Volume 4, Issue 10, October 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Survey on Geometric Efficient Matching Algorithm for Firewalls

Mr. C. M. Jadhav, Ms. Priyanka Harish Pachkore

Bharat Ratna Indira Gandhi COE,

Solapur, India

Abstract– A firewall is a security guard placed at the point of entry between a private network and the outside Internet such that all incoming and outgoing packets have to pass through it. The function of a firewall is to examine every incoming or outgoing packet and decide whether to accept or discard it. Each firewall packet has 5 fields source IP, destination IP, source port, destination port and protocol. Each packet needs to be checked with every firewall rule to find the first matching rule. The geometric efficient packet matching algorithm gives a logarithmic matching time performance. Based on statistics from real firewall rule-bases a perimeter rule model is created that generates random, but non-uniform, rule-bases. Firewalls need to filter all the traffic crossing through the network, it should be able to sustain high throughput, or risk becoming a bottleneck. In this system, Geometric Efficient Matching Algorithm is adapted to the firewall domain, which needs logarithmic time. But, the algorithm's worst-case space complexity is O (n4) for a rule-base with n rules. Because of this, a Perimeter rules model that generates random, but non uniform, rule-bases is designed which shows GEM is an excellent choice. Based on statistics from real firewall rule-bases, GEM is evaluated via extensive simulation using the Perimeter rules model. GEM uses near-linear space, and only needs approximately 13 MB of space for rule-bases of 5,000 rules. It reduces the space requirement to 2-3 MB for 5,000 rules using space optimizations. Therefore, GEM is an efficient and practical algorithm for firewall packet matching.

Keywords—Firewall, Network security, Stateful filtering,

I. INTRODUCTION

The firewall is central technologies which allowing high-level access control to organization networks. Packet matching in firewalls involves matching on many fields from the TCP and IP packet header. Five fields (protocol number, source and destination IP addresses, and ports) are involved in the decision which rule applies to a given packet. In modern firewall, we need to deploy very efficient algorithm to avoid the bottleneck problem related to firewall. Since firewalls need to filter all the traffic crossing the network, they should be able to sustain a very high throughput, or risk becoming a bottleneck. Thus, algorithms from computational geometry can be applied. In this paper we consider a classical algorithm that we adapted to the firewall domain. We call the resulting algorithm ―Geometric Efficient Matching‖ (GEM). The GEM 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. Because of this perceived high space complexity, GEM-like algorithms were rejected as impractical by earlier works. Contrary to this conclusion, this paper shows that GEM is actually an excellent choice. Based on statistics from real firewall rule-bases, we created a Perimeter rules model that generates random, but non-uniform, rule bases. We evaluated GEM via extensive simulation using the Perimeter rules mode l.

II. FIREWALL

A firewall can either be software-based or hardware-based and is used to help keep a network secure. Its primary objective is to control the incoming and outgoing network traffic by analyzing the data packets and determining whether it should be allowed through or not, based on a predetermined rule set. A network's firewall builds a bridge between an internal network that is assumed to be secure and trusted, and another network, usually an external (inter)network, such as the Internet, that is not assumed to be secure and trusted.

Stateless firewall *:* A *stateless* firewall filter can filter packets transiting the device from a source to a destination, or packets originating from, or destined for, the Routing Engine. Stateless firewall filters applied to the Routing Engine interface protect the processes and resources owned by the Routing Engine. You can apply a stateless firewall filter to an input or output interface, or to both. Every packet, including fragmented packets, is evaluated against stateless firewall filters [6]. Stateless Firewalls are the most basic and they are the most common type of firewalls. Stateless Firewalls (SlF) basically watch the traffic and compares the packets with the rules from its rules database. If a malicious activity is found it drops the packet. They are not aware of the traffic flowing among them. For simple lightweight host – based protections usually stateless firewalls are preferred. There are many examples for stateless firewalls: ip tables from Linux, default Windows XP SP2 firewall …etc. But we can't say that a computer protected within stateless firewalls are totally away from network attacks because of its non-adaptive nature, stateless firewall vulnerable to the ACK scanning reconnaissance attack (and more). Unfortunately KISS principle doesn't always induce good results.

Stateful firewall : In computing, a Stateful firewall (any firewall that performs Stateful packet inspection (SPI) or Stateful inspection) is a firewall that keeps track of the state of network connections (such as TCP streams, UDP communication) traveling across it. The firewall is programmed to distinguish legitimate packets for different types of connections. Only packets matching a known active connection will be allowed by the firewall; others will be rejected.[6] A Stateful firewall keeps track of the state of network connections (such as TCP streams or UDP communication) and is able to hold significant attributes of each connection in memory. These attributes are collectively known as the state of the connection, and may include such details as the IP addresses and ports involved in the connection and the sequence numbers of the packets traversing the connection. Stateful inspection monitors incoming and outgoing packets over time, as well as the state of the connection, and stores the data in dynamic state tables. This cumulative data is evaluated, so that filtering decisions would not only be based on administrator-defined rules, but also on context that has been built by previous connections as well as previous packets belonging to the same connection. The most CPU intensive checking is performed at the time of setup of the connection. Entries are created only for TCP connections or UDP streams that satisfy a defined security policy. After that, all packets after that (for that session) are processed rapidly because it is simple and fast to determine whether it belongs to an existing, pre-screened session. Packets associated with these sessions are permitted to pass through the firewall. Sessions that do not match any policy are denied, as packets that do not match an existing table entry. In order to prevent the state table from filling up, sessions will time out if no traffic has passed for a certain period. These stale connections are removed from the state table. Many applications therefore send keep alive messages periodically in order to stop a firewall from dropping the connection during periods of no user-activity, though some firewalls can be instructed to send these messages for applications. Many stateful firewalls are able to track the state of flows in connectionless protocols. UDP hole punching is the technique associated with UDP. Such sessions usually get the ESTABLISHED state immediately after the first packet is seen by the firewall. Sessions in connectionless protocols can only end by time-out. By keeping track of the connection state; stateful firewalls provide added efficiency in terms of packet inspection. This is because for existing connections the firewall need only check the state table, instead of checking the packet against the firewall's rule set, which can be extensive. Also, the concept of deep packet inspection is unrelated to stateful firewalls, because of its stateful feature, which checks incoming traffic against its state table first instead of jumping to the firewall's rule set. In this case if the state table is matched, then it doesn't need deep packet inspection.

III. 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 ith field in a packet header. The field values are in $0 \leq li; r i \geq Ui$, where Ui = 232 _ 1 for IP addresses, Ui =65535 for port numbers, and Ui = 255 for ICMP message type or code. Table 1 lists the header fields we use (the port fields can double as the message type and code for ICMP packets). For notation convenience later on, we assign each of these fields a number, which is also listed in the table.

Table 1 Remarks

Number	Description	Space
0	Source IP Address	32 bit
1	Destination IP Address	32 bit
2	Source Port Number	16 bit
3	Destination Port Number	16 bit
4	Protocol	8 bit

Most firewalls allow matching on additional header fields, such as IP options, TCP flags. Nearly all the firewall rules that we have seen only refer to the five fields listed in Table 1.

- In view of the description above, the GEM algorithm is mostly suitable to firewalls whose rules use contiguous ranges of IP addresses.
- 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.

IV. THE SUBDIVISION OF SPACE

In one dimension, each rule defines one range and each range divides space into at most three parts. It is easy to see that n possibly overlapping rules define a subdivision of 1D space into at most [n _ 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 4D space algorithm, we have four levels of subdivisions. The last level is exactly a 1D case—among all the active rules, only the

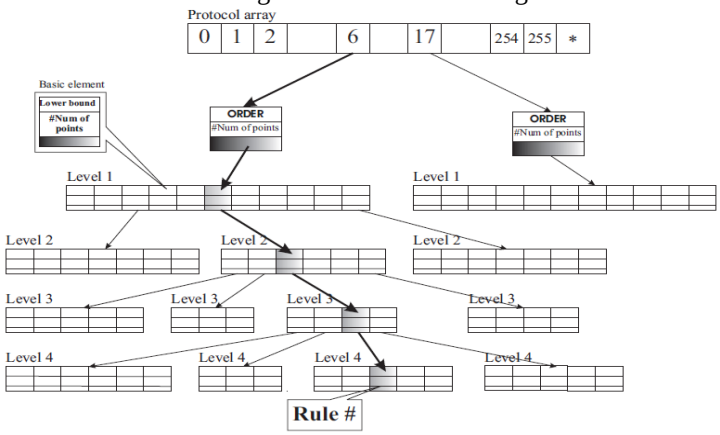
winner rule matters. At this point, we have a subdivision of d-dimensional space made up of simple hyper rectangles, each corresponding to a single winning rule. In Section 2.4, we shall see how to efficiently create this subdivision of d-dimensional space, and how to translate it into an efficient search structure.

DEALING WITH PROTOCOL FIELD: In 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 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 nontrivial range.

Thus, the GEM algorithm only deals with single values in the protocol field, with special treatment for rules with ―*‖ as a protocol. We preprocess the firewall rules into categories, by protocol, and build a separate search data structure for each protocol (including a data structure for the ―*‖ protocol). The actual geometric search algorithm only deals with four fields. Now, a packet can only belong to one protocol—but it is also affected by protocol = ―*‖ rules. Thus, every packet needs to be searched twice: once in its own protocol's data structure, and once in the ―*‖ structure. Each search yields a candidate winner rule. We take the action determined by the candidate with the lower number. In this paper, we focus on the TCP protocol, which has d = 4 dimensions, although the same discussion applies for UDP and ICMP

V. DATA STRUCTURE

The geometric efficient matching algorithm data structure consists of three parts. The size of this data structure is 12 bytes. Each part is divided in 4 bytes. First 4 bytes contains the array pointer of protocol which include each protocol and * protocol. Second part contain protocol database header. It includes order of data structure. The order number include source IP Address, Destination IP address, Source port number, Destination Port number. The order contains the number of field header which explains in table 1. According to this order numbering data structures level are defined



Overall GEM data structure overview

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 themselves. 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 Fig. 2 for an illustration. 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: four for the value of the left boundary of the range, four for the pointer to the next level, and four for the number of cells in the next-level node. The nodes at the deepest level are slightly different, consisting of only 8 bytes: four for the left boundary of the range and four for the number of winner rules. Note that the order of levels is encoded in the protocol database header, which gives us convenient control over the field evaluation order.

SEARCH ALGORITHM*:*

Input: Protocol number.
1: Apply Binary search on Protocol array.
2: Get Simple range of protocol header.
3: Select Protocol database header;
4: According to combination of field apply binary search on each level.
5: Get Simple range;
6: Finding the winner rule.

The packet header contains the protocol number, source and destination address, and port number fields. First, we check the protocol field and go to the protocol array of the search data structure, to select the corresponding protocol database header. From this point, we apply a binary search with the corresponding field value on every level, in order to find the matching simple range and continue to the next level. The last level will supply us with the desired result—the matching

rule number. For example, suppose we have an incoming TCP packet. Assume that the GEM protocol header for TCP shows that the order of levels is 1,203. The first level 1 denotes the Destination address. We execute a binary search of the destination address value from the 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, 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.

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! ¼ 24 possible orders we can choose from, to check four 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.

Sweep Line Technique:

1. Initialized data structures. Insert site events into the priority queue based on their y-coordinate value.
2. While (Queue not empty) event = Pop first event from queue
3. If (event = = Site Event)

Process Site (event) Else Process Circle (event)

4. Finish all edges in binary search tree

VI. SPACE OPTIMIZATION:

There Are Two Space optimization used in this technique:

1. first optimization

a. Work on last level data structure.

b. In it two or more adjacent ranges point to same winner rule.

c. Replace all ranges with single ranges using union.

2. Second optimization

a. Work on one before last level data structure.

b. Exist simple ranges that point to equivalent last level structure.

c. Instead of storing last level multiple times, keep single last level & replace duplicate by pointers to main copy

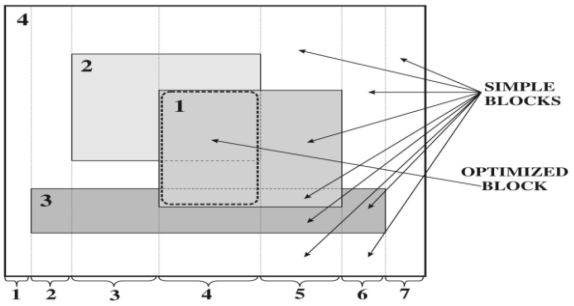


Fig 3. Space Optimization with four rule

For example, in Fig. 3, ranges 2 and 6 are equivalent (rules 4-3-4, with boundaries in the same vertical positions). A space complexity of O (n4) may be theoretically acceptable since it is polynomial.

 RANDOM RULES: We started by testing GEM using uniformly generated rules: for every rule, each endpoint of each of the four 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. On one hand, these early simulations showed us that the search itself was indeed very fast; a single packet match took around 1 μsec.since it only required four executions of a binary search in memory.

 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. *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.

 FIRST MATCH: They introduced an efficient packet classification algorithm, which is optimized for a hardware implementation. Their algorithm divides the address space into ranges created by borders of the rules, and encodes these

ranges into a much smaller ―number space.‖ They then project the rules onto this smaller space, and repeat until the number space is small enough, at which point they assign the winning rule to each encoded range. The authors did not present an asymptotic time complexity analysis. GEM enjoys a logarithmic matching time, but suffers from an O (nd) worst-case space complexity, when the matching is performed on d fields. However, this algorithm works in one dimension, and may be scaled to two dimensions, but it seems.

LONGEST PREFIX MATCH: There is an extensive literature dealing with router packet matching, usually called ―packet classification.‖ Existing algorithms implement the ―longest prefix match‖ semantics, using several different approaches. The IPL algorithm divides the search space into elementary intervals by different prefixes for each dimension, and finds the best (longest) match for each such interval. The Tuple Space Search algorithm all the prefixes are divided into tuples by field prefix length, and then searched linearly.

VII. CONCLUSION

GEM algorithm is an efficient and practical algorithm for firewall packet matching, GEM's matching speed is far better than the naive linear search. GEM's space complexity is well within the capabilities of modern hardware. The GEM algorithm enjoys a logarithmic matching time performance.GEM algorithm is an efficient and practical algorithm for firewall packet matching. GEM's space complexity is good as compare to modern hardware.

REFERENCES

[1] Dmitry Rovniagin and Avishai Wool, Senior Member, IEEE ―THE GEOMETRIC EFFICIENT MATCHING ALGORITHM FOR FIREWALLS‖ IEEE *Transactions On Dependable And Secure Computing ,Vol. 8, No. 1, Jan-Feb 2011*.

[2] Alex X. Liu, Member, IEEE, and Mohamed G. Gouda, Member, IEEE ―Firewall Policy Queries‖ IEEE *Transactions On Parallel And Distributed Systems*, Vol. 20, No. X, Xxx 2009.

[3] Alex X. Liu Eric Torng Chad R. Meiners, Department of Computer Science and Engg, Michigan State University, East Lansing, MI 48824, U.S.A. ―*Firewall Compressor: An Algorithm for Minimizing Firewall Policies*‖.

[4] Andronescu Alexandra ―LIBFW: *GENERIC FIREWALL LIBRARY FOR MULTIPLE OPERATING SYSTEMS*‖.

[5] Case study & implementation of‖ *THE GEOMETRIC EFFICIENT MATCHING ALGORITHM FOR FIREWALLS*‖by S.Pradeep, V.Mahesh,Y.Ramesh Kumar,Department of Computer Science & Engineering, Avanthi Institute Of Engineering & Technology,Cherukupalli,Andra Pradesh,India.

[6] Kuhn, Josh. Grouper: *A Packet Classification Algorithm Allowing Time-Space* Tradeoffs. Diss. University of South Florida, 2011.

[7] ―*Optimizing Firewall Performance*‖, Anssi Kolehmainen, Helsinki University of Technology, anssi.kolehmainen@hut.fi

[8] Mehlhorn, Kurt, and Stefan Näher. "Implementation *of a sweep line algorithm for the straight line segment intersection problem."* TR MPI-I-94-105, Max-Planck-Institut fur Informatik, Saarbrucken. 1994.

