International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 1 Issue 7, September - 2012

Case study and Implementation of Geometric Efficient Matching Algorithm for Firewalls

S.Pradeep — MTech IInd Year
V. Mahesh — Associate Professor
Y. Ramesh Kumar — Head of the Department
Department of Computer Science and Engineering
Avanthi Institute of Engineering & Technology, Cherukupalli, Andhra Pradesh, India

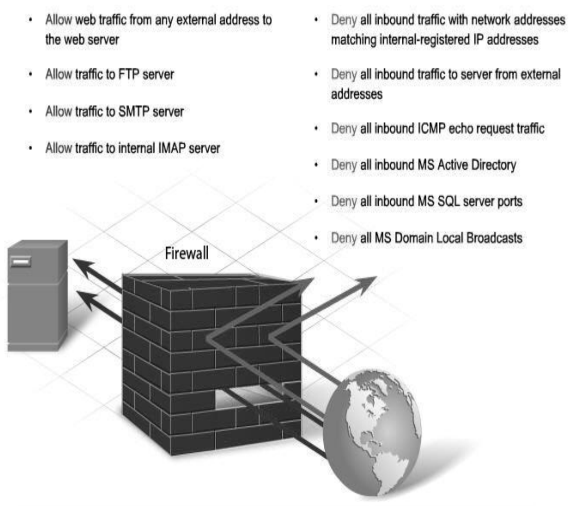
Abstract

A firewall is a device that controls the flow of communications across networks of computers by examining their source, destination and type - and comparing these with predetermined lists of allowed and disallowed transactions. Packet matching in firewalls involves matching on many fields from the TCP and IP packet header. At least five fields (protocol number, source and destination IP addresses, and ports) are involved in the decision which rule applies to a given packet. With available bandwidth increasing rapidly, effective matching algorithms need to be deployed in modern firewalls to ensure that the firewall does not become a bottleneck , Since firewalls need to filter all the traffic crossing the network perimeter, they should be able to sustain a very high throughput. In this paper we consider a classical algorithm that we adapted to the firewall domain. We call the resulting algorithm "Geometric Efficient Matching". The Geometric Efficient Matching algorithm enjoys a logarithmic matching time performance. However, the algorithm's theoretical worst-case space complexity is O (n4) for a rule-base with n rules. Based on statistics from real firewall rule-bases, we created a perimeter rules model that generates random, but non-uniform, rule bases. We evaluated Geometric Efficient Matching algorithm via extensive simulation using the perimeter rules model. Geometric Efficient Matching algorithm speed is far better than the naive linear search algorithms, and it is able to increase the throughput by an order of magnitude.

1. Introduction

A firewall is a part of a computer system or network that is designed to block unauthorized access while permitting outward communication. It is a device or set of devices configured to permit, deny, encrypt, decrypt, or proxy all computer traffic between different security domains based upon a set of rules and other criteria. Firewalls can be implemented in both hardware and software, or a combination of both. Firewalls are frequently used to prevent unauthorized Internet users from accessing private networks connected to the Internet, especially intranets. From the fig1 all messages entering or leaving the intranet pass through the firewall, which examines each message and blocks those that do not meet the specified security criteria.

Figure 1: Simple Firewall with Rules


Allow web traffic from any external address to the web server
Allow traffic to FTP server
Allow traffic to SMTP server
Allow traffic to internal IMAP server
Deny all inbound traffic with network addresses matching internal-registered IP addresses
Deny all inbound traffic to server from external addresses
Deny all inbound ICMP echo request traffic
Deny all inbound MS Active Directory
Deny all inbound MS SQL server ports
Deny all MS Domain Local Broadcasts
Firewall

1.1 Types of Firewalls:

The International Standards Organization (ISO) Open Systems Interconnect (OSI) model for networking defines seven layers, where each layer provides services that ``higher-level" layers depend on. In order from the bottom, these layers are physical, data link, network, transport, session, presentation and application. The important thing to recognize is that the lower-level the forwarding mechanism, the less examination the firewall can perform. Generally speaking, lower-level firewalls are faster, but are easier to fool into doing the wrong thing. The different firewalls can be developed based on the functionality and requirement.

1.2.1 Packet-filtering firewall
Typically is a router with the capability to filter some packet content, such as Layer 3 and sometimes Layer 4 information.

1.2.2 Application gateway firewall (proxy firewall)
A firewall that filters information at Layers 3, 4, 5, and 7 of the OSI reference model. Most of the firewall control and filtering is done in software.

1.2.3 Address-translation firewall
A firewall that expands the number of IP addresses available and hides network addressing design.

1.2.4 Host-based (server and personal) firewall
A PC or server with firewall software running on it.

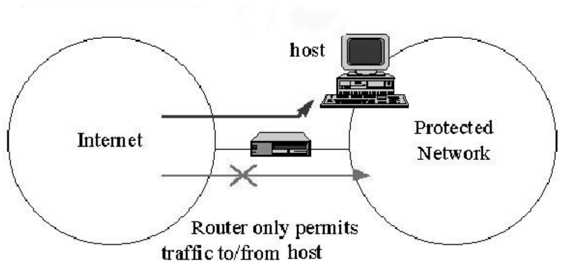
1.2.5 Transparent firewall
A firewall that filters IP traffic between a pair of bridged interfaces.

1.2.6 Hybrid firewall
A firewall that is a combination of the various firewalls types.

1.3 Case Study on Network layer and Application layer Firewalls
For example, an application inspection firewall combines a stateful firewall with an application gateway firewall. Conceptually, there are two types of firewalls: one is Network layer firewalls and second is Application layer firewalls. They are not as different as you might think, and latest technologies are blurring the distinction to the point where it's no longer clear if either one is ``better'' or ``worse.'' As always, you need to be careful to pick the type that meets your needs. It depends on what mechanisms the firewall uses to pass traffic from one security zone to another.

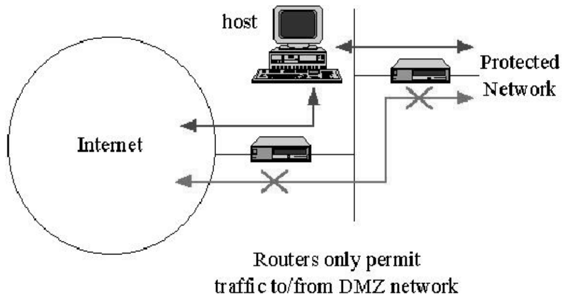
1.3.1 Network layer Firewalls
These generally make their decisions based on the source, destination addresses and ports in individual IP packets. A simple router is the ``traditional'' network layer firewall, since it is not able to make particularly sophisticated decisions about what a packet is actually talking to or where it actually came from. Modern network layer firewalls have become increasingly sophisticated, and now maintain internal information about the state of connections passing through them, the contents of some of the data streams, and so on. One thing that's an important distinction about many network layer firewalls is that they route traffic directly though them, so to use one you either need to have a validly assigned IP address block or to use a ``private internet'' address block . Network layer firewalls tend to be very fast and tend to be very transparent to users.

Figure 2: Screened Host Firewall



In Figure 2, a network layer firewall called a ``screened host firewall'' is represented. In a screened host firewall, access to and from a single host is controlled by means of a router operating at a network layer. The single host is a highly-defended and secured strong-point that (hopefully) can resist attack.

Figure 3: Screened Subnet

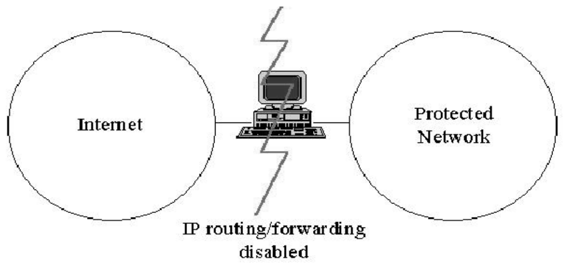


Example Network layer firewall: In figure 3, a network layer firewall called a ``screened subnet firewall'' is represented. In a screened subnet firewall, access to and from a whole network is controlled by means of a router operating at a network layer. It is similar to a screened host, except that it is, effectively, a network of screened hosts.

1.3.2 Application layer Firewalls
These generally are hosts running proxy servers, which permit no traffic directly between networks, and which perform elaborate logging and auditing of traffic passing through them. Since the proxy applications are software components running on the firewall, it is a good place to do lots of logging and access control. Application layer firewalls can be used as network address translators, since traffic goes in one ``side'' and out the other, after having passed through an application that effectively masks the origin of the initiating connection. Having an application in the way in some cases may impact

performance and may make the firewall less transparent. Modern application layer firewalls are often fully transparent. Application layer firewalls tend to provide more detailed audit reports and tend to enforce more conservative security models than network layer firewalls.

Figure 4: Dual Homed Gateway



Example Application layer firewall: In figure 4, an application layer firewall called a ``dual homed gateway" is represented. A dual homed gateway is a highly secured host that runs proxy software. It has two network interfaces, one on each network, and blocks all traffic passing through it. The Future of firewalls lies someplace between network layer firewalls and application layer firewalls. It is likely that network layer firewalls will become increasingly ``aware" of the information going through them, and application layer firewalls will become increasingly ``low level" and transparent. The end result will be a fast packet-screening system that logs and audits data as it passes through. Increasingly, firewalls (network and application layer) incorporate encryption so that they may protect traffic passing between them over the Internet. Firewalls with end-to-end encryption can be used by organizations with multiple points of Internet connectivity to use the Internet as a ``private backbone" without worrying about their data or passwords being sniffed.

1.4 Different Firewall Techniques:

1.4.1 Packet filters: Looks at each packet entering or leaving the network and accepts or rejects it based on user defined rules. Packet filtering is fairly effective and transparent to users, but it is difficult to configure. In addition, it is susceptible to IP spoofing. Packet -filtering firewalls work primarily at the Network Layer of the OSI model. Firewalls are generally considered Layer 3 constructs. However, they permit or deny traffic based on Layer 4 information such as protocol, and source and destination port numbers. Packet filtering uses ACLs to determine whether to permit or deny traffic, based on source and destination IP addresses, protocol, source and destination port numbers, and packet type. Packet-filtering firewalls are usually part of a router firewall.

1.4.2 Application gateway: Applies security mechanisms to specific applications, such as FTP and Telnet servers. This is very effective, but can impose performance degradation.

1.4.3 Circuit-level gateway: Applies security mechanisms when a TCP or UDP connection is established. Once the connection has been made, packets can flow between the hosts without further checking.

1.4.4. Proxy server: Intercepts all messages entering and leaving the network. The proxy server effectively hides the true network addresses

2. Existing Algorithms

Most modern firewalls are stateful. This means that after the first packet in a network flow is allowed to cross the firewall, all subsequent packets belonging to that flow, and especially the return traffic, is also allowed through the firewall. This statefulness has two advantages. First, the administrator does not need to write explicit rules for return traffic—and such return-traffic rules are inherently insecure since they rely on source-port filtering .So stateful firewalls are fundamentally more secure than simpler, stateless, packet filters. Second, state lookup algorithms are typically simpler and faster than rule match algorithms, so statefulness potentially offers important performance advantages.

Firewall statefulness is commonly implemented by two separate search mechanisms: (i) a slow algorithm that implements the “first match” semantics and compares a packet to all the rules, and (ii) a fast state lookup mechanism that checks whether a packet belongs to an existing open flow. In many firewalls, the slow algorithm is a naive linear search of the rule-base, while the state lookup mechanism uses a hash-table or a search-tree. There are strong indications that commercial firewalls use linear search for the slow rule-match as well. Moreover, the standard advice for improving firewall performance, for all vendors, is to place the most popular rules near the top of the rule-base. This advice doesn’t make much sense if the firewall rearranges the rules into a complex search data structure. Note that a stateful firewall’s two-part design provides its highest performance on long TCP connections, for which the fast state lookup mechanism handles most of the packets. However, connectionless UDP and ICMP traffic, and short TCP flows, like those produced in extremely high volume by Distributed Denial of Service attacks, only activate the “slow” algorithm, making it a significant bottleneck. Our main result is that the “slow” algorithm does not need to be slow, even in a software-only implementation running on a general-purpose operating system.

Existing algorithms implement the “longest prefix match” semantics, using several different approaches. The IPL algorithm, which is based on results, divides the search space into elementary intervals by different prefixes for each dimension, and finds the best (longest) match for each such interval.
The note to be made regarding the existing algorithms, there is no secure when the packet sending and time consuming is high.

3 The Algorithm

3.1 Definitions

The firewall packet matching problem finds the first rule that matches a given packet on one or more fields from its header. Every rule consists of set of ranges [li , ri] for i = 1, . . . , d, where each range corresponds to the i-th field in a packet header. The field values are in 0 = li , ri = Ui , where Ui=232 - 1 for IP addresses, Ui = 65535 for port numbers, and Ui = 255 for ICMP message type or code. For notation convenience later on, we assign each of these fields a number, which is also listed in the table.

Table 1
Header field numbering.

number	description	space
0	source IP address	32bit
1	destination IP address	32bit
2	source port number	16bit
3	destination port number	16bit
4	protocol	8bit

• We use =* to denote wildcard: An =* in field i means any value in [0, Ui].
• We are ignoring the action part of the rule (e.g., pass or drop), since we are only interested in the matching algorithm.

3.2 The Sub-Division of Space

In one dimension, each rule defines one range, which divides space into at most 3 parts. It is easy to see that n possibly overlapping rules define a subdivision of one-dimensional space into at most (2n - 1) simple ranges. To each simple range we can assign the number of the winner rule. This is the first rule which covers the simple range. In d-dimensions, we pick one of the axes and project all the rules onto that axis, which gives us a reduction to the previous one- dimension case, with a subdivision of the one dimension into at most (2n - 1) simple ranges. The difference is that each simple range corresponds to a set of rules in (d - 1) dimensions, called active rules. We continue to subdivide the (d - 1) dimensional space recursively. We call each projection onto a new axis a level of the algorithm, thus for a 4- dimensional space algorithm we have 4 levels of subdivisions. The last level is exactly a one-dimensional case—among all the active rules, only the winner rule matters. At this point we have a subdivision of d-dimensional space into simple hyper-rectangles, each corresponding to single winning rule. we shall see how to efficiently create this subdivision of d-dimensional space, and how it translates into an efficient search structure.

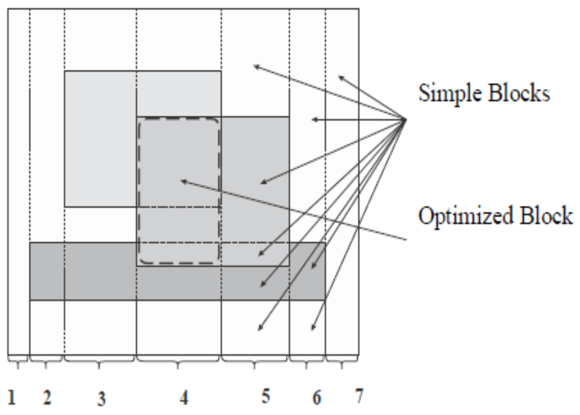
3.3 Dealing with Protocol Field

Before delving into the details of the search data structure, we first consider the protocol header field. The protocol field is different from the other four fields: very few of the 256 possible values are in use, and it makes little sense to define a numerical range of protocol values. This intuition is validated by the data gathered from real firewalls .The only values we saw in the protocol field in actual firewall rules were those of specific protocols, plus the wildcard *, but never a non-trivial range.

3.4 The Data Structure

The GEM search data structure consists of three parts. The first part is an array of pointers, one for each protocol number, along with a cell for the * protocol. We build the second and third parts of the search data structure for each protocol separately.
The second part is a protocol database header, which contains information about the order of data structure levels. The order in which the fields of packet header are checked is encoded as a 4-tuple of field numbers, using the numbering of Table 1. The protocol database header also contains the pointer to the first level and the number of simple ranges in that level.
The third part represents the levels of data structure them- selves. Every level is a set of nodes, where each node is an array. Each array cell specifies a simple range, and contains a pointer to the next level node. In the last level the simple range information contains the number of the winner rule instead of the pointer to the next level. See Figure 6 for an illustration

Figure 5: Search Data Structure

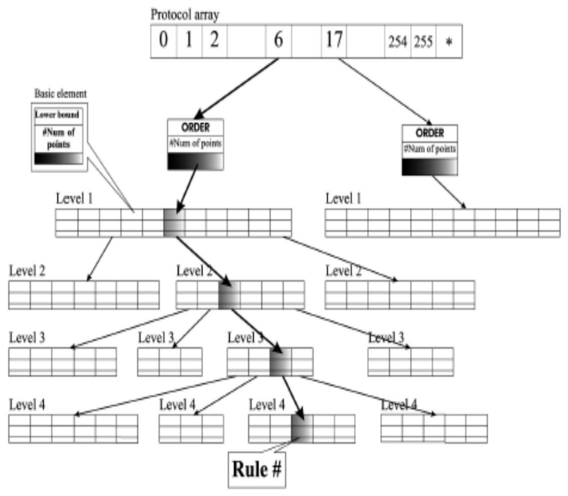


The basic cell in our data structure (i.e., an entry in the sorted array which is a node in the structure) has a size of 12 bytes: 4 for the value of the left boundary of the range, 4 for the pointer to the next level, and 4 for the number of cells in the next-level node. The nodes at the deepest level are slightly different, consisting of only 8 bytes: 4 for the left boundary of the range and 4 for the number of winner rule. Note that the order of levels is encoded in the protocol database header, which gives us convenient control over the field evaluation order.

3.5 The Search Algorithm

The packet header contains the protocol number, source and destination address and port numbers fields. First, we check the protocol field and go to the protocol array of the search .The last two levels of building the search data structure. At this point the rules are two-dimensional, e.g., the X axis may represent the destination IP and the Y axis is the destination port. We can see three rules, shown as shaded overlapping rectangles, plus the default rule in white. The critical points and simple ranges are projected onto the X axis. Three blocks in rule 1 are optimized. For example, suppose we have an incoming TCP packet. Assume that the GEM protocol header for TCP shows that the order of levels is 1203. The first level - 1 - denotes the destination address. We execute a binary search of the destination address value from packet header against the values of the array in the first level. The simple range associated with the found array item points us to the corresponding node from the second level. The second level, in our example (2) denotes the source port number. By binary search on the second level array we find a new simple range, which contains the packet source port number. Similarly, we search for the source address (field 0) and destination port (field 3). In the last level node we find the winner rule information. We repeat the search procedure for protocol *, and get another winner rule. From the two candidates we choose the one with the lower rule number.

Figure 6: GEM Data Structure Overview



3.6 The Build Algorithm

The build algorithm is executed once for each protocol. The input to the build algorithm consists of the rule- base, plus the field order to use. The order dictates the contents of each data structure level, and also, the order in which the header fields will be tested by the search algorithm. There are 4! = 24 possible orders we can choose from, to check 4 fields. The data structure is built using a geometric sweep-line algorithm. All four levels of the search data structure are built in the same manner. We start with the set of active rules from the previous level. For the first level all the rules with the specified protocol (e.g., TCP) are active. We then construct the set of critical points of this level— these are the endpoints of the ranges, which are the projections of the active rules onto the axis that corresponds to the currently checked field. For example, if the first field is .1. (Destination IP address), then the critical points are all the IP addresses that start or end a destination IP address range in any rule. We sort the list of critical points in increasing order, and run the sweep-line over them. Note that there are two implicit critical points: 0, and the maximal value for the level. Every critical point corresponds to a start of one simple range, which in turn relates to a subset of active rules. For each simple range we calculate its set of active rules, by choosing all the rules that overlap the simple range in the current field. For example, in Fig 5, rules 2, 3 and 4 are relevant for the third simple range on the X axis. With this new set of active rules we continue to the next level for each one of the simple ranges. In the deepest level we only need to list the number of the .winner rule.: the rule with lowest number among the active rules associated with the current range.

Table 2

SOURCE PORT DISTRIBUTION	
*	98%
ranges	1%
single port	1%

PROTOCOL DISTRIBUTION	
*	6%
TCP	75%
UDP	14%
ICMP	4%
Other	1%

DESTINATION PORT DISTRIBUTION		
*	0%	
ranges	4%	
average range size	27030	
single ports	96%	
average # single ports per rule-base	50	
most referred-to ports in rules	80	6.89%
	21	5.65%
	23	4.87%
	443	3.90%
	8080	2.25%
	139	2.16%

3.7 Reducing the Space Usage: Basic Optimizations

A space complexity of O (n) ^{4} may be theoretically acceptable since it is polynomial. However, with n reaching thousands of rules, conserving space is crucial. Here, we introduce two optimization heuristics, which significantly reduce GEM's space requirement. The first optimization works on the last level of the data structure. If we take a closer look at last-level ranges, we see that occasionally two or more adjacent ranges point to the same "winner" rule. This means that we can replace all these ranges with a single range which is their geometrical union. The second optimization works on the one-before-last level of the search data structure. Occasionally, there exist simple ranges that point to equivalent last level structures. Instead of storing the same last-level structure multiple times, we keep a single last-level structure, and replace the duplicates by pointers to the main copy. For example, in Fig. 5, ranges 2 and 6 are equivalent (rules 4-3-4, with boundaries in the same vertical positions). As part of the simulation study, we tested the effectiveness of these optimizations. Our simulations on rule bases of sizes from 500 to 10,000 shows that the optimizations reduce the search data structure size by 30 to 60 percent on average, and that the effect grows with rule-base size. We also tried to apply this optimization method on the higher levels of our data structure, but found that this greatly increases the preprocessing time, and only give minor improvements to the space complexity. We remark that additional optimization techniques for GEM-like data structures are known to perform well in the computational geometry literature; hence, it would be interesting to test their effectiveness in the firewall matching domain. Possibilities include: not using the same field ordering in every branch of the search tree; switching to the next branch before completing the search along an axis; or even replacing the last two levels of binary search tree with a data structure optimized for 2D queries such as that of [11] or [4].

3.8 Firewall Rule Base Statistics

To get a better understanding of what real- life firewall rule- bases look like, we gathered statistics from firewall rule-bases that were analyzed by the Firewall Analyzer. The statistics are based on 19 rule-bases from enterprise is firewalls collected during 2001 and 2002. The rule-bases came from a variety of corporations from the financial, telecommunications, automotive, and Pharmaceutical industries. We analyzed a total of 8434 rules. Table 2 shows the distribution of protocols in the rules we analyzed. The data shows that 75% of rules from typical firewall rule-bases match TCP, and a total of 93% match TCP, UDP or ICMP. Of these the most important is clearly TCP. Therefore, we concentrate on these protocols in the rest of paper. In our problem context, these protocols are the most difficult for evaluation since they imply a 4dimensional space. The same table shows the distribution of TCP source and destination port numbers. We can clearly see that the source port number is rarely specified: 98% of the rules have a wildcard =`*` in the source port. This makes sense because both PIX and FireWall-1 are stateful firewalls that do not need to perform source-port filtering to allow return traffic through the firewall and source port data is generally unreliable because it is usually under the control of the attacker. On the other hand, the TCP destination port is usually specified precisely. The vast majority of rules specified a single port number, but 4% allowed a range of ports, and the ranges tended to be quite large. Common ranges are .all high ports. (1024–65535) and .X11 ports. (6000-6003).The single port numbers we encountered were distributed among some 200 numbers, the most popular of which are shown in Table 2: these correspond to the HTTP, FTP, Telnet, HTTPS, HTTP- Proxy, and NetBIOS services.

4 The Simulation Study

4.1 The Random Rules Simulation

As the first step of our performance evaluation of GEM we implemented and tested it in isolation. The GEM builds and search algorithms were implemented in Microsoft Visual Studio. The simulations were per formed on a 2.20 GHz Intel Dual with PC with 1 GB of RAM running the Windows XP Service Pack 2 Operating system. We started by testing GEM using uniformly-generated rules: for every rule, each endpoint of each of the 4 fields (IP address ranges and port ranges) was selected uniformly at random from its domain. We built the GEM data structure for increasing numbers of such rules and then used the resulting structure to match randomly generated packets. We omit the details for lack of space, and instead refer the reader to. On one hand, these early simulations showed us that the search itself was indeed very fast: a single packet match took around 1µsec, since it only required 4 executions of a binary search in memory. On the other hand, we learned that the data structure size grew rapidly—and that the order of fields had little or no effect on this size. The problem was that since the ranges in the rules were chosen uniformly, almost every pair of ranges (in ever y dimension) had a non-empty intersection. All these intersections produced a very fragmented space subdivision, and effectively exhibited the worst -case behavior in the data structure size. We concluded that a more realistic rule model is needed

4.2 The Perimeter Rules Model

Real firewall rule-bases have a large degree of structure. Thus, we hypothesized that realistic rule-bases rarely cause worst-case behavior for the GEM algorithm. Furthermore, we wanted to test the effects of the field order on the performance of GEM on such rule-bases. For this purpose, we built the Perimeter firewall rules model, and simulated the behavior of GEM on rule-bases generated in this model.

4.2.1 The Modeled Topology

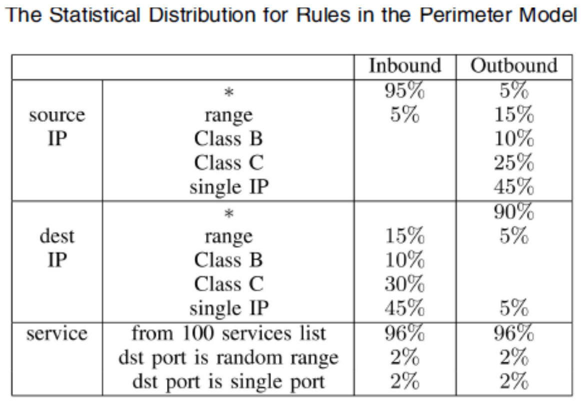
The model assumes a perimeter firewall with two "sides": a protected network on the inside, and the Internet on the outside. The inside network consists of 10 class B networks, and the Internet consists of all other IP addresses. Thus, the internal network contains 10 - 65,536 possible IP addresses. In reality, organizations that actually own 10 class B networks are quite rare. However, we used this assumption for two reasons:

1. Many organizations use private IP addresses internally, and export them via network address translation (NAT) on outbound traffic. Such organizations often use large subnets liberally, e.g., assign a 172.x.*.* class B subnet to each department.
2. Having a large internal subnet stresses the GEM algorithm since we pick random ranges from the internal ranges.

4.2.2 The Rules

The Perimeter rules model produces rules of two types: Inbound rules that allow traffic from the Internet into the protected network, and outbound rules that allow traffic from the protected network out to the Internet. Each rule in the rule-base is constructed randomly according to the distribution detailed in Table 3 for its type (Inbound or Outbound). Inbound rules. When we are modeling rules for inbound traffic, the source IP addresses are rarely specified in the rules, and 95 percent of the rules have "*" as their source address. The remaining five percent have a range in their source address field, chosen uniformly at random from the Internet's IP addresses. The destination addresses for inbound rules are always internal, belonging to the 10 internal class B subnets. Forty-five percent of the rules have a randomly chosen individual internal IP address as a destination, modeling server machines. Another 15 percent have a small random range: a range which completely lies inside one of the internal class C networks. These ranges model clusters of servers and small classless subnets such as '/27's and '/28's. Then, 30 percent of the rules have a complete class C as a destination. Finally, 10 percent allow access to a full class B. Note that inbound rules produce many "collisions" in the destination field. For example, consider the 30 percent of rules with a full class C destination. Essentially, the same is true for collisions of a single-internal-IP-destination and an internal class C subnet, since every internal IP address has exactly a 1:2,560 chance of falling inside a particular internal class C. Outbound rules. When we are modeling the outbound rules, 90 percent of the rules have a destination IP address of "*" Ten percent of the rules have either a specific address or a range in the destination field, modeling a rule that restricts or allows access to some particular server or network. The source addresses for outbound rules are selected from the internal addresses with the frequencies shown in Table 3.

Table 3



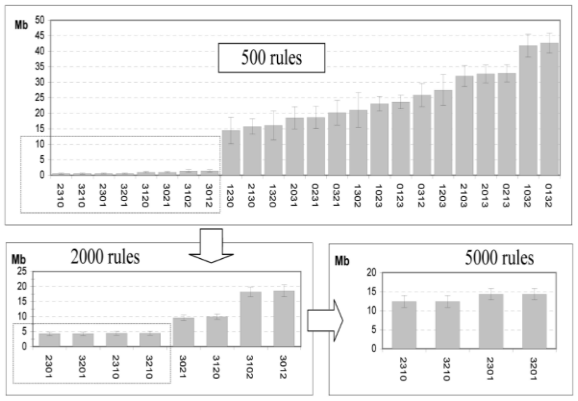
The Statistical Distribution for Rules in the Perimeter Model

		Inbound	Outbound
source IP	*	95%	5%
	range	5%	15%
	Class B		10%
	Class C		25%
	single IP		45%
dest IP	*		90%
	range	15%	5%
	Class B	10%	
	Class C	30%	
	single IP	45%	5%
service	from 100 services list	96%	96%
	dst port is random range	2%	2%
	dst port is single port	2%	2%

Services. The service field in the rules is selected similarly for both Inbound and Outbound rules. The service is selected uniformly at random from a collection of 100 services, whose definitions were taken from real firewall rule-bases (recall Table 2). Most of these services have individual destination port numbers; however, a few include port ranges, and one service is the "*" service. We allow a small rate of growth in the number of services by adding two percent of randomly generated services, where the destination port is randomly picked from 0 to 65,535.

One concern we had was that, occasionally, the model generated a rule of the form "from * to *, with service '*'. When such a rule shows up in the rule-base, it acts as the default rule, and all subsequent rules become redundant, because of the "first match" semantics. This effectively shortens the rule-base, and prevents us from simulating GEM's behavior on large rule-bases. Thus, our model checks for, and discards, such randomly generated catchall rules. The rule-bases generated by the model are still much less structured than actual firewall rule-bases. In real firewall rule-bases, the number of internal servers is usually rather small, and they have many rules that refer to them. Also, it is considered insecure to allow many TCP services into large parts of the internal networks. Both considerations would cause more repetitions in IP addresses, and hence, reduce the number of simple ranges, which would lead to smaller search data structures. Therefore, we believe our Perimeter model stresses the GEM algorithm more than real firewall rule-bases would.

Figure 7: Finding the Best field order



4.3 Selecting the Best Field Order

Our first goal in the Perimeter model is to determine if any efficiency can be achieved by selecting the GEM data structure field order. Preliminary simulations showed us that the order of fields had a very strong impact on the size of the data structure in the Perimeter model (several orders of magnitude between best and worst choices). The variance was so large that we were unable to simulate the worst choices on large rule-bases, since the data structure grew to hundreds of megabytes and took up to 20 minutes to build.

The rationale is that the usage patterns in the different fields are non uniform (as we saw in Figure 7), so some choices of fields in the high levels of the hierarchy cause large amount of subdivisions in the lower levels (many ranges are created). Therefore, we used a three-stage process to identify the best order. In the first stage, we generated small (500 rules) rule-bases, and built the data structure for each of the 4! = 24 possible orders. This simulation showed that 16 orders were clearly much worse than others, so we dropped them and continued to 2,000-rule sets with the remaining eight orders. Here, we found that the best four orders were better than the rest. The top four candidates were evaluated on 5,000-rule sets, which identified the best and second-best orders. The process of finding the best order for the "Perimeter" model is shown in Fig. 7. Moreover, a closer look shows that the position of field "2" (source port) among the best eight orders is less significant: there are really only two orders (310 and 301) with the "2" field inserted in all four possible positions. This is reasonable because the source port in the Perimeter model is almost always "*" so its position in the order has a limited impact. Therefore, for all subsequent tests, we somewhat arbitrarily used the "natural" order of 3210 (destination port, source port, destination IP address, source IP address).

5 Conclusions

We conclude that the GEM algorithm is an efficient and practical algorithm for firewall packet matching. We implemented it successfully, and tested its packet-matching speeds on live traffic with realistic large rule-bases. GEM's matching speed is far better than the naive linear search, and it is able to increase the throughput of IP tables by an order of magnitude. On rule-bases generated according to realistic statistics, GEM's pace complexity is well within the capabilities of modern hardware. As for GEM itself, we would like to explore the algorithm's behavior when using more than four fields, e.g., matching on the TCP flags, metadata, interfaces, etc.

6 References

[1]Dmitry Rovniagin and Avishai Wool, The Geometric Efficient Matching Algorithm for Firewalls, IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, January- 2011

[2] F. Baboescu, S. Singh, and G. Varghese, ―Packet classification for core routers: Is there an alternative to cams,‖ in Proc. IEEE
INFOCOM, 2003.

[3] F. Baboescu and G. Varghese ―Scalable packet classification, in Proc. ACM SIGCOMM, 2001, pp. 199–210.

[4] N. Bar-Yosef and A. Wool, ―Remote algorithmic complexity attacks against randomized hash tables,in Proc. International Conference on Security and Cryptography (SECRYPT), Barcelona,
Spain, Jul. 2007, pp. 117–124.

[5] W. R. Cheswick, S. M. Bellovin, and A. Rubin, Firewalls and
Internet Security: Repelling the Wily Hacker, 2nd ed. Addison- Wesley, 2003.

[6] M. Christiansen and E. Fleury, ―Using interval decision diagrams for packet filtering,‖ 2002, http://www.cs.auc.dk/_fleury/publications.html.

[7] E. Cohen and C. Lund, ―Packet classification in large ISPs: Design and evaluation of decision tree classifiers,‖ in Proc. ACM SIGMETRICS. New York, NY, USA: ACM Press, 2005, pp. 73–84.

[8] S. Crosby and D. Wallach, ―Denial of service via algorithmic complexity attacks,‖ in Proceedings of the 12th USENIX Security
Symposium, August 2003, pp. 29–44.

[9] M. de Berg, M. van Kreveld, and M. Overmars, Computational Geometry: Algorithms and Applications, 2nd ed. Springer-Verlag, 2000.

[10] D. P. Dobkin and R. J. Lipton, ―Multidimensioal searching problems,‖ SIAM J. Comput., vol. 5, no. 2, pp. 181–186, 1976.

[11] D. Eppstein and S. Muthukrishnan, ―Internet packet filter management and rectangle geometry,‖ in ACM-SIAM Symp. On Discrete Algorithms (SODA), 2001, pp. 827–835.

[12] A. Feldmann and S. Muthukrishnan, ―Tradeoffs for packet classification,‖ in Proc. IEEE INFOCOM, 2000, pp. 1193–1202.

[13] W. Feller, An Introduction to Probability Theory and Its Applications, 3rd ed. New York: John Wiley & Sons, 1967, vol. 1.

[14] ―Firewall Wizards,‖ Electronic mailing list, 1997–2009,archived at
https://listserv.icsalabs.com/pipermail/firewall-wizards/.

[15] P. Gupta and N. McKeown, ―Algorithms for packet classification,‖ *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.

[16] ——, ―Packet classification on multiple fields,‖ in *Proc. ACM
SIGCOMM*, 1999, pp. 147–160.

[17] D. Hartmeier, ―Design and performance of the OpenBSD stateful packet filter (pf),‖ in *Proc. FREENIX Track: 2002 USENIX Annual Technical*

[18] R. Jain, *The Art of Computer Systems Performance Analysis* John Wiley & Sons, 1991.

[19] S. Kandula, D. Katabi, M. Jacob, and A. Berger, ―Botz-4-sale:
Surviving organized DDOS attacks that mimic flash crowds,‖ in *NSDI*, 2005.

[20] T. V. Lakshman and D. Stiliadis, ―High-speed policy-based packet forwarding using efficient multi-dimensional range matching,‖ in *Proc. ACM SIGCOMM*, 1998, pp. 203–214.

[21] M. Kasi Viswanadh , An Efficient & Robust
Packet Matching Algorithm
for Firewall - International Conference on Computing and Control Engineering (ICCCE 2012), 12 & 13
April, 2012

[22] M. Kasi Viswanadh , An Efficient & Robust
Packet Matching Algorithm
for Firewall - International Conference on Computing and Control Engineering (ICCCE 2012), 12 & 13
April, 2012

[23] CISCO Firewall Material

[24] *Dmitry Rovniagin, Avishai Wool*
, THE GEOMETRIC EFFICIENT MATCHING
ALGORITHM FOR FIREWALLS EXTENDED
ABSTRACT
School of Electrical Engineering,
Tel Aviv University, Ramat Aviv 69978, Israel.