which is more realistic.

larity of rules from the incoming packets and checks packets against these popular rules. If a match is not found among these popular rules, then the full classification is performed using algorithm $\mathcal{A}$, using all of the rules in the second stage. This traffic-aware approach of Scheme I leads to performance improvements over the corresponding traffic-unaware classification algorithm, $\mathcal{A}$, both in terms of legitimate packet drop rate and classification latency. These gains have been consistently verified in our simulations.

Therefore, we pick Scheme I as the exact classification algorithm that uses the most efficient traffic-unaware algorithm in its second stage. The running time of this traffic-unaware classification algorithm is $\log n$ for each packet as reported in [4]. In the simulation results we present here, we shall focus on comparing our proposed approximate classification scheme (which adaptively switches between Scheme I and Scheme II) with Scheme I. Scheme II itself is an approximate classification scheme that has no second stage and hence, and does not depend on the the computational speed of the best known exact classification algorithm.

One problem that confronted us in our simulation setup is to construct reasonable scenarios where overload does happen, at least from time to time. The recorded traffic logs are for packets that were successfully classified by the firewalls. Hence, the aggregate of these packets will not cause overload of the firewall they traverse. In particular, packets that were dropped due to the overload will simply not be recorded in these logs.

Thus, in our simulations, we artificially make overload present by properly setting the processing capacity of the simulated firewall. To make our presentation of simulation results as intuitively clear as possible, we present traffic load in a normalized form. In particular, each simulation involves a traffic log, a classification scheme, and a rule set. We calculate for each simulation an *average capacity demand (ACD)*, which is basically the product of the average packet rate of the traffic log and the average number of comparisons needed to classify a packet using the classification scheme and the rule set. The average capacity demand of each simulation is normalized to the firewall's capacity.

## 3.2 Results

To evaluate the performance of our proposed approximate classification scheme, we have conducted many simulations with different rule sets and traffic logs. In general, for low traffic loads, the proposed approximate classification scheme will use the exact classification technique in Scheme I, and hence will neither perform better or worse than the exact classification scheme. It is only when the traffic volume approaches or exceeds the firewall capacity, does the inexactness of our proposed classification mechanism take effect, and lead to performance gains. Hence, in this section, we focus primarily on scenarios where the average traffic demands are relatively high, ranging from 70% to 110% of the firewall's capacity.

We first present detailed simulation results from a representative pair of traffic log and rule set. Subsequently, we will present results from a number of other representative rule sets and traffic logs.

**Representative rule set and traffic log:** In Figure 3–5, we present the legitimate packet drop rate achieved by different schemes, with the average capacity demand being equal to 90%, 100% and 110% of the firewall's capacity, which represent the cases where incoming traffic load approaches, reaches, and exceeds the firewall's capacity, respectively. In these simulations, we calculate and present the legitimate packet drop rate observed during each minute-long period of the entire hour. The first five minutes are taken as a warm-up stage and hence not presented in the figures.

As we can see from Figure 3, even if the firewall has sufficient capacity to handle the *average* capacity demand, traffic burstiness still generates overloads on the firewall from time to time. There are many one-minute durations when the exact classification scheme was overloaded and dropped between 5-30% of the incoming traffic (between time 15–47 minutes). In contrast, our proposed approximate scheme was very effective in avoiding most drops of legitimate packets.

As the incoming traffic load increased to 100% in Figure 4 and 110% in Figure 5, the performance of the exact classification approach (Scheme I) degrades significantly, while our approximate scheme only suffers slight performance degradation.

The above conclusions are not only the case for legitimate packet drop rate, but also the case for packet delay. To demonstrate that, we also calculate the average packet delay for each minute-long period, and present the results in Figure 6–8.

To further evaluate how the performance of different schemes degrades as the incoming traffic load (relatively) increases, we have also conducted the same simulation with other different firewall capacities. The observed cumulative drop rate of legitimate packets and average packet delay are presented in Figure 9 and Figure 10, respectively. As we can see, the performance of Scheme I proportionally degrades with incoming traffic load, and the effectiveness of our approximate scheme steadily increases with incoming traffic load. In general, the legitimate packet drop rate and average packet delay of our approximate scheme is roughly an order of magnitude lower than Scheme I.

**Other rule sets and traffic logs:** Next, we present simulations results from other rule sets and traffic logs obtained from the same firewalls. The observed cumulative drop rate of legitimate packets and average packet delay are presented in Figure 11 and Figure 12, respectively. For these rule sets and traffic logs, our approximate scheme can reduce legitimate packet drop rate by as much as an order of magnitude, and reduces average packet delay by as much as a factor of 4.

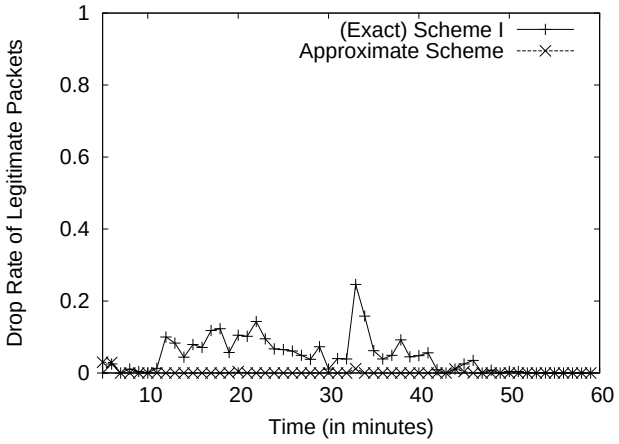


**Figure 3: ACD = 90% of Capacity.**

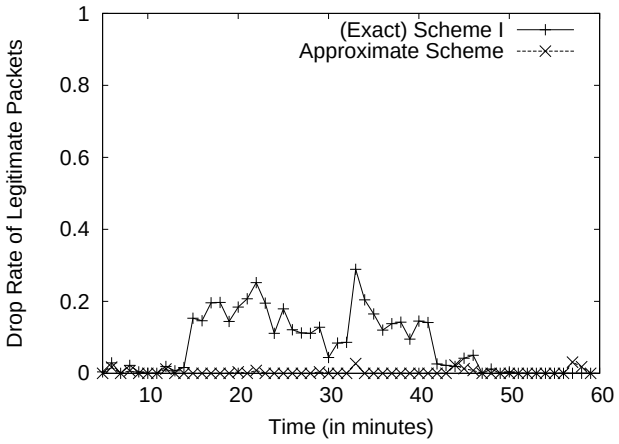


**Figure 4: ACD = 100% of Capacity.**

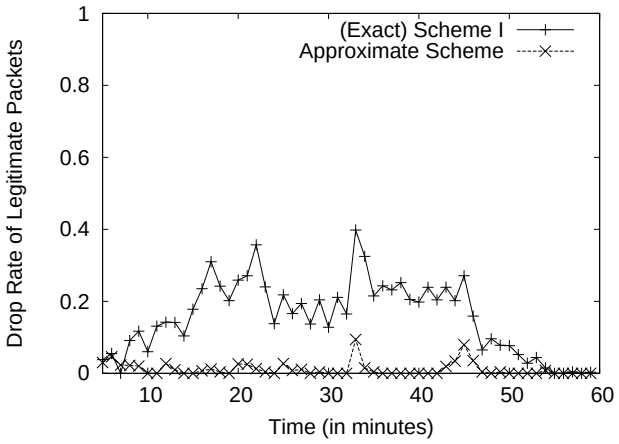


**Figure 5: ACD = 110% of Capacity.**

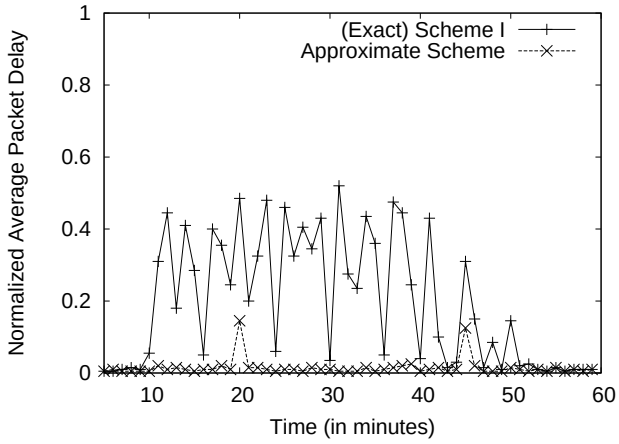


**Figure 6: ACD = 90% of Capacity.**

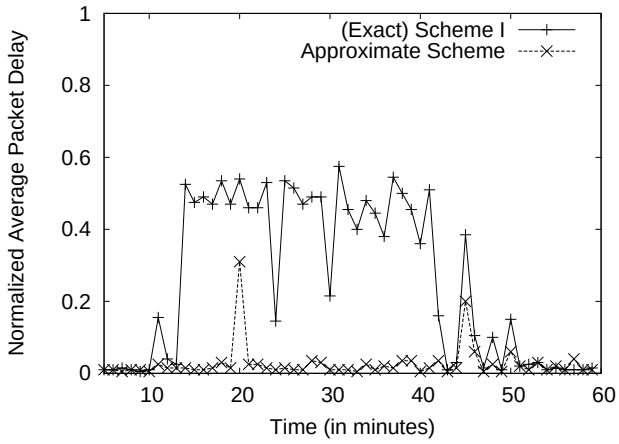


**Figure 7: ACD = 100% of Capacity.**

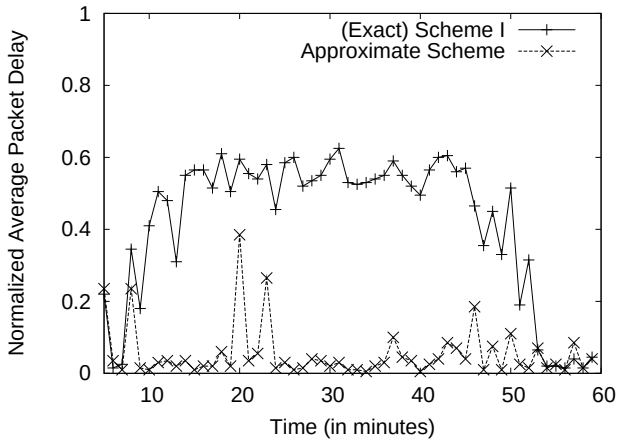


**Figure 8: ACD = 110% of Capacity.**

## 4. RELATED WORK

Packet classification on multiple fields was first studied in [5] and [6]. Since then, there have been two primary lines of research on designing efficient packet classification schemes. A long thread of research [5, 6, 8, 9, 10, 12, 15, 14, 19, 18, 21, 22, 27, 28, 31] has been devoted to designing efficient algorithms for packet classification. The other thread of research focuses on designing efficient packet classification schemes based on TCAMs [17, 20, 23, 25, 29, 32].

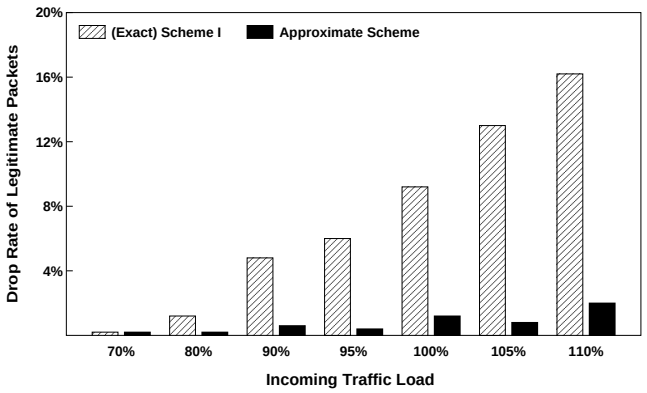


**Figure 9: Legitimate packet drop rate.**

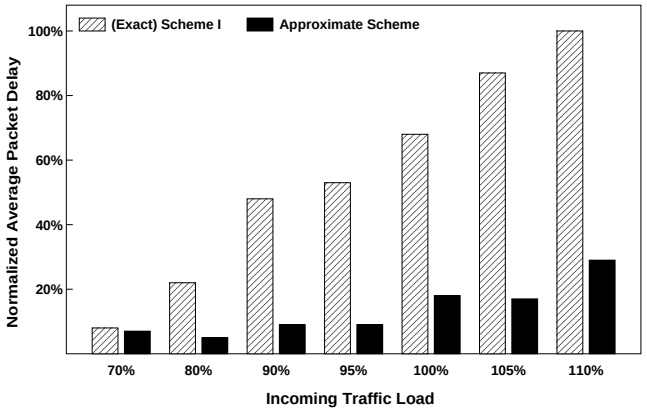


**Figure 10: Average packet delay.**

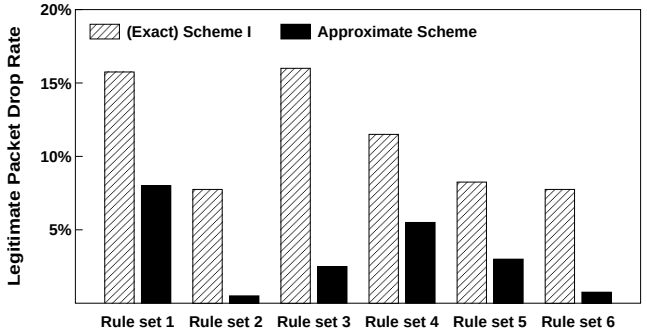


**Figure 11: Legitimate packet drop rate.**

In addition to these packet classification schemes, some researchers have also proposed cache-like techniques to further improve the efficiency of packet classification. For example, two packet cache techniques have been proposed in [13] and [24], respectively. These packet cache techniques cache recently observed packets to speed up the classification of succeeding packets of the same flows. However, the increasingly large number of concurrent flows witnessed by routers/firewalls present serious threat to the performance of packet cache schemes. Based on the notion of rule evolution, Dong *et al.* propose smart rule cache [32], which is able to handle many more concurrent flows, requires much smaller cache size and delivers much higher cache hit ratios.

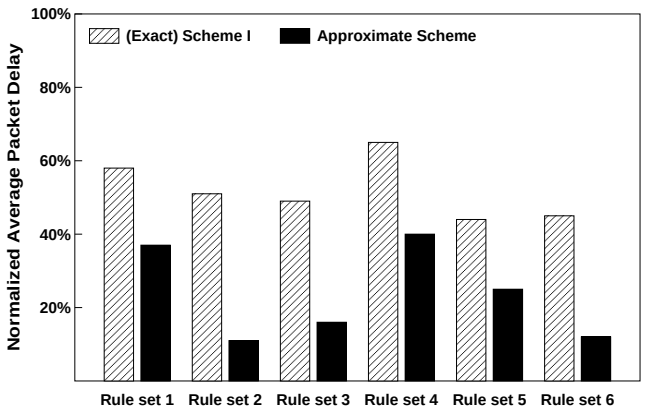


**Figure 12: Average packet delay.**

The idea of speeding up packet classification by first checking "hot" rules (like smart rule cache) has also been explored by other researchers as well. In [28], Cohen and Lund propose to reorder rules based on popularity. Although their goal is to reduce the expected time of sequentially searching through a rule set to classify packets, this technique can actually be used to reorder rules and then cache the hottest rules. In that sense, their proposal shares some common observation with smart rule cache. However, simply reordering given rules has limited effectiveness. For one example, it is often the case that the last default rule matches a significant portion of incoming traffic. Therefore, to preserve semantic integrity, the default rule cannot be placed before any other rule that has a different decision, because the default rule overlaps and hence conflicts with such a rule. In smart rule cache, rules in cache are not necessarily present in the original rule set and dynamically evolve in response to incoming traffic pattern changes. Use of such independently defined and constantly evolving rules is decisive to the success of smart rule cache.

More recently, Hamed *et al.* [31] propose to add some "early reject" rules to the beginning of firewall rule sets, in pursuit of the same goal of reducing the expected time needed to sequentially search through a rule set. Compared with the proposal by Cohen and Lund, Hamed *et al.* have gone one step further in that the early reject rules they add are not necessarily in the original rule set. However, the key idea of dynamically evolving rules is still absent. Moreover, in identifying early reject rules, they have not been able to take a systematic approach based on the semantics of the original rule set. This greatly limits the flexibility and effectiveness of added early reject rules.

Almost all of these previous proposals are constrained in the domain of exact packet classification, where every packet must be correctly classified based on the

original rule set. The only exception is [24], in which the authors propose approximate cache using bloom filters [1]. More recently, Bonomi *et al.* have proposed approximate state machines, which employ bloom filters to achieve increased efficiency. However, mistakenly accepting unwanted packets, especially those malicious attacking packets, is not acceptable in robust firewalls. Bloom filters cannot avoid such false positives, and hence are not appropriate for use in robust firewalls. The reason is briefly explained as follows. For more information about bloom filters, interested readers are referred to [1].

1. As a data structure for membership checking, false positive is inherent in bloom filters. If we define packets present in a bloom filter to be legitimate packets, unwanted packets may be mistakenly determined to be present in the bloom filter and hence interpreted as legitimate packets.

2. Even if we define packets present in a bloom filter to be unwanted packets, we are still not able to avoid accepting unwanted packets. Because if an unwanted packet has never been seen before, it is likely to be not present in the bloom filter and hence interpreted as a legitimate packet.

Our proposed approximate packet classification scheme, therefore, is the first known applicable technique for approximate packet classification that can avoid mistakenly accepting unwanted packets, which is of vital importance in robust firewalls.

## 5. CONCLUSIONS

Given the increase in unwanted traffic, firewalls in networks and enterprises are getting configured with increasing number of rules, each encoding more complex specifications than before. Additionally, as wire speeds continue to increase at a more rapid rate than memory access latencies, software based packet classification systems are more fragile against conditions of traffic overload. To improve the performance of such classification systems, we propose to employ approximate packet classification for increased robustness. Our ideas have been specifically designed for stateless systems. However, we believe the concept itself is fairly general, and may find different uses in the context of more stateful classification systems.

## 6. REFERENCES


[1] B. Bloom. Space/time tradeoffs in in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[2] W. Leland, M. Taqqu, W. Willinger, and D. Wilson. On the self-similar nature of ethernet traffic. *IEEE/ACM Transactions on Networking*, 2(1):1–15, February 1994.

[3] R. K. Montoye. Apparatus for storing "don't care" in a content addressable memory cell. United States Patent 5,319,590, June 1994.

[4] M. H. Overmars and A. F. van der Stappen. Range searching and point location among fat objects. *Journal of Algorithms*, 21(3):629–656, November 1996.

[5] T. Lakshman and D. Stiliadis. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. In *ACM SIGCOMM*, September 1998.

[6] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and scalable layer four switching. In *ACM SIGCOMM*, pages 191–202, September 1998.

[7] R. A. Kempke and A. J. McAuley. Ternary CAM memory architecture and methodology. United States Patent 5,841,874, November 1998.

[8] P. Gupta and N. McKeown. Packet classification on multiple fields. In *ACM SIGCOMM*, 1999.

[9] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Packet classification using tuple space search. In *ACM SIGCOMM*, 1999.

[10] P. Gupta and N. McKeown. Packet classification using hierarchical intelligent cuttings. In *Hot Interconnects*, 1999.

[11] G. Gibson, F. Shafai, and J. Podaima. Content addressable memory storage device. United States Patent 6,044,005, March 2000.

[12] T. Y. Woo. A modular approach to packet classification: Algorithms and results. In *IEEE INFOCOM*, 2000.

[13] J. Xu, M. Singhal, and J. Degroat. A novel cache architecture to support layer-four packet classfication at memory access speeds. In *IEEE INFOCOM*, 2000.

[14] F. Baboescu and G. Varghese. Scalable packet classification. In *ACM SIGCOMM*, 2001.

[15] L. Qiu, G. Varghese, and S. Suri. Fast firewall implemention for software and hardware based routers. In *IEEE ICNP*, 2001.

[16] C. Estan and G. Varghese. New directions in traffic measurement and accounting. In *ACM SIGCOMM*, 2002.

[17] H. Liu. Efficient mapping of range classifier into Ternary-CAM. In *Hot Interconnects*, 2002.

[18] J. van Lunteren and T. Engbersen. Fast and scalable packet classification. *IEEE Journal on Selected Areas in Communications*, 21(4):560–571, 2003.

[19] F. Baboescu, S. Singh, and G. Varghese. Packet classification for core routers: is there an alternative to CAMs? In *IEEE INFOCOM*, 2003.

[20] F. Zane, G. Narlikar, and A. Basu. Coolcams: Power-efficient tcams for forwarding engines. In *IEEE INFOCOM*, 2003.

[21] S. Singh, F. Baboescu, G. Varghese, and J. Wang. Packet classification using multidimensional cutting. In *ACM SIGCOMM*, 2003.

[22] M. E. Kounavis, A. Kumar, H. Vin, R. Yavatkar, and A. T. Campbell. Directions in packet classification for network processors. In *NP2 Workshop*, 2003.

[23] E. Spitznagel, D. Taylor, and J. Turner. Packet classification using extended tcams. In *ICNP*, 2003.

[24] F. Chang, W. C. Feng, and K. Li. Approximate caches for packet classification. In *IEEE INFOCOM*, 2004.

[25] F. Yu and R. H. Katz. Efficient multi-match packet classification with TCAM. In *Hot Interconnects*, 2004.

[26] A. X. Liu and M. G. Gouda. Removing redundancy from packet classifiers. Technical Report TR-04-26, Department of Computer Sciences, The University of Texas at Austin, June 2004.

[27] D. E. Taylor and J. S. Turner. Scalable packet classification using distributed crossproducting of field labels. In *IEEE INFOCOM*, 2005.

[28] E. Cohen and C. Lund. Packet classification in large ISPs: Design and evaluation of decision tree classifiers. In *ACM SIGMETRICS*, 2005.

[29] K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary. Algorithms for advanced packet classification with Ternary CAMs. In *ACM SIGCOMM*, 2005.

[30] M. G. Gouda and A. X. Liu. A model of stateful firewalls and its properties. In *DSN*, 2005.

[31] H. Hamed, A. El-Atawy, and E. Al-Shaer. Adaptive statistical optimization techniques for firewall packet filtering. In *IEEE INFOCOM*, 2006.

[32] Q. Dong, S. Banerjee, J. Wang, D. Agrawal, and A. Shukla. Packet classifiers in ternary CAMs can be smaller. In *ACM SIGMETRICS*, 2006.

[33] Citation deleted to double blind review. A copy of this paper (under submission to Sigmetrics 2007) has been sent to the Program Chairs of Sigcomm 2007 with author names removed. If useful, please request the Program Chairs for this paper.

[34] F. Bonomi, M. Mitzenmacher, R. Panigrah, S. Singh and G. Varghese. Beyond bloom filters: from approximate membership checks to approximate state machines. In *ACM SIGCOMM*, 2006.