A SET-BASED APPROACH TO PACKET CLASSIFICATION

Venkatesh Prasad Ranganath, Daniel Andresen
Department of Computing and Information Sciences
Kansas State University
{rvprasad, dan}@cis.ksu.edu

ABSTRACT
Firewalls, and packet classification in general, are becoming more and more significant as data rates soar and hackers become increasingly sophisticated - and more forceful. In this paper, we present a new packet-classification approach that uses *set theory* to classify packets. This approach has significant theoretical advantages over current approaches. We demonstrate its practicality by implementing a firewall subsystem in Linux which approaches the performance of today's naive packet-filtering implementations.

KEY WORDS
TCP/IP, firewalls, Linux, iptables, packet filtering

1 Introduction

As the use of Internet becomes ubiquitous in day-to-day life, the protection of information and availability of services via the Internet assumes utmost importance. The protection of information on the wire and the control of access to it is addressed by various authentication mechanisms and secure protocols such as SSH and HTTPS. However, these protocols/mechanisms cannot alone protect the information. It is still possible for a malicious user to flush a service with spurious requests to create a "denial of service." Although the benefit of such an act is unknown, it is harmful to the business entity providing the service, as it causes inconvenience to the real users of the service.

The above issue results from the lack of access control to the service. The *firewall* is a concept that addresses this issue by controlling the connections to the host by which services can be accessed. In a simple scenario, one can configure the firewall to deny access to a particular service when the access is initiated from a particular network domain. To deny such access, messages arriving at the host need to be classified as arriving from the given domain. Such classification of messages has been a longterm research interest of the network community under the name of *"packet classification"* and has uses beyond firewalls.

Due to the increase in internal and external security threats, as well as the exponentially-increasing data rates, both the number of classification rules in firewalls and the demand for better performance are likely to grow rapidly. In this paper, we present a new packet-classification approach that uses *set theory* to classify packets. We will demonstrate its application in implementing a firewall subsystem in Linux.

Detailed information about packet classification and how it is used to realize a firewall is presented in Section 2. We also present the packet classification as implemented in some of the prevalent firewall subsystems in Linux. In Section 3, we present the set-based packet classification in theory and implementation. This is followed by the results of the experiments we conducted to compare performance of `Iptables`[1], a common firewall subsystem for Linux, to the firewall subsystem implemented using the proposed packet classification algorithm [1, 4]. We will present our conclusions and future work to extend/improve this algorithm in Section 5.

2 Background

As introduced in the previous section, *packet classification* is a general concept of classifying a packet arriving at a network node based on certain prespecified criteria. The criteria depend on the purpose for which the packet is being classified. In the case of packet classification at routers, the criteria would classify a packet to channel it to the interface through which it can reach its destination. The criteria can be further refined to improve reliability, improve performance, decrease number of hops, etc. A naive but important application would also be to collect accounting information that can be used to restructure the network or dynamically configure the network to improve throughput.

The main concerns in packet classification are time taken to classify, cost of classification (CPU cycles), storage space required by the criteria, and scalability of the classification algorithm. Other minor concerns

[1]Iptables is available at http://www.netfilter.org.

are ease of specification of the criteria and the cost to modify the set of criteria. The priority of the concerns mentioned above is determined by the actual application of packet classification. Hence, the packet classification algorithm that addresses the most important concern best should be chosen to ensure better performance. Please refer to [3] for a description of various types of packet-classification algorithms and the situation in which they are most applicable.

In firewalls, packet classification occurs based on a set of *rules* satisfied by a packet. A *rule* in a firewall describes a set of attributes of a packet. These attributes may be the originating IP address, the source port, the protocol, etc. A packet satisfies, triggers, or *matches* a rule when it possesses attributes as described by the rule. On such an occasion, the firewall subsystem executes the action or *target* associated with the rule. This action or target can determine if the packet should be let through the firewall, dropped, transformed, logged, etc. In short, the action/target determines the fate of the packet. All of the concerns mentioned above for packet classification are to be addressed in the same order when classifying packets in a firewall [2, 4].

There are many commercial as well as free implementations of firewall subsystems for various operating systems. For the purpose of presenting our ideas, we will use `Iptables`, a free implementation on Linux, as the reference implementation in our paper.

`Iptables` is built on `netfilter` framework available in Linux [6, 5, 7]. This framework divides the possible network paths inside a node into five segments. It then provides hooks to attach external functions at any of these segments. When a packet arrives at a segment, the hook functions are executed in the order they were registered. Each hook function is required to provide a "*verdict*" from a set of verdicts predefined in this framework. Only if the verdict of the current hook function is to let the packet continue its journey will the next hook function be executed and likewise the next segment be traversed.

Each rule in `Iptables` is comprised of a set of *matches* and a *target*, which is executed if all the matches are matched. While matching a rule, the matches are traversed in the specified order. Each set of rules is grouped into a *chain* that corresponds to a segment mentioned above. The rules in a chain are executed in the order they were added to the chain. If a packet satisfies a rule and the verdict of the rule is one of the "netfilter" verdicts (as specified by `netfilter`) then the traversal of rules is stopped and the verdict is returned. If not, the following rules in the chain are traversed till such a verdict is encountered. If no matched rule provides a "netfilter" verdict, then the default verdict (*policy*) associated with the chain is returned. It is possible for a matched rule to produce a non-netfilter verdict that is local to `Iptables`. In such cases, the traversal of the rules continues with the following rule or the specified rule.

A set of chains is grouped as a *table*. Each table represents a particular application of packet classification. `filter` table represents the application of packet classification for the purpose of *packet filtering*. This realizes the main purpose of a firewall: to control the connections to the node. Other tables such as `nat` and `mangle` realize *network address translation* and specialized packet alterations, respectively.

In terms of filtering algorithm, `Iptables` uses a "first-match" policy during packet filtering. On the other hand, `Packet Filter(PF)`[2], which is the firewall subsystem in the OpenBSD OS, uses "last-match" as the default policy for packet filtering. However, the same set of rules can be ordered suitably to achieve the same effect in both systems. The worst-case execution complexity of both subsystems is in the order of $O(n)$, where n is the number of executed matches. `nf-hipac`[3], a recent firewall implementation in Linux, uses B-trees to store the rules as opposed to linear lists of rules in `Iptables` and `PF`. Hence, it provides better performance in terms of throughput. As very little documentation about the implementation is available, based on the data structure used to store the rules, we speculate that the worst-case performance will be in the order of $O(\log_m n)$, where n is the largest number of matches in a rule in the given rule set and m is the arity of the B-tree.

It is interesting that none of these systems allow disjunctive matches of rule, i.e. all of the rules that the packet matches. The only issue in such matching is that the rules matched may have a contradicting verdict. However, this issue can be addressed by imposing a total ordering on the possible verdicts.

3 Algorithm

In this section we shall present the motivation for a new approach to packet classification, followed by the theoretical and implementation detail of the approach. To make the presentation concrete and simple, we will present the packet-classification approach as applicable to firewalls based on IP protocol.

3.1 Motivation

Most of the prevalent firewall subsystems traverse the rule sets in a linear fashion, i.e. each match (sub-criteria) is matched in the specified order as each rule (criteria) is traversed in the order it was added to the firewall. However, this is not the best approach, as

[2]PF is available at http://www.benzedrine.org.

[3]nf-hipac is available at http://www.hipac.org.

there may be more than one rule, say a and b, in the rule set with an identical match (sub-criteria) and although the packet failed to match the rule a due to this match, the packet will still be matched via rule b. A concrete example would be the rule sets given below. The first rule says "*accept any TCP packet originating from 192.168.1.1.*" Likewise, the second rule says, "*drop any TCP packet originating from 129.130.1.1.*"

```
-p tcp -s 192.168.1.1 -j ACCEPT
-p tcp -s 129.130.1.1 -j DROP
```

Both the rules will be matched to classify a UDP packet, even though upon matching the first rule it can be decided that any following rules in which TCP is specified as the protocol should not be considered for classifying the current packet. This happens because the relationship between the given rules based on the relation between the constituent matches is not captured and utilized during packet classification.

3.2 Theory

The above situation can be remedied by identifying properties of various match sorts in the system and establishing relationships between them based on their properties. These relationships can then be used to separate the rules based on their constituent matches and their mutual relationship.

To be more precise, for each match sort, M_i, a set of properties, $\{P_{i1}, P_{i2}, \ldots P_{in}\}$, and a set of relations, $\{R_{ij} : P_{ij} \times P_{ij} \rightarrow V\}$, that relate values in the domain of property P_{ij} can be defined. V is a set of possible set theoretical relations such as *mutually exclusive*, *proper subset*, etc. Given this information, it is possible to arrive at the set theoretical relation between various matches, hence, at a set theoretical relation between the rules of a rule set.

At this point, an interesting question would be that whether it is always possible to relate two rules via these relations. The answer would be no if there is no ordering between the properties. However, in the presence of such ordering with suitable characteristics, it should be possible to always relate 2 rules via these relations. We will not address these issues any further in this paper.

A concrete example would be in IP packet filtering. *IP* can be considered as a match sort. Given that, *transport layer protocols* layered on top of IP can be considered as a property of *IP* match sort. In an implementation, the domain of this property can be restricted to {TCP, UDP, ICMP}. These protocols involve ports and some protocol-level information which can be considered for packet classification purposes. Hence, *TCP, UDP,* and *ICMP* can be considered as match sorts with suitable properties. According to

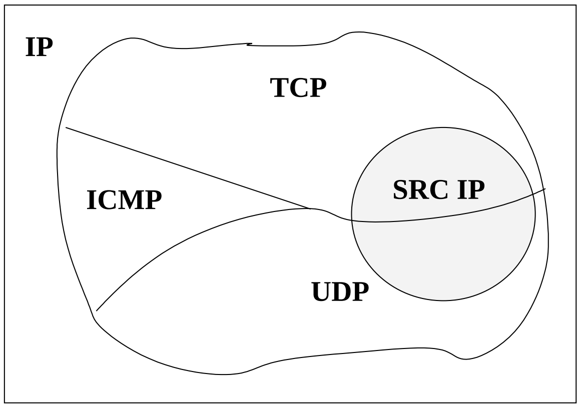

Figure 1. Venn diagram illustrating the match sorts and the set theoretical relation between them.

TCP/IP standards, it is known that a packet cannot belong to more than one transport-layer protocol. Given this information, it is trivial to conclude that the match sorts *TCP, UDP,* and *ICMP* are mutually exclusive. In terms of set theoretic relations, $\mathrm{TCP} \cap \mathrm{UDP} = \emptyset$. This information can be cast as a membership test during packet classification to decide if the match (and hence the containing rule) needs to be considered to classify the packet. If a packet fails to pass the membership test in one rule, it will do so in all the rules with a match from the same match sort. Hence, all such rules need not be considered to classify the packet.

In terms of complexity, if the rules against which the packet needs to be matched are picked based on the properties of the match sorts, then it would reduce the total number of rules considered to classify the packet. Also, the time taken to arrive at the rule set to be considered for classification bears direct impact on the classification time. It is possible to structure the match sorts in hierarchies, hence rendering it to be represented as a tree. Such representation will have a time complexity of the order of $O(\log n)$, where n is the height of the hierarchy, to arrive at the rule set to be considered for classification.

3.3 Implementation

We have implemented a firewall subsystem for Linux called `ipt-ksu` based on the proposed approach for packet classification. In our implementation, we maintain a list of chains that contain a list of rules in the specified order. Each chain also maintains a list of match sort. Each match sort in this list maintains a list of references to the rules in its parent chain. A reference exists if the target rule contains an instance of the parent match sort.

When the rules are added to the chain, we check if there is a rule which subsumes the given rule. This is done in order to prune unnecessary rules. After the rule has been added to the firewall, we reorganize the match sort list in the affected chain. During reorganization, we associate a value, "elimination factor," with each match sort in the list of match sorts. This normalized positive integer indicates the number of rules that will be eliminated for the purpose of classification in case the corresponding match in a rule referred by the considered match sort is satisfied. For example, an elimination factor of two indicates that at least two rules will be eliminated in case a match occurs.

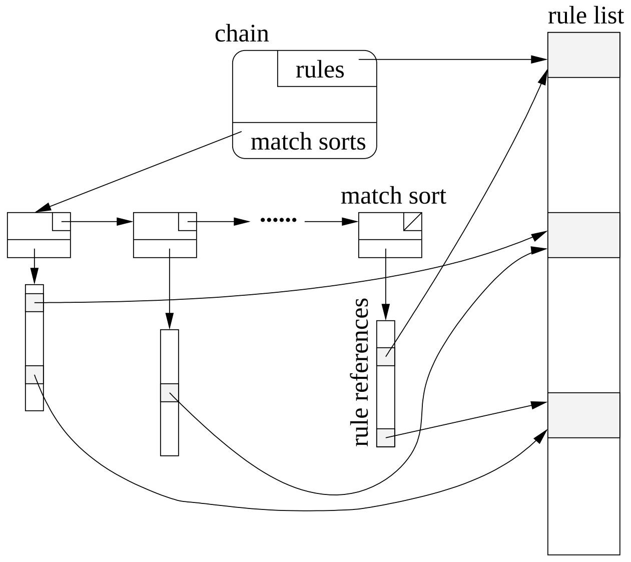

Figure 2. Data structures in our implementation of the firewall subsystem on Linux.

When a packet arrives for classification, each match sort in the chain is traversed in the decreasing order of the elimination factor associated with the match sort. If the packet fails the membership test of the match sort, all the rules referred to in the match sort are marked as eliminated for packet classification. If not, the packet is matched against the corresponding match in each of the referred rules. If it does not match, then the rule is marked as eliminated. After all the match sorts have been traversed, the targets of all rules left unmarked are executed. It is possible to have unmarked rules with contradicting targets. In such cases, as mentioned previously, a total ordering on the results of targets can determine the result of packet classification.

The asymptotic complexity of our implementation will be linear in the number of rules. Unlike `iptables`, the order of the rules does not alter its performance, but the composition of the rules (in terms of the matches) impacts its performance.

4 Experimental results

Our experimental setup was comprised of the firewall subsystem built into Linux 2.4.21, running on a 2.5GHz Pentium 4 with 472MB of RAM and a 10Mbps ethernet card. The firewall was set up to filter incoming packets with varying size of rule sets (200, 1000, and 2500 rules) using both our implementation of firewall (`ipt-ksu`) and `iptables` v1.2.6. The rule sets were structured in two flavors. The first flavor called "harsh" rule set, would force `iptables` to traverse all the rules in the rule set by having the most general rules at end of the rule set. The second flavor, called "kind" rule set, would have the most general rules at the beginning of the rule set, thereby forcing the matches to happen early in case of iptables.

The tests were instrumented by using `netperf` v2.2p14. The Netperf client was run on the system running the firewall with varying packet sizes (128, 512, and 1518 bytes). The Netperf client was instructed to contact the local netperf server as well as the remote Netperf server in various runs. A comparison of the results obtained from one such test run for `iptables` and `ipt-ksu` is illustrated in Figure 3.

The results of a few of the various test runs are presented in Table 1 and it clearly indicates that `ipt-ksu` fails to outperform `iptables`, which uses a naive packet classification algorithm. However, a mere comparison of implementation strategy shows that in case of `iptables` the functions that are used to match IPT, TCP, and UDP properties happen to occur in the main firewall module, whereas in case of `ipt-ksu` these functions occur in different modules. This alone can contribute up to 5% overhead in terms of function calls across different kernel modules. The rationale behind this design decision was to keep the implementation simple, modular, and uniform. Also, a linear approach was used to realize set operations in `ipt-ksu.` A different approach based on tree structures will improve the performance.

As part of this implementation we also implemented, an algorithm in the kernel space to check if a rule being added will be subsumed by an existing rule. This algorithm aids in improving the performance of `ipt-ksu` in the case of "kind" rule set by eliminating many rules which will be subsumed by the general rules entered earlier on. As the asymptotic complexity of this algorithm is in the order of $O(n^2)$, where n is the number of rules, it largely impacts the time taken to set up the rule list.

5 Future Work and Conclusion

Following is a non-exhaustive list of work or ideas that can improve the current implementation and make the

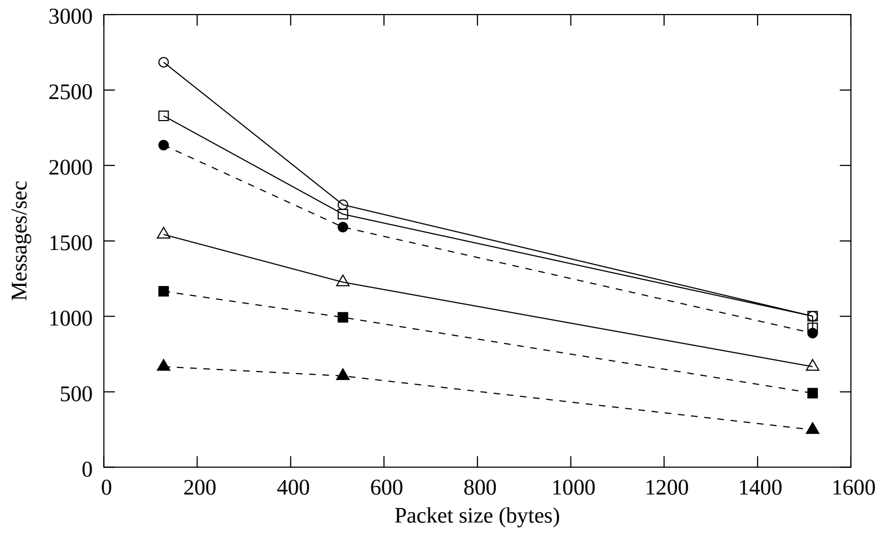

Figure 3. This illustrates the performance of `iptables` (solid lines/unfilled points) and `ipt-ksu` (dashed lines/filled points) with varying number of rules measured in terms of the number of messages handled per second. Circular, square, and triangular points indicate data corresponding to 200, 1000, and 2500 rules, respectively.

Netperf test	Rule set flavor	Firewall	Packet Size (Bytes)	Number of rules		
				200	1000	2500
TCP_RR	Kind	`iptables`	128	3013	2969	2891
			1518	1054	1057	1036
		`ipt-ksu`	128	2214	1213	669
			1518	902	490	247
	Harsh	`iptables`	128	2684	2328	1542
			1518	1000	922	666
		`ipt-ksu`	128	2135	1165	666
			1518	888	490	248
TCP_STREAM	Kind	`iptables`	128	87447	87085	87326
			1518	7377	7373	7372
		`ipt-ksu`	128	69345	21015	7484
			1518	7370	1961	802
	Harsh	`iptables`	128	81796	77310	33199
			1518	7195	7369	3162
		`ipt-ksu`	128	69002	20824	7378
			1518	7368	1923	800

Table 1. The data depicts the performance of `iptables` and `ipt-ksu` over various runs with varying test cases, flavors of rule sets, and number of rules measured in terms of messages handled per second. (Some intermediate data points were skipped to keep the table short.)

proposed approach more attractive.

1. It has been our experience during test runs that different orderings of matches in rules affect the performance of various firewall subsystems. For this reason, the existing implementation of `ipt-ksu` needs to be extended to include more matches and targets as provided by `iptables`. This will provide a richer rule set in terms of the matches, thus enabling more elaborate testing. The results from the testing will aid us to arrive at a crisper comparison of the firewall subsystems, hence determining the applicability of the proposed algorithm.

2. The study of the disparity in data collected for `iptables` and `ipt-ksu` has revealed some serious limitations of the implementation of `ipt-ksu`. Remedying these limitations should improve performance and enable more interesting comparison of the proposed approach with existing approaches to packet classification.

3. As mentioned before, at present the rule subsumption is being calculated in kernel space. However, as this is a one-time calculation for a given rule set, it can be performed offline in user space to arrive at the least subset of the given rule set which can then be set up in kernel space with no checking. This will improve the rule set up time by a large factor. This also creates opportunities for various static analyses to be conducted on the rule set to further improve the performance of packet filtering. These techniques can be implemented and used independent of the underlying firewall subsystems.

4. It would be an interesting exercise to recast the current `ipt-ksu` implementation to work off tree structures that capture the hierarchical relation between various match sorts. This should provide interesting data to compare the new approach to that implemented in `hipac`.

5. The division of criteria into various sorts affects the application of packet classification. Also, the choice of sorts is dependent on the environment in which packet classification is used. Hence, it would be beneficial as a first step to provide features to specify a firewall in such detail. Later on this can be generalized to packet classification.

We have contributed a new approach to packet classification based on relations in set theory, with significant theoretical advantages over current approaches. We have applied the same approach to packet filtering by implementing it as part of a new firewall subsystem in Linux. We have conducted experiments to study its impact on the performance of packet filtering. As illustrated in Section 4, the current implementation falls short of naive packet-filtering implementations. However, in the process, we have discovered several shortcomings in our implementation as listed above, and we plan to address these issues in the future to enable a crisper comparison of the proposed approach with the existing approaches to packet classification.

Acknowledgments

This material is based in part upon work supported by the National Science Foundation under the award numbers CCR-0082667 and ACS-0092839. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

References

[1] David A. Bandel. Taming the wild netfilter. *Linux Journal*, 89:64, 66–68, 70, 72, September 2001.

[2] Mick Bauer. Paranoid penguin: Using iptables for local security. *Linux Journal*, 100:??–??, August 2002.

[3] P. Gupta and N. McKeown. Algorithms for packet classification, 2001.

[4] Duncan Napier. IPTables/NetFilter – Linux's next-generation stateful packet filter. *Sys Admin: The Journal for UNIX Systems Administrators*, 10(12):8, 10, 12, 14, 16, December 2001.

[5] Bryan Pfaffenberger. *Linux networking clearly explained*. Academic Press, New York, NY, USA, 2001.

[6] R. W. Smith. *Advanced Linux Networking*. Addison_Wesley, June 2002.

[7] Robert L. (Robert Loren) Ziegler and Carl B. Constantine. *Linux firewalls*. New Riders Publishing, Carmel, IN, USA, second edition, 2002.