Bit Vector Algorithms Enabling High-Speed and Memory-Efficient Firewall Blacklisting

Lane Thames
Georgia Institute of Technology
210 Technology Circle
Savannah, GA 31407
001-912-966-7210
lane.thames@gatech.edu

Randal Abler
Georgia Institute of Technology
210 Technology Circle
Savannah, GA 31407
001-912-966-7210
randal.abler@gatech.edu

David Keeling
Georgia Institute of Technology
210 Technology Circle
Savannah, GA 31407
001-912-966-7210
dkeeling@gatech.edu

ABSTRACT
In a world of increasing Internet connectivity coupled with increasing computer security risks, security conscious network applications implementing ***blacklisting*** technology are becoming very prevalent because it provides the ability to prevent information exchange from known malicious sources. Current technology implementing blacklisting does so at the application level. However, there are numerous benefits for implementing blacklisting filters in the firewall. These benefits include reduced application workload and reduced bandwidth consumption. But, because the *de facto* algorithm in firewalls is based on a linear search first match principle, large blacklists are not feasible to implement in firewalls due to the O(N) timing complexity of linear search methods. This paper addresses this issue by describing techniques that solve the O(N) time complexity issue without changing the internal input-output behavior of the firewall.

Categories and Subject Descriptors
C.2.0 [**Computer-Communication Networks**]: General – *Security and Protection*

General Terms
Algorithms, Security

Keywords
Firewalls, Packet Classification, Bit Vectors, Blacklisting, Path Matrix

1. INTRODUCTION
Blacklisting is a technique where a centralized store of Internet Protocol (IP) addresses (or address ranges) of known malicious sources are kept in a database. Applications query the blacklist database when a source is attempting to exchange information with the application. If the source trying to send or receive data from the application is a member of the blacklist, then the information exchange is not allowed. Email systems have used blacklisting techniques for years because of its ability to prevent significant amounts of unsolicited email (SPAM) from well known SPAM generators (SPAM sources). However, many new applications are adopting this technique. These new application areas include DNS blacklisting, URL blacklisting, URI blacklisting, and Phishing-based blacklisting. Currently, blacklisting techniques are performed at the application level. Independent applications query the specific blacklist database as needed. However, it would be beneficial to perform the blacklisting operation at the firewall. Instead of querying the database for each information exchange, keep a copy of the blacklist in the firewall. If the incoming packet is going to or coming from a blacklist entity, the firewall should deny the packet. Applications waste resources for information exchanges belonging to blacklist sources. If multiple applications that use blacklists are active on a machine, then there is a potential to waste a significant amount of resources such as CPU time and network bandwidth due to multiple blacklist queries. However, the firewall can be a single application such as a host-based software firewall or dedicated device such as a network-based firewall that handles the blacklisting process for many applications. This, in turn, helps applications to use their resources for their primary functionality. Unfortunately, the *de facto* search method for firewalls is based on linear search first match principles. Linear search methods have O(N) time complexity, where N is the number of elements in the search list. Hence, firewalls will suffer from decreased network traffic throughput if the number of firewall rules in the rule list, i.e. N, increase. Depending on the type of database, blacklists can be very large. Hence, translating blacklists into firewall rules is prohibitive in terms of search time complexity, and this is the fundamental reason why blacklisting is currently implemented at the application level. This paper addresses this fundamental issue and shows that a firewall can be designed such that the O(N) time complexity can be solved such that firewall based blacklisting can be achieved.

ACMSE '09 March 19-21, 2009, Clemson, SC, USA.

2. RELATED WORK
Zhang *et al*. [8] recently published their work in the area of blacklisting regarding a technique known as highly predictive blacklisting (HPB). The goal of HPB is to extract a blacklist subset from a global blacklist such that the extracted blacklist subset is more relevant to a particular network. Their algorithm is based on a relevance ranking system equivalent to Google's

PageRank system. Soldo *et al*. [7] provide algorithmic techniques, with a goal similar to [8], to extract the most relevant blacklist members from a global blacklist. In [7], the authors realize that blacklist firewall filtering is limited because of the O(N) complexity. Their paper introduces a family of algorithms that provide the most relevant blacklist members when there is a constraint on the total number of rules that can be implemented in a firewall. The algorithms in [7] and [8] only provide techniques to extract the most relevant blacklist members whereas the algorithms discussed in this paper are aimed at enhancing the firewall's classification performance in terms of classification time, memory consumption, and data structure update time. Another work focused on blacklisting is provided by Kim *et al*. [4]. Their work is a study regarding the effectiveness of blacklisting "Martian" (also know as bogon) IP addresses in routers and firewalls to counter denial of service attacks. There have been many works in the area of general packet classification—Gupta [3] provides a good overview of the seminal works. However, general packet classification is still an open research problem because there is currently no general solution. Most current solutions have good search time performance but suffer from large memory consumption or poor data structure update time due to extensive pre-computation. Other bit vector packet classification algorithms have been proposed [5], [6] for general packet classification. But, these solutions suffer from poor update time due to pre-computation and from the "prefix-expansion problem" [3].

3. FIREWALLS AND PACKET CLASSIFICATION

Packet classification is the act of some computing node receiving a data packet from the network, examining certain pre-defined fields from the packet, comparing these fields against a database (or rule set) of classification statements, and performing some type of action defined by the classification statement that matches the incoming packet's fields. Many types of network devices such as firewalls, routers, and switches have the ability to perform packet filtering via packet classification algorithms. The algorithms described in this paper are targeted towards general packet filter based packet classification for blacklisting; however, the discussion is dedicated primarily to firewalls.

At the heart of packet classification is the set of rules that define a particular classifier. For network and security based classification, the rules are composed of field values from the underlying network protocol stack, i.e. TCP/IP. The classic 5-tuple used in most firewalls consists of the transport layer source and destination port numbers, the internetworking layer source and destination IP addresses, and transport layer protocol type. Associated with each rule is an action to be applied for packets that match the rule. The common set of actions for a firewall include ***allow*** or ***deny***. An allow action means that the firewall should pass the packet onto the network towards its destination, and deny means that the packet should be dropped and not allowed to pass through the firewall. The semantics of most common firewall rules follow an if-then-else construct. The set of rules are created by the network administrator, and each rule is inserted into the rule set based on a specific order. Normally, the order is dictated to be that the first rule has higher priority than the second rule, etc. This leads to the following type of logic: "if(packet header fields match rule set 1) then perform action1, else if(packet header fields match rule set 2) then perform action2, **…** else perform default action". The next section provides formal definitions for packet classification.

3.1 Formal Definitions of Packet Classification

A packet classifier $\boldsymbol{F}$ is defined over the set of rules $\{\mathbf{R_i}\}$, $0 \le i \le N$. Each $\mathbf{R_i}$ is defined over a set of fields $\{\mathbf{f_{ik}}\}$ $0 \le k \le d$ and each field corresponds to some value in the specification of the underlying network protocol stack, i.e. the TCP/IP 5-tuple described above. In the general case of packet classification, the classifier fields can be any of d values from the protocol stack. In the case of the classic 5-tuple, d = 5. Each field value is defined over the set of integers such that $\mathbf{f_{ik}} \in [0, 2^W\text{-}1]$ where W is the number of bits specified for the particular protocol value, i.e. W = 32 for IP version 4 (IPv4) addresses. For each $\mathbf{f_{ik}}$, the field can be described as an integer (point-wise) value, a prefix value, or a range value. Each $\mathbf{R_i}$ is also defined by its associated action $\mathbf{A_i}$. The common set of action classes is $\mathbf{A}$ = {allow, deny}. In general, an incoming packet can match multiple rules. Therefore, a priority is given to the set of rules. In most packet classifiers, the priority is defined as $\mathbf{P(R_i)} = i$ and priority is measured such that $i < j$ implies that $\mathbf{P(R_i)} > \mathbf{P(R_j)}$. Given an incoming packet, the classifier extracts the corresponding set of fields $\mathbf{p} = \{p_0, \ldots, p_d\}$. Then, the output of the classifier is given by: $\boldsymbol{F}(\mathbf{p}, \{\mathbf{R_i}\}) = \mathbf{A_m}$, where $\mathbf{A_m}$ is the action defined by the highest priority rule $\mathbf{R_m}$ that matches $\mathbf{p}$.

Implementing firewall rule sets can be a non-trivial matter for large and complex networks. Several works such as [1], [2] have addressed the issue of firewall rule set correctness. Two common problems in firewall rule sets are known as rule *redundancy* and rule s*hadowing*. Both of these are the result of rules that are defined such that packets can match multiple rules. Redundancy is a result of a rule $\mathbf{R_n}$ that never gets processed because some rule $\mathbf{R_m}$, $m < n$, matches all packets that also match $\mathbf{R_n}$. In this case, $\mathbf{R_m}$ "covers" $\mathbf{R_n}$. Shadowing is very similar to redundancy except that only a subset of packets that match $\mathbf{R_m}$ will match $\mathbf{R_n}$. Shadowing is the primary reason that priority is required for packet classifiers. Rearranging a shadowed set such that $\mathbf{R_n}$ is placed before $\mathbf{R_m}$ will change the underlying logic of the rule set if $\mathbf{A_m} \neq \mathbf{A_n}$. An example will help to clarify this issue. Consider the following pseudo rule definition: $\mathbf{R_i}$ = {protocol, source IP address, destination IP address, destination port, action type}. Now consider the following 2 rules:

$\mathbf{R_1}$ = {tcp, 10.10.10.10, 10.20.30.40, 25, deny}

$\mathbf{R_2}$ = {tcp, 10.10.10.0/24, 10.20.30.40, 25, allow}

This example illustrates the issue of shadowing and the need for priority (or correct rule arrangement). The first rule denies access to the SMTP server 10.20.30.40 on TCP port 25 from source 10.10.10.10. The second rule allows access to all other hosts in the 10.10.10.0/24 network to the SMTP server. The first rule only matches a subset of the second rule. Further, the actions are different. Therefore, changing the rule order for these two rules changes the logic (or input-output behavior) of the classifier. If the second rule is placed before the first rule, then we will have an instance of redundancy such that the new second rule will never be matched because the new first rule completely covers the second.

Shadowing and redundancy are the result of rule *correlation*. Rules can be completely correlated, partially correlated, or uncorrelated. Two rules $\mathbf{R_i}$ and $\mathbf{R_j}$ are partially correlated *iff* there is at least one field and no more than d-1 fields such that Equation 1 holds true.

$$\mathbf{f_{im}} \cap \mathbf{f_{jm}} \neq \boldsymbol{\varphi}\,,\ 0 \leq m \leq d,\ 0 \leq i,j \leq N,\ i \neq j \quad (1)$$

In Equation 1, $\boldsymbol{\varphi}$ represents the empty set. $\mathbf{R_i}$ and $\mathbf{R_j}$ are completely correlated if Equation 1 holds true for *all* d fields. Finally, $\mathbf{R_i}$ and $\mathbf{R_j}$ are uncorrelated if they are neither partially correlated nor completely correlated. We now define the rule correlation measure, C(i,j). C(i,j) = 0 *iff* $\mathbf{R_i}$ and $\mathbf{R_j}$ are uncorrelated or partially correlated. C(i,j) = 1 *iff* $\mathbf{R_i}$ and $\mathbf{R_j}$ are completely correlated. With the ideas of rule correlation, we can now define rule independence.

Definition 3.1.1: Two rules $\mathbf{R_i}$ and $\mathbf{R_j}$ are independent *iff*:

$$C(i,j) = 0 \ \vee \ \mathbf{A_i} = \mathbf{A_j} \quad (2)$$

Two rules are independent if they are not completely correlated or if their actions are the same. If two rules are independent and j = i + 1, then their order (priority) does not matter. If j = i + 1, these rules can be reordered such that i' = j and j' = i without changing the underlying logic of the rule set. Further, if j > i + 1 and all rules between i < r ≤ j are independent, then this cluster of rules can be reordered without changing the underlying logic of the firewall.

Definition 3.1.2: Given a rule set $\{\mathbf{R_i}\}$ with a cluster of rules $\{\mathbf{R_i}\}_\mathbf{r}$ i ≤ r ≤ j that are independent, a new rule set $\{\mathbf{R_i}\}'$ can be created by rearranging the cluster set $\{\mathbf{R_i}\}_\mathbf{r}$ such that $\boldsymbol{F}(\mathbf{p}, \{\mathbf{R_i}\}) = \boldsymbol{F}(\mathbf{p}, \{\mathbf{R_i}\}')\ \forall\ \mathbf{p}$.

There are many more definitions describing packet classification, but a complete discussion of packet classification is beyond the scope of this paper. However, with the definitions provided in this section, we can now describe a solution for firewall blacklisting.

3.2 Enabling Firewall Blacklisting

If a network administrator wants to blacklist a particular IP address in the firewall, then a rule denying access to a particular service from the blacklisted IP address must be created. For example, $\mathbf{R_1}$ from above could be a rule denying access to the SMTP server from 10.10.10.10 where 10.10.10.10 is a blacklist member. However, this paper proposes the idea of denying access at the firewall for **all** packets to or from a blacklist member under the assumption that the blacklist database is accurate. An inaccurate blacklist database can cause "collateral damage" if it contains IP addresses that should not really be blacklisted. This "deny all" proposition for blacklist members is grounded on the fact that blacklist members are considered to be malicious sources. So, stopping traffic coming from or destined to the blacklist source can preemptively protect hosts within the network protected by the blacklisting firewall. Blacklist firewall rules will have the following form: {*,*,…,*, IP_x ,*,*,…,*, deny}. The * symbol is the wildcard character that represents all possible values for that particular field. This rule form means to deny access to or from the blacklist member IP_x for any set of possible field values. Since all blacklist rules will have the same action, i.e. deny, they will be independent of one another according to definition 3.1.1. Recall that the *de facto* firewall search technique is based on a linear search first match (LSFM) algorithm. With LSFM, when an incoming packet arrives at the firewall, the appropriate fields (**p**) are extracted and each rule is analyzed starting from $\mathbf{R_1}$ and proceeding until a match is found or until the end of the rule list is reached. If no match is found in the rule set, then a default action is applied to the packet. If a rule set has N rules, then LSFM based firewalls have an O(N) search time complexity. This is a problem that must be solved in order for firewall blacklisting to be realizable. Without loss of generality and without changing the internal logic of the rule set, the blacklist rules can be placed at the top of the rule set. Any blacklist rule should be placed before any other rule in which it is correlated and in which the other rule has an allow action. Therefore, blacklist rules can be placed before any other rule in the rule set without changing the input-output behavior of the firewall. Given these observations along with definitions 3.1.1 and 3.1.2, we can design a firewall from a new perspective. We can create a firewall rule processing system that classifies based solely on an incoming packet's source and destination address against a single point-wise list of blacklist IP addresses followed by normal firewall classification. Figure 1 illustrates the design.

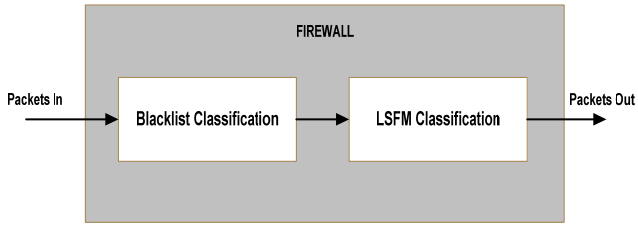

Figure 1. The blacklisting firewall design concept.

Since the structure of the firewall can be changed as seen in Figure 1, high-speed memory-efficient search algorithms can be used in the blacklist classification firewall subsystem. The basic concept is simple. Given an incoming packet, extract **p**. Then, send the source and destination IP addresses contained in **p** into the blacklist classification subsystem. If either source or destination is found in the blacklist, deny the packet. Otherwise, send **p** to the LSFM classification subsystem, which contains the local security policy defined by the network administrator. If LSFM finds a matching rule, execute the action of the matching rule with highest priority. Otherwise, execute the default action. The next section describes two bit vector based algorithms that can implement the blacklisting subsystem classification. One is suitable for software implementation whereas the other is suitable for software or hardware implementation.

4. BIT VECTOR BASED FIREWALL BLACKLISTING

The blacklist rules can be stored as a list of IP addresses. IP addresses are normally written in dotted-decimal notation such as 10.10.10.10. However, this notation represents an integer on the interval $[0, 2^{32}-1]$. Therefore, the blacklist can be represented as a list of sorted integers and classification can be performed by a simple binary search. This gives a worst case search time complexity of O(lgN). However, bit vectors can be used to give better search performance. For the fastest case, one can create a bit vector to do the entire search operation. Let the bit vector be denoted as **b[i]**. Then, the bit values of **b[i]** are determined as follows. Let **i** be equal to the integer value of the blacklist IP

address. Then, **b[i]** = 1 $\forall$ IP addresses $\in$ blacklist and **b[i]** = 0 otherwise. Given this technique, the integer value of the IP address to be classified is used as an index into the bit vector. If the value is 1, then deny. Otherwise, send **p** to the LSFM subsystem. This technique provides an O(1) search time complexity. However, for IPv4, this will require 2^{32} bits = 0.5 GB of memory. This memory consumption is not feasible for general purpose firewalls. The first bit vector based algorithm this paper introduces, EBVBL, is a coupled approach using a bit vector and binary search.

4.1 EBVBL: The Enhanced Bit Vector Based Blacklisting Algorithm

The goal of this algorithm is to use a bit vector to perform fast classification on IP addresses that are *not* in the blacklist. IPv4 addresses are represented with 32 bits. We define the bit field factor **f** = 32 – **α**, the bit vector radius **bvr** = $2^{\alpha} = 2^{32-f}$ and the bit vector size **bvs** = 2^{f}. The bit vector **b[i]** will consume 2^{f} bits. The bit vector indexing is performed as follows. For all IP addresses in the blacklist, extract the first **f** bits of the address and calculate the integer value of these bits and store into variable **x**. Then, set **b[x]** = 1. All other bit vector values are set to 0. If **b[x]** = 0, then the IP address is definitely not in the blacklist. However, for a given **f**, a total of **bvr** IP addresses will index (or collide) into a given bit vector location. So, when classifying an IP address, if **b[x]** = 1, further processing will be required. In this algorithm, if **b[x]** = 1, the algorithm will perform a binary search on the 32-bit integer value of the IP address against the blacklist represented as a sorted list of their integer values. The performance of this algorithm depends on **f**. To determine the optimal values, one can plot **bvs** and **bvr** as a function of **f** as shown in Figure 2.

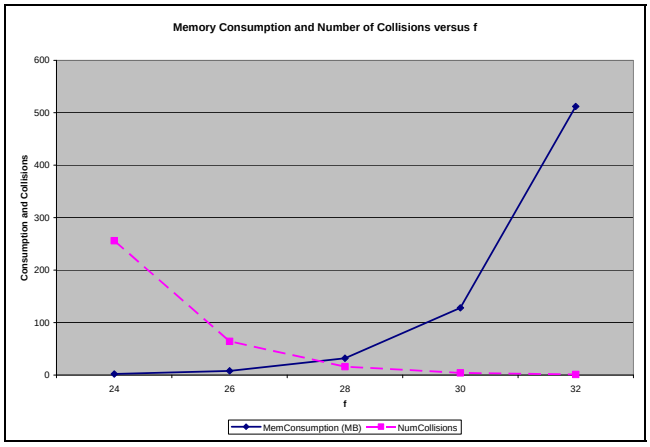

Figure 2. Memory consumption and total collisions versus f.

The solid, increasing curve in Figure 2 represents the total amount of memory (**bvs**) consumed by the bit vector versus **f**, and the dashed, decreasing curve represents the total number of collisions (**bvr**) versus **f**. If one considers the intersection of these two curves as an equilibrium point, then the closest integer value to equilibrium is **f** = 28. This gives a total memory consumption of **bvs** = 2^{28} bits = 32 MB, and a total collision space of **bvr** = 2^{4} = 16 IP addresses per bit vector entry (the collision only matters if the bit is set to 1). The design parameter **f** can be modified depending on the total amount of memory available to the target system.

Algorithm EBVBL:
Input: An initialized bit vector **b[i]**, a binary valued IP address IP_x contained in **p** to be classified, and a blacklist set {**BL**} of IP addresses as sorted integers.
Begin:
if (**b[** toInteger(IP_x(1 **..** **f**)) **]** = = 0)
then LSFM(**p**);
else if (binarySearch(toInteger(IP_x(1 **..** 32)), {**BL**}) = = 0)
then LSFM(**p**);
else
deny(**p**);
End Algorithm EBVBL

4.2 PMBL: The Path Matrix Bit Vector Based Blacklisting Algorithm

PMBL makes use of bit vectors from a different perspective. IPv4 uses a 4 octet valued representation where each octet is in the range [0, 255]. An IP address is defined as $x_1.\ x_2.\ x_3.\ x_4$ where $x_i \in [0, 255]$. If one considers an octet space, then an IP address can be represented as a path P through the octet space: $P = x_1 \rightarrow x_2 \rightarrow x_3 \rightarrow x_4$ and this path is unique in the octet space. P can be represented by a set of *path matrices* $\{\mathbf{P_i}\}$. A path matrix $\mathbf{P_i}$ represents an octet subspace for x_i. The dimension of $\mathbf{P_i}$ is $N \times 256$. The columns of $\mathbf{P_i}$ are indexed by x_i and each row represents a unique point in the i^{th} dimension of the octet space. For a k-dimensional octet space (k = 4 for IPv4 and k=16 for IPv6), the N blacklist IP addresses are encoded as follows: the r^{th} IP address of the blacklist sets bits as $\mathbf{P_i}(r, x) = 1$ for $x = x_i$ and $\mathbf{P_i}(r, x) = 0$ for $x \neq x_i, 0 \leq x \leq 255$, $\forall$ $\mathbf{P_i}$, $1 \leq i \leq k$, $\forall$ r, $0 \leq r \leq N$. Each row of the set $\{\mathbf{P_i}\}$ represents a unique path P in the octet space. Each column of $\mathbf{P_i}$ represents a bit vector for each x_i where the column element has a bit set to 1 *iff* that octet value is present in a particular path. For example, if the r^{th} blacklist IP address is 10.20.10.30, then $\mathbf{P_1}(r, 10) = \mathbf{P_2}(r, 20) = \mathbf{P_3}(r, 10) = \mathbf{P_4}(r, 30) = 1$. Classifying an IP address consists of indexing into $\mathbf{P_i}$ with its associated octet value x_i and extracting the bit vector in the x_i^{th} column. Then, perform a conjunction on each extracted column. If the conjunction results in one bit set at the r^{th} location, then the IP address is the r^{th} member of the blacklist. If the IP address is not in the blacklist, then the conjunction will result in all bits set to 0. We now formally describe the algorithm.

Algorithm PMBL:
PM_Initialize:
Input: A blacklist set of IP addresses {**BL**} in dotted decimal format.
Begin:
//set all bits in each path matrix to zero, then
//set appropriate bits to 1
$P_i(*,*) = 0$ with $1 \leq i \leq k$;
for(r = 1; r ≤ N; r++) {
for(i = 1; i ≤ k; i++) {
x_i = **BL**(r)→octet_i;
$P_i(r, x_i) = 1$;
}
}
End PM_Initialize

PM_Classify:
Input: An IP addresses IP_x contained in **p** in dotted decimal notation to be classified.

Begin:
//bit wise AND the set of bit vectors in P_i indexed by octet
R(*) = P_1(*, IP_x→octet_1) **&** P_2(*, IP_x→octet_2) **&**
… **&** P_k(*, IP_x→octet_k)
//If the result of the conjunction results in all 0, call LSFM
//Otherwise, deny the packet
if((R || R) == **false**)
then LSFM(**p**);
else
deny(**p**);
End PM_Classify;

5. ALGORITHM ANALYSIS AND EXPERIMENTAL EVALUATION

The EBVBL algorithm has O(lgN) worst case time complexity and requires O(N + **bvs**) memory. This bit vector technique provides opportunity for firewall performance optimization, but it depends on the statistical properties of the network traffic through the firewall with respect to the blacklist. If the firewall experiences small amounts of traffic that belong to blacklist members or from IP addresses that fall within the bit vector radius of a blacklist member, then most of the blacklist classification results will be provided by a single bit vector access. But, in the worst case, the blacklist classification will be no more than O(lgN). This is a vast improvement over the O(N) time complexity of LSFM blacklisting. However, the bit vector technique of EBVBL does not scale to IPv6. Considering that IPv6 IP addresses are 128 bits, creating a bit vector would be of no use because there will be a huge collision space. If **f** = 28, then **bvr** = 2^{100}. But, binary search techniques can be used for IPv6 blacklisting.

The PMBL algorithm can be implemented in software or hardware. The algorithm requires O(βN) memory where β = 128 bytes for IPv4 and β = 512 bytes for IPv6. For example, the algorithm can encode 2^{16} IPv4 addresses with only 8 MB of memory. The algorithm is computationally efficient because the bulk of operations being performed in the algorithm are elementary bit operations on W_s-wide chunks of the bit vector where W_s is the word size of the memory system. There are (k-1) bit-wise AND operations on each W_s chunk. Therefore, the search time complexity of PMBL implemented in ***software*** is O((k-1)N/W_s). Because the algorithmic operations of PMBL include bit vectors and bitwise AND and OR, the system can be implemented in hardware as an Application Specific Integrated Circuit (ASIC). Notice that if W_s → N and if the bit wise AND is done in parallel, the algorithm is O(1). In hardware, the algorithm can classify a blacklist IP address on the order of a few clock cycles (clock cycle time will be hardware specific). Figure 3 is a very high level block diagram of the hardware implementation for k = 4.

Several simulations were conducted so that a performance comparison between EBVBL, PMBL, and LSFM could be made. These simulations were performed in a Redhat Enterprise Linux 5.2 operating system environment with an Intel Core2 Duo 2.2 GHz CPU and 2 GB RAM, and the simulation code was written in JAVA (JAVA provides the BitSet class that implements bit vector data structures). Note that these results are only comparing the software implementations. This paper does not include a study of the PMBL hardware implementation and its timing simulations. For all results shown, the search times are given in units of nanoseconds. For LSFM, worst case performance will occur when an IP address to be classified is not in the blacklist, i.e. an exhaustive search will always happen for IP addresses not in the blacklist. The experimentation with these algorithms consisted of 4 types of firewall inputs and included simulations where the probability that an incoming IP address to be classified had probability p equal to 0, 0.25, 0.50, and 0.75 of being a member of the blacklist. The 4 simulations used a blacklist of 65,536 members.

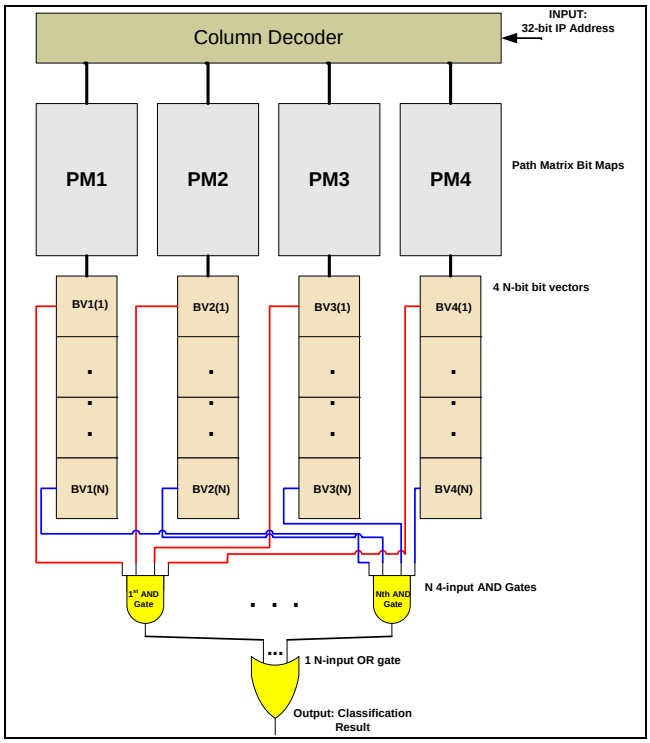

Figure 3. PMBL hardware block diagram.

Figure 4 shows the search time results for the worst case scenario, which implies probability p = 0 that the IP to be classified is in the list.

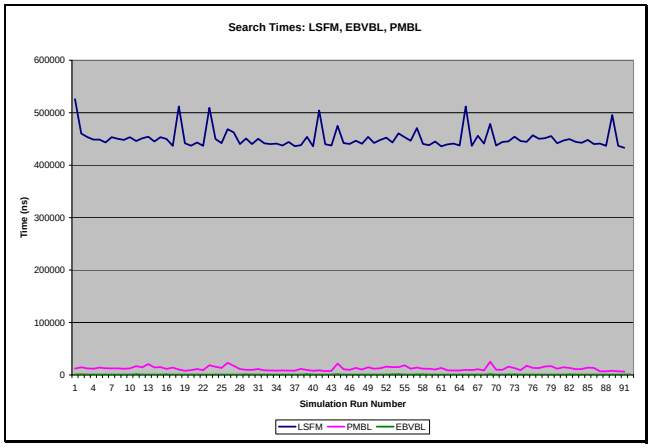

Figure 4. Worst case search time for LSFM, EBVBL, and PMBL with p = 0.

For the worst case simulation, LSFM is almost a 1000 times slower than EBVBL and approximately 50 times slower than

PMBL. Figure 5 shows the results of the same simulation as Figure 4, but without plotting LSFM search times. Note that in the worst case scenario, no incoming IP address to be classified will be in the blacklist. So, unless the incoming IP address falls within the bit vector radius of a blacklist member, the EBVBL classification times will be O(1).

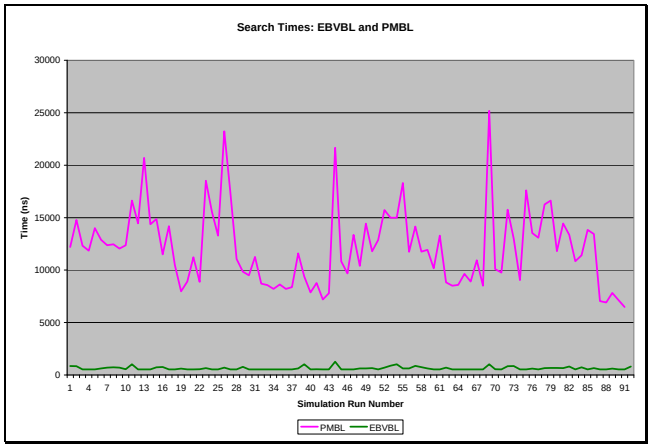

Figure 5. Worst case search time for EBVBL and PMBL with p = 0.

Figure 6 shows results for a probability of 0.75 that an incoming IP address will be in the blacklist. Comparing Figures 5 and 6, one can see that there is a definite gain in the EBVBL algorithm when the probability approaches zero that an incoming IP address into the firewall will be in the blacklist.

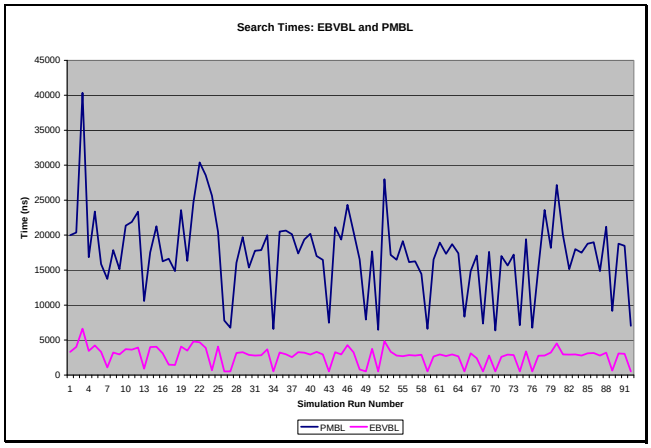

Figure 6. Search time for EBVBL and PMBL with p = 0.75.

Table 1. Average search times for the 4 simulations.

Probability	LSFM	EBVBL	PMBL
0	600676	654	12458
0.25	736138	1645	17095
0.5	413480	2400	19170
0.75	587539	3077	21752

Table 1 shows the average search time results for all 4 probability cases for the LSFM, EBVBL, and PMBL algorithms. The average search times are in units of nanoseconds. As seen from these results, the EBVBL algorithm is the best choice of the three algorithms for a software implementation.

6. CONCLUSION

This paper introduced the EBVBL and PMBL algorithms for enabling high-speed memory-efficient firewall blacklisting. The paper discussed how the firewall can be redesigned such that blacklisting can be performed in the firewall. The idea is to separate the blacklist classification from the classical *de facto* LSFM based firewall classification and apply point-wise search algorithms on IP addresses for blacklist classification. As the results show, EBVBL performs best when the algorithms are implemented in software. However, PMBL can be implemented in hardware where classification can be performed on the order of a few clock cycles. We believe that the future trends of next generation networking and security will cause a shift from implementing blacklists at the application layer to implementation at the firewall level. The benefit is that the firewall offloads the blacklisting workload from the application. This allows system resources at the application level to be used for their true purposes. Further, providing blacklisting at the firewall can preemptively protect all hosts, not just the application hosts, within the network protected by the firewall.

7. REFERENCES

[1] Al-Shaer, E., Hamed, H. 2004. Discovery of policy anomalies in distributed firewalls. In Proceedings of the 23rd Annual Joint Conference of the IEEE Computer and Communications Societies. INFOCOM'04. vol. 4. pp. 2605-2616. IEEE.

[2] Gouda, M., Liu, X. 2004. Firewall design: consistency, completeness, and compactness. In Proceedings of the 24th International Conference on Distributed Computing Systems. ICDCS'04. pp. 320-327. IEEE Computer Society.

[3] Gupta, P. 2000. Algorithms for routing lookups and packet classification. PhD Dissertation. Stanford University, Department of Computer Science.

[4] Kim, H., Kang, I. 2003. On the effectiveness of martian address filtering and its extensions. In Proceedings of the 2003 Global Telecommunications Conference. GLOBECOM'03. pp. 1348-1353. IEEE.

[5] Lakshman, T.V., Stiliadis, D. 1998. High-speed policy-based packet forwarding using efficient multi-dimensional range matching. SIGCOMM Computer Communication Review, vol. 28, no. 4, pp. 203-214.

[6] Li, J., Liu, H., Sollins, K. 2003. Scalable packet classification using bit vector aggregating and folding. MIT LCS Technical Memo. MIT-LCS-TM-637. MIT Laboratory for Computer Science. Cambridge, MA.

[7] Soldo, F., Defrawy, K. Markopoulou, A. 2008. Filtering sources of unwanted traffic. In Proceedings of the 2008 Information Theory and Applications Workshop. pp. 199-208. IEEE.

[8] Zhang, J., Porras, P., Ullrich, J. 2008. Highly predictive blacklisting. In Proceedings of the 17th USENIX Security Symposium. pp. 107-122.