Classifying DDoS packets in high-speed networks

Yang Xiang and Wanlei Zhou

School of Engineering and Information Technology, Deakin University, Victoria, Australia

Summary
Recently high-speed networks have been utilized by attackers as Distributed Denial of Service (DDoS) attack infrastructure. Services on high-speed networks also have been attacked by successive waves of the DDoS attacks. How to sensitively and accurately detect the attack traffic, and quickly filter out the attack packets are still the major challenges in DDoS defense. Unfortunately most current defense approaches can not efficiently fulfill these tasks. Our approach is to find the network anomalies by using neural network and classify DDoS packets by a Bloom filter-based classifier (BFC). BFC is a set of space-efficient data structures and algorithms for packet classification. The evaluation results show that the simple complexity, high classification speed and accuracy and low storage requirements of this classifier make it not only suitable for DDoS filtering in high-speed networks, but also suitable for other applications such as string matching for intrusion detection systems and IP lookup for programmable routers.

Key words:
DDoS attacks, packet classification, high-speed network, intrusion detection.

1. Introduction

Computer networks and the Internet have now evolved into a ubiquitous information infrastructure. High-speed backbones and local area networks (wired or wireless) provide the end-user with bandwidths that increase rapidly, linking millions of end-users to many critical services. In the past a few years, companies, organizations and government agencies have been attacked by successive waves of Distributed Denial of Service (DDoS) attacks [8]. A DDoS attack is characterized by an explicit attempt from an attacker to prevent legitimate users from using the desired resource [21]. The attacker usually recruits thousands of hosts as zombies to launch the attack by sending malicious packets from multiple sites towards a single target at the same time. The target then will be flooded and out of service to legitimate users, such as downtime of web servers.

The rapid development of high-speed networks has spurred new applications and has in turn been driven by the popularity of those applications. However, it also provides DDoS attackers advantages to start an attack. Although many defense approaches have been proposed to fight against DDoS attacks, such as filtering [6, 14], traceback [1], congestion control [7, 9] and replication [13, 23], it is still difficult to separate unambiguously the attack traffic from legitimate traffic, and then quickly remove the attack traffic, especially when the ongoing traffic volume is high. There are two major challenges of DDoS defense in high-speed networks. One is to sensitively and accurately detect attack traffic, and the other is to quickly filter out the attack traffic, which mainly depends on high-speed packet classification [10]. Here packet classification means the process of classifying packets into normal or attack flows in a router. Since packet classification has been one of the major bottlenecks in routers that enable security services, a fast packet classification algorithm is critical to a router-based DDoS defense system.

To address the first challenge mentioned above, we use neural networks to differentiate normal and abnormal traffic [20] by Mark-Aided Distributed Filtering (MADF) system. By the aid of the marks of a packet marking traceback scheme, Flexible Deterministic Packet Marking (FDPM), in the IP header [22], this system can accurately separate the attack packets from the legitimate packets.

This paper mainly addresses the second challenge, which is to quickly classify DDoS attack packets in high-speed networks. Our major contribution is that we propose a Bloom filter-based packet classification scheme for classifying DDoS packets. Specifically, we solved problems in high-speed packet classification by improving many performance metrics. First, the classification speed is high. For example, it can quickly filter out the packets in a speed at 7.6 Gb/s, accordingly greatly improves the legitimate traffic throughput and reduces the DDoS attack traffic throughput. Second, the memory that this scheme consumes is very limited, compared to other schemes such as H-Trie [10]; thus it makes possible for hardware implementation. Third, this scheme greatly reduces false positive rate that is brought by a traditional Bloom filter [2]. If the assumption of perfect hash function is true, this scheme is virtually near *zero* false positive. In this paper we propose this scheme for DDoS packet classification. However, it can be widely applied to solve other classification problems such as string matching for intrusion detection systems and IP lookup for programmable routers [18].

Manuscript received January 2006.
Manuscript accepted Feburuary 2006.

The rest of paper is organized as follows. Section 2 briefly introduces the background of the traditional Bloom filter. Section 3 presents the system design of the DDoS defense system and the Bloom filter-based classifier in it. Section 4 shows the experiments and performance of this scheme. Related work and some discussion are presented in section 5. Section 6 summarizes this paper.

2. Background of Bloom Filter

A Bloom filter is a simple space-efficient randomized data structure for representing a set in order to support membership queries [2, 3]. The space efficiency is achieved at the cost of a small probability of false positives. Here we briefly introduce the Bloom filter theory.

A Bloom filter for representing a set $S=\{x_1, x_2, \ldots, x_n\}$ of n elements is described by an array of m bits, initially all set to 0. It uses k independent hash function $h_1, \ldots, h_k$ with range $\{1, \ldots, m\}$. Here we have an assumption that hash functions are perfectly random, which means the hash functions map each item in the universe to a random number uniform over the range $\{1, \ldots, m\}$. For each element $x\in S$, the bits $h_i(x)$ are set to 1 for $1\le i\le k$. A location can be set to 1 multiple times, but only the first change has an effect. For the membership query if $y\in S$, we check if $\forall i$, $h_i(y)=1$. If $\exists h_i(i)\neq 1$, then $y\notin S$. If $\forall i$, $h_i(y)=1$ is true, we can assume $y\in S$ with a false positive rate as

$$f=\left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \approx \left(1-e^{-\frac{kn}{m}}\right)^{k}=\exp\left(k\ln(1-e^{-kn/m})\right) \quad (1)$$

Let

$$g=k\ln(1-e^{-kn/m}) \quad (2)$$

And we have

$$\frac{dg}{dk}=\ln(1-e^{-kn/m})+\frac{kn}{m}\frac{e^{-kn/m}}{1-e^{-kn/m}} \quad (3)$$

Because minimizing the f is equivalent to minimizing g with respect to k, we have when the above equation equals 0, $k=\ln 2\cdot(m/n)$. Then the optimized false positive is $(0.6185)^{m/n}$.

3. System Design

3.1 Overview

Before presenting the details about the Bloom filter-based classifier (BFC), we need briefly introduce some background about Flexible Deterministic Packet Marking (FDPM) [22] and Mark-Aided Distributed Filtering (MADF) [20]. We do not discuss how to gather intelligence and set signatures in order to drop attack packets here, which is mainly MADF's work. Instead, in this paper, we discuss packet classification based on known signatures.

As shown in figure 1, the MADF system has an Offline Training System (OTS) and an Online Filtering System (OFS) and is deployed between the source end (one hop behind FDPM encoding module) and the victim end. The FDPM encoding modules are deployed at the edge routers that are close to the attack source end. When packets enter the network, they are dynamically marked by the encoding modules. The real source IP addresses of the entry points are stored in the marking fields. When the packets reach the victim end, the source IP addresses of entry points can be reconstructed.

Packets are tapped into both OTS and OFS. OTS is a lightweight neural network with back-propagation algorithm [11], which consists of three parts, data collecting part, training part and rules generating part. It is usually deployed close to the victim end, in order to obtain better training result. The trained neural networks are transferred back to OFS for testing. Once the packets are identified as the attack packets, they will be filtered out by the Bloom filter-based classifier (BFC).

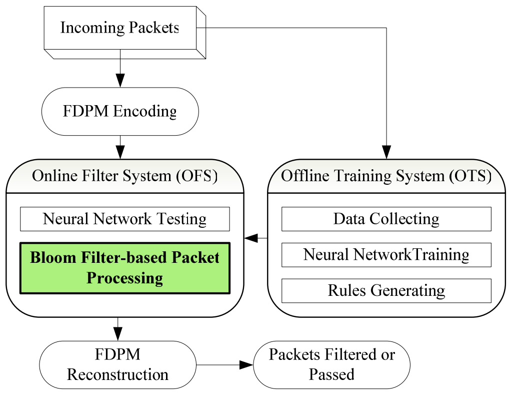


Fig. 1 System architecture.

In DDoS packet filtering problems, packet classification becomes a two-category classification process. While

Bloom filter provides good space and speed efficiencies with low false positives, it offers a fast decision making function to filter the attack packets. The OFS can be deployed at any point in the protected network. If it is deployed close to the attack source end, it can protect even better the rest of network from it to victim, because the attack traffic has been removed before it travels to the victim, without causing overall network congestion.

3.2 Online Filtering System

The Online Filtering System (OFS) is the key sub-system that enables the filtering function. We test the incoming packets by the trained neural network that transferred from the Offline Training System. If the output indicates anomalies, we further investigate the composition of the packets. If the number of packets that have the same address digest bits that marked by FDPM exceeds a threshold N_{drop}, this flow of packets will be filtered. After the attack packets are identified, it turns into the packet classification phase.

For each incoming packet, BFC examines the bits in IP header for DDoS signatures. In our experiments, the bits are 8 bits of TOS, 16 bits of Fragment ID, 1 bit of Reserved Flag and 32 bits of Destination IP Address. Justification of using these bits can be found in [22]. Once a match of signature is found, the packet is dropped. BFC has no false negative (missing a real DDoS packet) and only an extremely low false positive rate (in order of 10e-15).

3.3 Data Structures and Algorithms

The data structures of the Bloom filter-based classifier (BFC) are shown in figure 2. The filter is an array of data structure BFC, which length is decided by the number of element (the same as the number m discussed in section 2). An element BFC consists of two field. One bit hit is of the same purpose as in the traditional Bloom filter, which is used to test if the member exists. The second field in the BFC element is a pointer lead to a link list of structure l_hit_array, which is introduced below. The node of link list l_hit_array has two fields, one is a 16-bit word hit_index_sum, and the other is the pointer that leads to the next node, if the next one exists.

There are two steps of the classification by BFC. One is construction of BFC, and the other is membership testing. Here we have the assumption that the hash functions used are perfectly random. That is, for the same signature S and different hash functions H_i and H_j ($i \neq j$), the hashed values are always different.

```
BFC filter[int NUMBER_ELEMENT];

struct BFC {
  bit hit;
  struct l_hit_array* list;
};

struct l_hit_array {
  bit hit_index_sum[16];
  struct l_hit_array* next;
};
```

Fig. 2 Data structures of BFC.

The construction of BFC is shown in figure 3. Initially, each hit bit in the element BFC is set to 0 and the pointer list is set to null. Then each attack signature S_i, $i \in [1, n]$ is hashed by function H_j, $j \in [1, k]$ with corresponding hit bit in BFC being set to 1. A new node of link list l_hit_array is created with the hit_index_sum field being filled by the sum of previous value and the last 16 bits of the index value of the filter that are being set to 1. The field hit_index_sum provides the ability to reduce false positives. In another word, it is used to determine which exact signature causes the value 1 set by hash functions. In traditional Bloom filter, there is no such detection mechanism thus different signatures will possibly result in the same bit being set to value 1. The attack signature S_i, $i \in [1, n]$ are hashed k times. If the bit has already been set to 1, a new node of link list l_hit_array is appended to the list. Note the index of the filter array usually can be an integer with 32 bits (it can also just be a short integer with 16 bits), while we only utilize 16 bits for hit_index_sum. This design does not affect much accuracy because in all the experiments the false positive rates are the same. However, this design saves nearly half of memory space for the link list structure. If the addition procedure has overflow, the overflow bit will be discarded to maintain this 16-bit structure. The algorithm of construction of BFC is shown in figure 4.

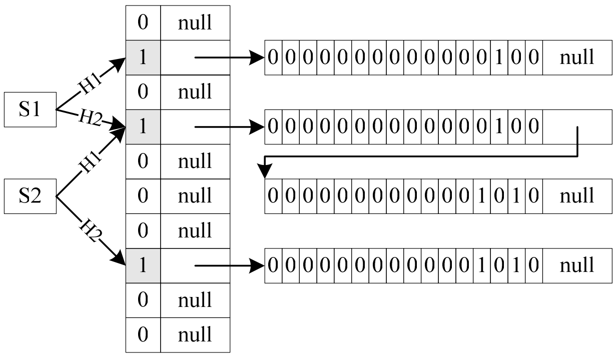


Fig. 3 Construction of BFC.

```
for each signature S do
  for each hash function H do
    index=H(S);
    if(BFC[index].hit!=0)
      set BFC[index].hit=1;
      create a node of l_hit_array;
    else
      append a new node of l_hit_array;
    endif
    set all corresponding entries for the current siganature's
      node.hit_index_sum=previous value+last 16 bits of index;
  end do
end do
```

Fig. 4 Algorithm of construction of BFC.

In the membership testing step, for each query *T*, we compute hashed value by H_j, $j\in[1, k]$. The sum of last 16 bits of all hashed values (compare_sum) is kept in the memory. Once any hashed bit is not equal to the corresponding bit in the preset BFC array, then it means the query does not belong to signature base. After all the hash procedures are finished, we search the current list for the same value of compare_sum. If no matching value is found, it means the query does not belong to signature base. If any match is found, then the result of query is true. Figure 5 shows some examples of membership testing. For instance, *T1* and *T3* are not the members of signature base, and *T2* is a true match. Our scheme is free of false positive that occurs in the traditional Bloom filter. For example, the query *T3* has all hashed value of 1. In traditional Bloom filter, it will be falsely classified as a match. In this scheme, this case will not happen. More precisely, the exact matching signature can be found. This is especially beneficial if multiple signatures lead the same bit to be set to 1. The algorithm of membership testing of BFC is shown in figure 6.

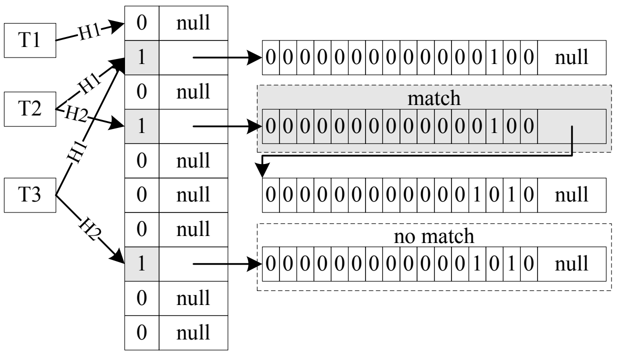

T1
T2
T3
H1
H1
H2
H1
H2
0 null
1
0 null
1
0 null
0 null
0 null
1
0 null
0 null
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 null
match
0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 null
no match
0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 null

Fig. 5 Membership testing of BFC.

```
for each query T do
  for each hash function Hj do
    index=Hj(T);
    if(BFC[index].hit==0)
      no match, return false;
    else
      set compare_sum=previous value+last 16 bits of index;
    endif
  end do
  if(compare_sum in BFC[index].list's hit_index_sum)
    match, return true;
  else
    no match, return false;
  endif
end do
```

Fig. 6 Algorithm of membership testing of BFC.

Updating BFC is straightforward and easy. Adding an entry of signature is the same as the steps of construction of BFC. To delete an entry of signature, for all the hashed indexed entries, the corresponding node in l_hit_array is deleted. Then if the link list is empty, reset the hit bit in the BFC elements to 0.

3.4 Optimization

From the above section we can see that in the construction steps of BFC when signature S_i is hashed by each hash function H_j, all the corresponding nodes in the l_hit_array must be updated. This not only consumes computing resource, but also wastes memory. We optimize the first design by modifying the data structure into figure 7. In the optimized design, we move the bit array of hit_index_sum into a separate list called sum_list. In l_hit_array only a pointer that leads to the hit_index_sum is kept. In sum_list, there is a pointer that leads to the signature array. This is for the case when we need find out which signature does the query *T* match (It is used in intrusion detection system). Figure 8 illustrated the construction steps of optimized BFC.

```
struct l_hit_array {
  struct sum_list* p_sum;
  struct l_hit_array* next;
};

struct sum_list {
  bit hit_index_sum[16];
  signature* p_signature;
  struct sum_list* next;
};
```

Fig. 7 Optimized data structure of BFC.

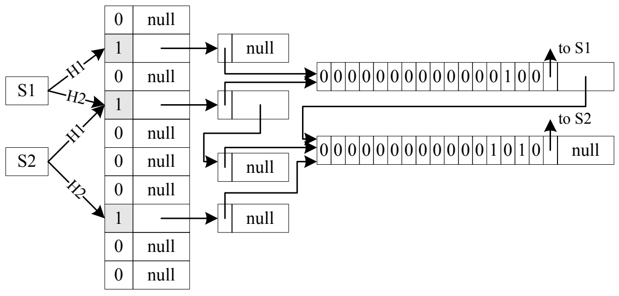


Fig. 8 Construction of optimized BFC.

3.5 False Positive

This scheme reduces the false positive that is a serious problem in traditional Bloom filter, as it is discussed in section 3.3. However, it still generates false positive caused by the calculation of hit_index_sum, because different combination of index sets still will result in the same value of sum. Therefore, the false positive rate of this scheme *f* can be loosely written as

$$f = \left(1-\left(1-\frac{1}{m}\right)^{kn}\right)^{k} \frac{1}{\prod_{i=0}^{k}(m-i)} \qquad (4)$$

Where *m* is the number of element of BFC, *n* is the number of signatures, *k* is the number of hash functions used. From the equation we can theoretically compute the false positive rate for this scheme. For example, when *n*=1000, *m*=10000, the optimized *k* for traditional Bloom filter is *k*≈7 and false positive *f*=8.19e-3. On the other side, for BFC, *f*=5e-15, which is much smaller than the one of traditional Bloom filter. Therefore, we can safely see that if the assumption of perfect hash function is true, this scheme is virtually near *zero* false positive. The experiments below will again verify this.

4. Experiments and Evaluation

4.1 Introduction

In the project of Distributed Denial of Service Simulators at Deakin University [5], two DDoS tools, TFN2K and Trinoo, are adopted and integrated into SSFNet simulator [17] to create virtual DDoS networks to simulate the attacks. The TFN2K and Trinoo are originally written in C language and are ported to Java to be embedded into SSFNet. By using the DDoS simulators, we can launch any DDoS attack with different features such as duration, protocol, attack rate, etc. In order to simulate the DDoS attack as real as possible, we also use the real Internet topology from Cooperative Association for Internet Data Analysis (CAIDA)'s Skitter project [4]. The data set used is generated from server aroot ipv4.20040120 on 09/Jan/2004. To simplify the problem, we connect all routers by 100M network interfaces. We randomly choose the 1000 attack hosts and let the rest be legitimate clients, and let the Skitter server be the victim. Constant rate attack of 300Kbps is applied to all attack hosts. According to the hop distribution (number of routers between the victim and its clients), most of the clients locate in the distance between 10 hops and 25 hops. Therefore, we deploy the FDPM encoding module at routers 10 hops from the victim, and the MADF at routers from 1 to 9 hops from the victim. Then the MADF generate the signatures for packet classification. The signatures in this simulation consist of two parts, one is the desination IP address (32 bits) and the other is the mark from FDPM (25 bits). Therefore, the total length of the signature is 57 bits.

4.2 Metrics of Evaluation

There are many metrics for packet classification algorithms such as search speed, low storage requirements, low false positives, fast updates, and flexibility. We summarize these metrics in table 1. Different classification applications have different requirements. For this DDoS packet classification application, search speed, low storage requirements and low false positives are the major goals. For fast updates, a relative low update rate is sufficient for this application because there is no need to change the signatures all the time, which require very frequent updates. We only need to classify packets into two categories, attack packets and normal packets. We also consider flexibility is not an obligatory requirement because the classification problem here is not a general purpose application. In section 5.2 we will discuss the fast update metric for an intrusion detection application.

Table 1 Metrics of classification

Metric	***Description***
Search speed	The speed to find the matched rules. Faster links require faster classification.
Low storage requirements	Small storage requirements enable the use of fast memory technologies like static random access memory (SRAM).
Low false positives	False positives is tolerable if the value is very small. Ideally it is zero.
Fast updates	The classifier changes from time to time, therefore fast and incrementally update of the data structure is essential to a good classification algorithm.
Flexibility	The capability to support general rules, including prefixes, operators and wild cards.

4.3 Performance

First, we test the search speed of BFC. The scheme is tested on a PC with Intel Celeron CPU 2.0GHz, 512M RAM. For comparison, the Hierarchical Tries algorithm is also implemented. This algorithm is an extension of the one dimensional radix trie data structure. Table 2 shows the search time for different number of signatures. When the number of signatures is 800, for example, BFC is on average 354% faster than a traditional H-Trie classifier. Under the same condition, the BFC can achieve the average search time at about 33.8ns and the maximum search time at about 42.6ns. This means the classifier can process at least 23.8 million packets per second. If we assume the minimum length of an IP packet is 40 bytes, this classifier is power enough to process packets at 7.6 Gbps speed, which meets the requirement of most current high-speed networks.

Table 2 Search time (ns)

number of signatures	***H-Trie***		***BFC***	
	Averag e	Maximum	Averag e	Maximum
50	143.3	157.2	27.1	37.1
100	146.2	159.7	30.4	37.3
200	148.4	161.4	30.3	38.4
300	149.3	162.6	30.7	38.7
400	149.2	163.2	31.4	39.4
500	150.1	163.7	31.5	39.4
600	152.4	164.6	32.9	41.5
700	151.9	168.1	33.5	42.7
800	150.4	167.4	33.8	43.6
900	152.4	169.2	33.9	44.1
1000	153.4	169.6	34.1	44.1

On different platforms the search time will be different depending on the power of the hardware and operating system. Therefore, another metric that can reflect search speed is the average number of memory access per search. This value is algorithm-dependent. From figure 9 we can see this scheme requires a very small number of memory access per search, which is below 10 times for 1000 signatures. Moreover, when the number of signatures increases, the number of memory access per search does not increase in a direct ratio. Instead, it increases slowly. When it is tested in the condition of 10000 signatures, the average number of memory access per search is 40. This proves the scheme's search speed is high when scalability is concerned.

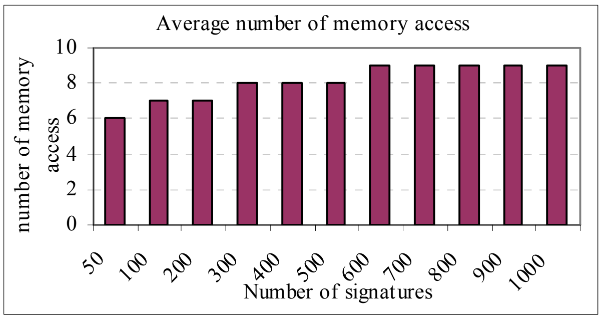


Fig. 9 Average number of memory access.

Second, we test the memory consumption of BFC. Although in our experiments the memory consumption is not a remarkable issue, because there are usually less than 500 rules to be used in the classification for DDoS filtering, the BFC still shows good storage efficiency compared with the H-Trie classifier. For example, for the 500-rule test in figure 10, our classifier consumes only 2.259% of what an H-Trie classifier needs. On scalability, even if tested by up to 10000 artificial generated rules (not by DDoS tests) the Bloom filter-based classifier only needs less than 3M memory. This proves it has potential to be applied for other more memory-consuming applications.

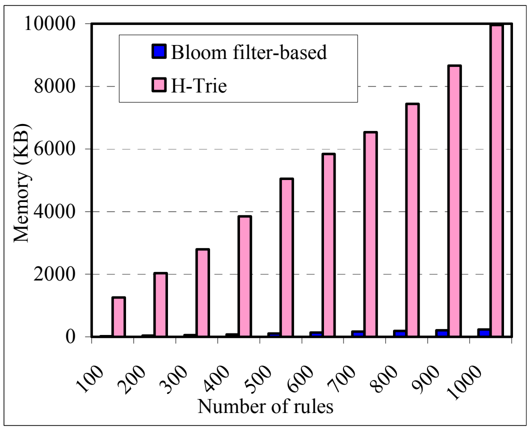


Fig. 10 Comparison of memory consumption.

Third, we test the false positive rate of BFC. Unlike some kinds of classification schemes that are free of false positives, BFC do theoretically have false positives, as we analyzed in section 3.5. However, practically, in our experiments all the tests show zero false positive. This confirms the conclusion that BFC is virtually near *zero* false positive.

5. Related Work and Discussion

5.1 Related Work

Fast packet classification is an important issue to deal with the DDoS attacks, which is inherently difficult. Currently there have been many different approaches for packet classification. Trie structure is the most popular approach used so far. Hierarchical trie [16] is constructed by building one-dimensional binary trie on each field recursively. The problem of this approach is the difficulty in finding all possible matches. Set-pruning trie proposed to copy signatures for all possible paths, and grid of trie proposed to use switch pointers directing next signature. These approaches still have an issue that they require a lot of pre-processing and are not easily applied to multiple dimensions [10].

Linear time search or parallelisms are used to search through all signatures sequentially, such as multi-dimensional range matching [16] and Ternary-CAMs [19]. However, these solutions will be expensive when the DDoS filtering signature sets are large. Heuristic methods such as Recursive Flow Classification (RFC) [9], Hierarchical Intelligent Cuttings (HiCuts) [10], and Tuple Space Search [28] have lower complexity in worst-case time requirements than linear search schemes. HiCuts partitions the multidimensional search space guided by heuristics of classifier. The partitioning is continued until leaves of the tree include a pre-defined small number of signatures, and linear search is performed for those signatures. Required pre-processing to construct the tree is reasonable. However, other limitations such as storage requirements and scalabilities make them unsatisfactory for fast packet filtering in DDoS problems.

5.2 Discussion

In this section we discuss some practical issues such as using BFC for intrusion detection systems (IDSs) [12] and hardware implementation.

Instead of looking up signatures from IP headers, as it is shown in classifying DDoS packets application, IDSs require signature matching for strings. An IDS collects intrusion signatures and scans the payload of the packets passed by (sometimes the information from different packets needs to be correlated), and then searches for match and sends alarm. For example, Boyer-Moore algorithm is used for string matching in an open source IDS Snort [15]. To test BFC in this application, we define different sets of signature as from 2 to 160 bytes of string. The search time curves of different number of signatures (NoS) and different lengths of signature are shown in figure 11. From the figure we can see the search time is between 19ns to 50ns for all the tests. Therefore, in the worst case, the system still can provide 2.62Gbps throughput. Based on current intrusion detection technology, most of IDSs are only capable of real-time analysis on Fast Ethernet links (100Mbps).

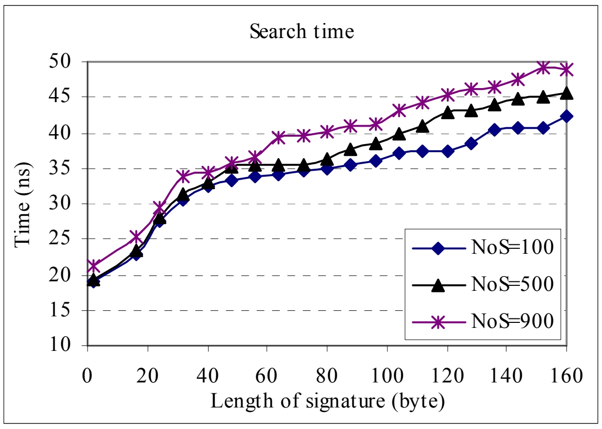


Fig. 11 Search time for IDS application.

When updating of signatures is considered, BFC also offers good performance, since updating signatures in IDS application is an important issue. Table 3 shows the updating time per signature with different signature numbers. To update total 1000 signatures in one of the experiments, it approximately costs 1.5ms, which is a very fast updating speed.

Table 3 Updating time per signature (ns)

Number of signatures	***Average***	***Maximum***
50	0.044	0.14
100	0.094	0.14
200	0.11	0.17
300	0.32	0.42
400	0.54	0.73
500	0.65	0.85
600	0.77	0.97
700	0.88	1.27
800	0.92	1.31
900	1.04	1.35
1000	1.24	1.56

BFC can be further implemented into a hardware device. By using a Field Programmable Gate Array (FPGA) [19], BFC can be realized as a fast hardware-based DDoS filter or a hardware-based IDS. Figure 12 depicts the FPGA implementation of BFC. Because an FPGA chip can usually have 10Mb SRAMs, we can conveniently put all the data structures of BFC on chip. For the IDS application, we need to move the signature base off-chip because it can be very large. However, this design does not affect the searching because in the data structure sum_list there is a pointer leads to the actual signature in the off-chip base.

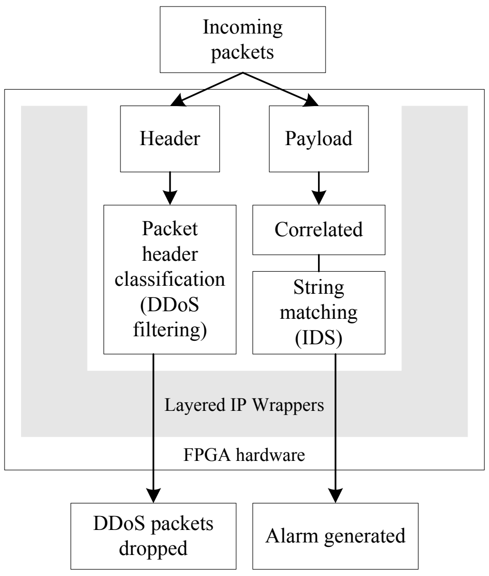


Fig. 12 FPGA implementation of BFC.

6. Conclusion

In this paper we present a Bloom filter-based packet classifier for DDoS packet classification. The proposed scheme shows good performance in terms of search speed, updating time, storage and false positive rate. We also demonstrate a design of FPGA hardware implementation, which can be used in high-speed networks that require gigabit processing speed for DDoS defense systems and intrusion detection systems. It is also possible to apply this scheme into other packet classification problems such as IP lookup for programmable routers.

References

[1] H. Aljifri, "IP Traceback: A New Denial-of-Service Deterrent?" *IEEE Security & Privacy*, vol. 1, no. 3, pp. 24-31, 2003.

[2] B. Bloom, "Space/Time Trade-offs in Hash Coding with Allowable Errors", *Communications of the ACM*, vol. 13, no. 7, pp. 422-426, 1970.

[3] A. Broder and M. Mitzenmacher, "Network Applications of Bloom Filters: A Survey", *Internet Mathematics*, vol. 1, no. 4, pp. 485-509, 2003.

[4] CAIDA, *Cooperative Association for Internet Data Analysis, Skitter Project*, http://www.caida.org/tools/measurement/skitter/, 2005.

[5] R. C. Chen, W. Shi and W. Zhou, *Simulation of Distributed Denial of Service Attacks*, Technical Report, TR C4/09, School of Information Technology, Deakin University, Australia, 2004.

[6] P. Ferguson and D. Senie, *RFC 2267 - Network Ingress Filtering: Defeating Denial of Service Attacks Which Employ IP Source Address Spoofing*, Network Working Group, 1998.

[7] S. Floyd and V. Jacobson, "Random Early Detection Gateways for Congestion Avoidance", *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397-413, 1993.

[8] L. Garber, "Denial-of-Service Attacks Rip the Internet", *IEEE Computer*, vol. 33, no. 4, pp. 12-17, 2000.

[9] P. Gevros, J. Crowcroft, P. Kirstein and S. Bhatti, "Congestion Control Mechanisms and the Best Effort Service Model", *IEEE Network*, vol. 15, no. 3, pp. 16-26, 2001.

[10] P. Gupta and N. McKeown, "Algorithms for Packet Classification", *IEEE Network*, vol. 15, no. 2, pp. 24-32, 2001.

[11] S. Haykin, *Neural Networks: A Comprehensive Foundation, 2nd Edition*, Prentice Hall, 1998.

[12] R. A. Kemmerer and G. Vigna, "Intrusion Detection: A Brief History and Overview", *IEEE Computer*, vol. 35, no. 4, pp. 27-30, 2002.

[13] S. M. Khattab, C. Sangpachatanaruk, R. Melhem, D. Mosse and T. Znati, "Proactive Server Roaming for Mitigating Denial-of-Service Attacks", *1st International Conference on Information Technology: Research and Education*, pp. 286-290, 2003.

[14] K. Park and H. Lee, "On the Effectiveness of Route-based Packet Filtering For Distributed DoS Attack Prevention in Power-law Internet", *ACM SIGCOMM*, pp. 15-26, 2001.

[15] M. Roesch, *Snort Project*, http://www.snort.org, 1998.

[16] V. Srinivasan, G. Varghese, S. Suri and M. Waldvogel, "Fast and Scalable Layer Four Switching", *ACM SIGCOMM*, pp. 191-202, 1998.

[17] SSFNet, *Scalable Simulation Framework*, http://www.ssfnet.org, 2005.

[18] D. E. Taylor, J. W. Lockwood, T. S. Sproull, J. S. Turner and D. Parlour, "Scalable IP Lookup for Programmable Routers", *IEEE INFOCOM*, pp. 562-571, 2002.

[19] S. M. Trimberger, *Field-Programmable Gate Array Technology*, Kluwer Academic Publishers, 1994.

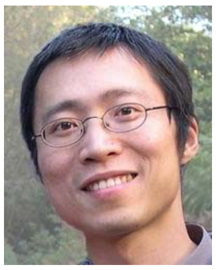
[20] Y. Xiang and W. Zhou, "Mark-Aided Distributed Filtering by Using Neural Network for DDoS Defense", *IEEE GLOBECOM*, pp. 1701-1705, 2005.

[21] Y. Xiang, W. Zhou and M. Chowdhury, *A Survey of Active and Passive Defence Mechanisms against DDoS Attacks*, Technical Report, TR C04/02, School of Information Technology, Deakin University, 2004.

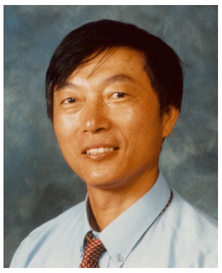
[22] Y. Xiang, W. Zhou and J. Rough, "Trace IP Packets by Flexible Deterministic Packet Marking (FDPM)", *IEEE*

International Workshop on IP Operations & Management, pp. 246-252, 2004.

[23] J. Yan, S. Early and R. Anderson, "The XenoService A Distributed Defeat for Distributed Denial of Service", *The 3rd Information Survivability Workshop (ISW 2000)*, 2000.



Yang Xiang is presently a PhD candidate at School of Engineering and Information Technology, Deakin University, Melbourne, Australia. He received the B.Eng degree from Dalian University of Technology in 1997 and M.Sc degree from the Chinese Academy of Sciences in 2000. Before he came to Deakin University in 2003, he was a software engineer in Mustek Opto-Electronics Inc., Taiwan and West Lake Software, China. His research interests include network security, web services and wireless systems. In particular, he is currently working in a research group developing new Internet security architectures and active defense systems against DDoS attacks. Mr. Xiang published many international journal and conference papers and has been involved many international conferences as reviewer and tutorial presenter. He is a member of IEEE and Australian Computer Society.



Professor **Wanlei Zhou** received the B.Eng and M.Eng degrees from Harbin Institute of Technology, Harbin, China in 1982 and 1984, respectively, and the PhD degree from The Australian National University, Canberra, Australia, in 1991. He is currently the Chair Professor of IT and the Head in School of Engineering and Information Technology, Deakin University, Melbourne, Australia. Before joining Deakin University, Professor Zhou has been a programmer in Apollo/HP at Massachusetts, USA, a Chief Software Engineer in HighTech Computers at Sydney, Australia, a Lecturer in National University of Singapore, Singapore, and a Lecturer in Monash University, Melbourne, Australia. His research interests include theory and practical issues of building distributed systems, Internet computing and security, distributed and heterogeneous databases, mobile computing, performance evaluation, and fault-tolerant computing. Professor Zhou is a member of the IEEE and IEEE Computer Society.

Professor Zhou has published more than 150 papers in refereed international journals and refereed international conferences proceedings. Professor Zhou was the Program Committee Co-Chair of the 2000 IEEE International Conference on Algorithms and Architectures for Parallel Processing (ICA3PP 2000), the Program Committee Co-Chair of ICA3PP 2002, and the Program Committee Co-Chair of The Second International Conference on Web-Based Learning (ICWL2003). Since 1997 Professor Zhou has been involved in more than 50 international conferences as PC Chair, Session Chair, Publication Chair, and PC member.