A High-Speed Network Content Filtering System

Guohong Zhao, Shuhui Chen, Baokang Zhao, Ilsun You, Jinshu Su, Wanrong Yu

A High-Speed Network Content Filtering System

Guohong Zhao[1], Shuhui Chen[1], Baokang Zhao[1], Ilsun You[2], Jinshu Su[1], Wanrong Yu[1]

[1]School of Computer Science, National University of Defense Technology, Changsha, Hunan, CHINA

[1]{ghzhao, csh, bkzhao, sjs, wlyu}@nudt.edu.cn

[2]School of Information Science, Korean Bible University, Seoul, South Korea

[2]{ilsunu@gmail.com}

Abstract. Current software based Content Filtering Systems are too computing intensive in large scale packets payload detection and cannot meet the performance requirements of modern networks. Thus, hardware architectures are desired to speed up the detection process. In this paper, hardware based Conjoint Network Content Filtering System (CNCFS) is proposed to solve the problem. In CNCFS, a TCAM based algorithm named Linking Shared Multi-Match (LSMM) is implemented, which can speed up large scale Multi-Pattern Multi-Matching greatly. Also, this system can also be used in high speed mobile networks which need to deal with the security of fast handover of mobile users. The results of performance evaluation show that our solution can provide 5 Gbps wire speed processing capability.

Keywords: Key words: Network Security, Hardware Accelerating, Content Filtering, Pattern Match

1 Introduction

Today, large number of malicious attack, illegal intrusions, worms and other harmful information are spreading over the Internet. CFS (Content Filtering System) and IDS based on software are used to isolate and monitor these harmful information. However, software based CFS and IDS are essentially computing intensive and can't keep up with the traffic rates requested by most of telecom backbone which employed OC48 or OC192 high-speed links. Moreover, they can't afford to support the high performance

requirements for secure and fast handover in mobile internet networks including Mobile IPv6 (MIPv6) or PMIPv6 [14-15], Thus, new content filtering based on hardware architectures is a promising way to fill up the gap between network traffic rates and NIDS analysis rates.

Content Filtering is a pattern matching process focus on the payload of network packet. There are many security applications which require content pattern matching, such as network intrusion detection and prevention, content filtering, and load balancing. Measurements on Snort IDS show that 80% of total processing time is spent on string matching [4]. Thus, using high-speed algorithms or customized hardware to accelerate the speed of content matching becomes a critical problem.

Currently, network content filtering systems mainly deal with packet reassembly, application recovery, content pattern matching, alarm and event log. Among all, content pattern matching consumes most of the computing resources. The process of pattern matching is as follows: given $P = \{p_0, p_1 \cdots p_k\}$ as a set of patterns, which are the character strings from fixed alphabet $\sum$; given $T = t_1, t_2, \cdots t_N$ as a very large text, whose characters are also from $\sum$, then, the purpose of pattern matching is to find out every p_i in T, where $p_i = t_{k0} t_{k0+1} \cdots t_{k1} (0 \le k0, k1 \le N)$.

Some characteristics of the network content filtering system are listed here:

- The matching speed should be above Gbps.
- The payload of every packet should be matched and there are thousands or more rules, whose length are various.
- Frequently updating of the rules is unnecessary. Adding or deleting single rule can be completed within several seconds.

Since content pattern matching is computing intensive, lots of researches concentrate on how to accelerate the pattern matching. In 1975, Aho [1] proposed the AC Algorithm, which maps the multi-pattern matching process to the state transfer on the state machine. Based on it, many optimized algorithms have been proposed, such as C.J. Coit's AC_BM [3] algorithms and M. Fish's Boyer-Moore-Horspool [4] algorithm. Software based improving algorithms were proposed in [8-9]. In recent years, content filtering turns to use customized hardware to accelerate the pattern matching. TCAM (Ternary Content Addressable Memory) is a component providing tri-state cells of fixed length. Every item (of the TCAM?) contains a bit string and each bit in the string can be 0, 1, or x (do not care). According to the content being searched, TCAM compares this string against all cells of it parallelly, and reports the matched entry.

TCAM have the characteristics of deterministic searching time and deterministic capacity, which make it quite suitable for packet classifying applications. Currently TCAM supports more than 100M times searching in parallel over 288 Bit, or even wider ranges. We can store more than 128K matched patterns in one TCAM.

Fang Yu [6] proposed a method which could store long-pattern segments into TCAM and approach Gigabit matching speed. However, this system needs to maintain a Match Table (MT) and its RAM requirement is too much for a network device. Based on the DIRPE method, Karthik Lakshminarayanan [7] proposed a fence code, which was used to solve the multi-matching problem of fixed area (e.g. five-meta item), and there are different characteristics between content rules and fixed field rules .

The architecture and method proposed in this paper are applicable to IDS or Content Filtering Systems. In this architecture, the software compiles rules and downloads them to the hardware, while the hardware completes the packet stream recovery and pattern match. This paper mainly discusses the following contents:

- A Conjoint Network Content Filtering System (CNCFS) is proposed and implemented.
- Based on CNCFS, a long-pattern hardware matching method using TCAM is proposed and actualized.
- By using little resources, a TCAM multi-matching scheme is implemented which can provide more than 5 Gbps wire speed processing ability.

2 Proposed Approach

In CNCFS model, hardware component (Line Card) does packet reception, packet stream recovery, pattern match and event alarm; software component (Control Card) does rule compiling, loads compiled rule image to hardware, interfaces to Administrator and so on, as illustrated in Figure 1. After packets enter the system from the interfaces, they are recovered to streams before being sent to Matching Engine for pattern matching. During the matching, some event results are sent to software for log records, alarming or composite rule processing on higher levels.

Packet reassembly and flow recovery are very critical in the system. In some special applications, e.g. BBS (telnet), every packet only transmits a byte, while the combination of many bytes in different packets may form illegal information. During packet transmission through the network links, large IP packets may be fragmented due to the various MTUs of different links. To escape detection, illegal information promulgators

often divide the large data into many small packets and transmit them into the network. Thus, the fingerprints are spread into several packets, which make the detecting of those illegal information very difficult for the Matching Engine. The stream recovery module takes the responsibility of preprocessing packets, combining the data from the same flow to form one message. The stream recovery module reassembles the inactive flows in every △t period, or buffer those data in a certain memory space (Memcap) and reassemble them later. Here, the Memcap and △t should be selected carefully.

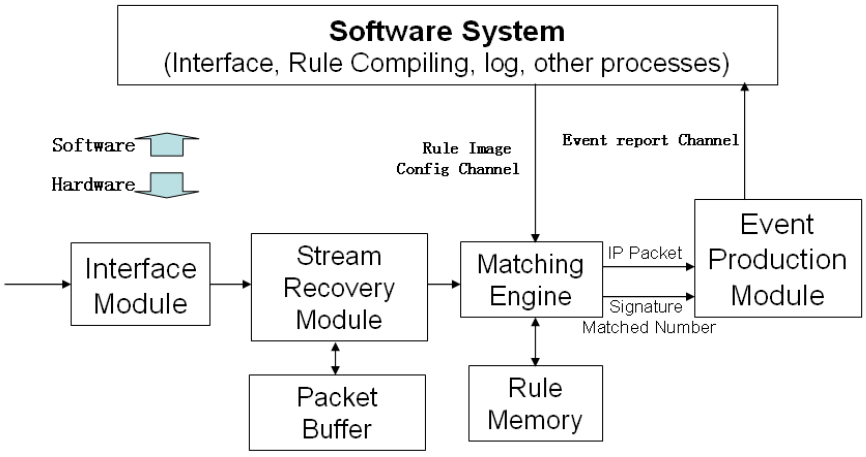

Fig. 1. CNCFS Architecture

The following of this paper mainly expatiates on the matching methods of multi-pattern multi-matching using TCAM.

2.1 TCAM

TCAM is widely used on IP head rules matching, e.g. the longest prefix matching in routing search. Due to its intrinsic ability of parallel searching, it is also used in other high-speed pattern matching cases.

In the field of hardware packets classification, TCAM is one of the most popular methods. Besides 0 and 1, TCAM can store “do not care (x)” state and compare the input keywords with its items in parallel. Given the number of different rules is M, the memory space TCAM requires is only O(M). For a packet of length N, W bytes of the packet are matched in TCAM each time, where W is the width of the TCAM, then shift one byte and check the TCAM again. The search speed TCAM can attain is O(N).

Besides its advantages, TCAM also has the shortcomings of low density and high power consumption, so it should be used efficiently.

TCAM is based on first-match, which just exports the lowest index among all matches of the input string if there are two or more matches. However, content filtering system and IDS are based on multi-match, which means that a packet may match multiple keywords. If TCAM is used for multi-matching of content patterns, we should first solve the long rules (rules that exceed the width of TCAM) and the rules storage sequence problems.

2.2 Rule Length

Content Security System often needs to add or delete some rules, but the proportion of various lengths in the rule set is relatively stable. Figure 2 shows the length distribution of 1070 rules. Here, the content of these rules is in unicode, so their lengths are all even. There is only one longest rule of 18 Unicode UCS-2 characters (36 bytes). The shortest rules have 2 unicode characters. The lengths of most rules are distributed between 6 and 10 bytes, which account for 80% of the rules. If we adopt the length of the longest rule as the configuration length of TCAM, a lot of TCAM space will be used to store x (do not care). Take Figure 2 as an example, if we adopt 36 bytes as the configuration length, the utility ratio of TCAM is only 26.2%. So, in order to save TCAM space, we need to find an effective method to store long rules.

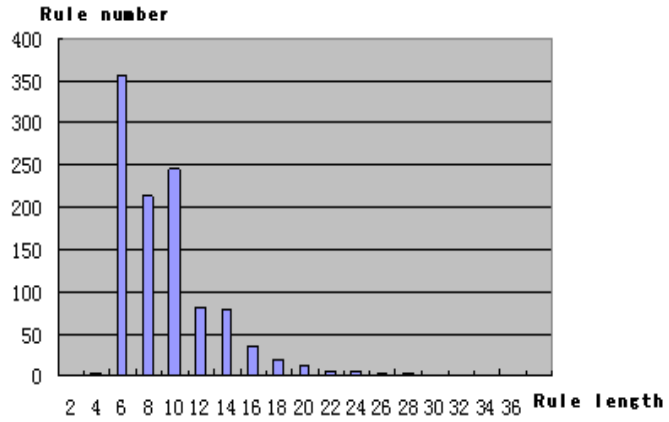

Fig. 2. Rule Length Distribution

2.3 LSMM

We introduce the Linking Shared Multi-Match (LSMM) to solve the storage problem of long rules. The storage strategy of LSMM is as following: every item in TCAM consists of prefix number and segment content. Suppose the length of segments in TCAM is 4, the rule "ABCDEFGHIJ" of length 10 is organized as Figure 3, in which it is divided into three segments: "ABCD", "EFGH" and "IJ", the last two bytes of the last segment are filled by "**" (denoting 16 "do not care" bits). In Figure 3, the leftmost column is the index of TCAM, identified by the addresses. Column 2 stores the address pointer of the preceding segment. Column 3 stores the segment patterns. Column 2 and 3 are stored in TCAM. Column 4 contains matching results which are stored in SRAM.

Index	B6…B	B3…B	Matched
0	1	IJ**	R_i
1	2	EFGH	
2	-1	ABCD	

Fig. 3. Rule Storage Example

If there are n items stored in TCAM, then it is necessary to increase $\log_2 n$ bits for every item to store the addresses of preceding segments (the preceding field). Each time before matching, we add the preceding field in the front of the matched content, then send it to TCAM to do the next comparison. When there is a hit in the preceding segment, we record it in memory. We call the data structure a partial hit list, which records both the position and the index of the hit packets.

Suppose the payload of an input packet is "ZABCDEFGHIJKLMN" and we want to perform content matching using the rules in Figure 3. First, we should add the preceding field "-1" (it is the preceding field value of the first segment). Second, use "-1ZABCDEFGHIJKLMN" to match. If there is no hit, shift one byte so that the string to be matched becomes "-1ABCDEFGHIJKLMN". If there is a hit on the second segment, the partial hit list records this hit. The desired next-hit position of the packet is recorded on the first field, and the current TCAM position of this hit is recorded on the second field. Then, we continue to shift and match "-1BCDEFGHIJKLMN". When reaching the position of the sixth byte, we use "-1EFGHIJKLMN" to match firstly. There is not hit, so we take out address 2 from PHL to constitute "2EFGHIJKLMN",

then there is a hit. According to the process, the last match is 0 segment and the matching rule is Ri (Figure 3 and Figure 4).

If the length all the rules are shorter than or equal the TCAM width, they are stored in TCAM according to their lengths in descending order to implement multi-match. If the length of some rules is greater than the TCAM width, then these rules cannot be simply stored due to the required segment. For example, if the TCAM width is 4, then two rules "ABCDEFGH" and "EFGHXYZW" will share the segment "EFGH". In this case, an effective assignment method should be adopted to solve the share problem and multi-match problem.

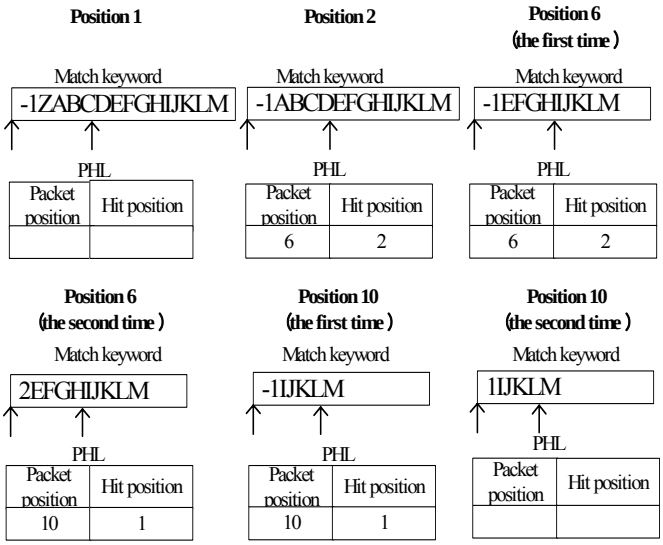

Fig. 4. Long pattern match process example

2.4 Well-ordered TCAM Rule Assignment

The aim of the Well-ordered TCAM Rule Assignment is that, after the rule assignment of a rule set which consists of various rule lengths is stored in TCAM, there shoud be no match missing for any packet and any rule set.

First, if the rules are within the TCAM width, for two rules R_i and R_j, the matching list are M_i and M_j, the storage position of TCAM are P_i and P_j, and TCAM width for storing content is W, the number of items in TCAM is H. Here are four cases:

(1) If $R_i \cap R_j = \varphi$, the sequence of R_i and R_j is not important for their position.

(2) If $R_i \subseteq R_j$, then $P_j < P_i$, and $M_j \supseteq M_i$.

(3) If $R_j \subset R_i$, then $P_i < P_j$, and $M_i \supseteq M_j$.

(4) If $R_i \cap R_j \neq \varphi$ and never meets the 2nd and 3rd conditions, then the sequence of R_i and R_j is not important for their position.

Definition 1: for the general rule $R = \langle Seg_1, Seg_2, \cdots Seg_n \rangle$, R is an ordered set, $|Seg_i| = W$, $1 \le i \le n-1$; $|Seg_n| \le W$, $|R| = \sum_{i=1}^{n} |Seg_i|$, S(R) = n.

Definition 2: for the connection operation "+", $R_i = \langle Seg_{i,1}, Seg_{i,2}, \cdots Seg_{i,n} \rangle$, $R_j = \langle Seg_{j,1}, Seg_{j,2}, \cdots Seg_{j,m} \rangle$, $R_i + R_j = \langle Seg_{i,1}, \cdots, Seg_{i,n}, Seg_{j,1}, \cdots Seg_{j,m} \rangle$.

Definition 3: given two rules $R_k, R_j \in \Sigma^*$, if there is a max length R_i, that $R_i + R_k = R_j$, then we call R_i is the prefix of R_j, recorded as $R_i \prec R_j$.

Definition 4: given two rules $R_k, R_j \in \Sigma^*$, if there is a max length R_i, that $R_i + R_k = R_j$, $|R_i| = MW$, then we call R_i is the ordered prefix of R_j, recorded as $R_i \prec_o R_j$.

Definition 5: If there are R_i, R_j and max-length R_s, and $R_s \prec_o R_j$, $R_s \prec_o R_i$, then R_i and R_j share prefix R_s. We call R_i and R_j sharing max ordered prefix R_s, recorded as MOP(R_i, R_j)= R_s.

There are several relationships between two different rules R_i and R_j:

(1) $R_i \prec_o R_j$, R_j contains R_i and S(R_i)=MW.

(2) $R_j \prec_o R_i$, R_i contains R_j and S(R_j)=MW.

(3) MOP(R_i, R_j)= R_s, R_i and R_j share the max-prefix R_s.

(4) R_i and R_j do not contain each other or do not share the max ordered prefix.

When there are rules which contain others, a string matching the "parent rule" should match the "child rule". In order to ensure the TCAM be well-ordered, the "child rule" must be stored after its "parent rule". If there is a max ordered prefix shared between different rules, then we just store the max-prefix R_s once.

2.5 Algorithm

The algorithm converts the rule set ($RuleSet = \{R_1, R_2 \cdots R_k\}$) to a well-ordered TCAM extended rule set E, and loads the rule into TCAM. Insert (R_i,E) is a process which inserts R_i into E, scans all the rules of E, evaluates the relation between R_i and every rule in E, and then makes some disposition. The algorithm is described as follows:

```
CompileRule{
  E = φ ;
  for all the rule R_i in RuleSet
```

```
    E=Insert( R_i , E);
    Convert PrePoint to Address of Segment and write E into TCAM;
  }
  Insert( R_i , E){
    int   MinPosition = 0;
    int   MaxPosition =H-1;
    int BeginStore=1;
    for all the  R_j  in E{
        if  R_i ≺_o R_j :
          M_{j,s(R_j)} = M_{j,s(R_j)} ∪ R_i ;
          return;
        if  R_j ≺_o R_i :
          M_{i,S(R_j)} = M_{i,S(R_j)} ∪ R_j ;
          Delete( R_j );
        if   MOP( R_i , R_j )= R_s , R_s ≠ R_i , R_s ≠ R_j
          BeginStore=|R_s|/W +1
          continue;
    }
    for all the  R_j  in E{
      if( R_j ⊂ R_i )    MaxPosition = Position of  Seg_{j,S(R_j)} + 1;
     if( R_i ⊂ R_j )    MinPosition = Position of  Seg_{j,1} - 1;
    }
     if BeginStore<=n
       Insert  Seg_{i,n} … Seg_{i,BeginStore} in any place from MinPosition to MaxPosition;
  }
```

Now we explain the algorithm through an example. Supposing we have five rules, which are R_1 = "ABCDEFGHIJKLM", R_2 = "ABCDEFGH", R_3 = "ABCDWXYZ", R_4 = "AB", and R_5 = "EFGH". If the TCAM width is 4, then the storage of these rules is demonstrated in Figure 5.

0	4	WXYZ	R_3
1	2	M***	R_1
2	3	IJKL	
3	4	EFGH	R_2 , R_5

4	-1	ABCD	R_4
5	-1	AB**	R_4
6	-1	EFGH	R_5

Fig. 5. Example of rule assignment

Following is a list of the packet matching algorithm:

```
Match(Packet){
  for CurPositsition from 1 to packetlength{
    for all item in PHL{
      if item.PackPosition=CurPosition
        PHLPop();
        MatchingCont=item.PackPosition+*Packet[CurPosition];
        if(index=MatchTCAM() is valid){
            output(SRAM[index]);
            if(not TCAM[index].LastSeg) PHLPush(CurPostion, index);
        }
     }
    MatchingCont=”-1”+*Packet[CurPosition];
   index=MatchTCAM();
    if(index is valid){
        output(SRAM[index]);
    if(not TCAM[index].LastSeg) Pop(CurPostion, index);
  }
}
```

An example using the algorithm is given in Section 2.3.

3 Analysis and Implementation

3.1 Implementation

The deployment of our system is plotted in Figure1. The control board uses Intel Pentium M 1.6G CPU with 512M memory. The OS is RedHat Linux 9.0. The TCAM of the line card uses Cypress CYN70256. There are two OC48 ports on the line card. One

Compact PCI bus is used to connect line card and control board as Control Channel and Event Transfer Channel. We use two Matching Engines to process in parallel.

Packets enter the system through AMCC S4803, and then are sent to the Stream Recovery Module for packet assembling. The stream recovery process adopts the idea of [10]. After that, packets go to the Matching Engine for content matching. If there are some hits, then software will deal with higher-level matching using more complicated rules, such as probability, accumulation, and group, etc.

Section 3.2 indicates that, given a rule set, there is a minimum limit for the capacity of the TCAM. Because there is a determinate configuration limit on the length of TCAM, we should compromise between the configurable TCAM width and the performance requirement [11].

When the length of a content rule is longer than 1 byte, it is necessary to match TCAM many times. Multiple matching is implemented though shifting bytes as following. If the packet length is M bytes, we should shift M-1 times in all. Before each shift, there is a match process for the items of TCAM. We can use S-Double technology to accelerate the shifting step. We shift one byte rightward for each segment of all rules, and then restore it. In order to speed up the matching, every segment is stored twice in the TCAM. After each TCAM matching, we can shift the content rightward two bytes before starting new matching. Thus, by trading space for time, we accelerate the speed of TCAM matching [12-13].

When we use the S-Double technology, our system can automatically decide whether to adopt S-Double or not, according to the number of rules. If the storage space used by the rules decreases to half of the TCAM space, the system adopts this technology. On the contrary, if the space increases to half of the TCAM space, S-Double will be shut down and the duplicated stored rules will be deleted. In order to avoid fluctuation during the process of adding or deleting rules, after many simulations, we select 2 proper threshold-values: waterline_nosd and waterline_sd. When the number of the whole segment falls to waterline_sd, we adopt the S-Double technology; when it grows up to waterline_nosd, we cancel it. This technology can effectively avoid fluctuation during the process of adding or deleting rules and achieve the optimal speed according to different states of the system.

Furthermore, the S-Double technology could be used repetitively. We can store rules 3 times of the original rules number and shift 3 bytes after each match, or store rules 4 times of the original rules number and shift 4 bytes every match, etc.

3.2 Memory Space Analysis

In a rule set of n rules, the length of every rule is L_i, 0≤i≤N-1, the segment number of the rules is SegNum, SegNum= $\sum_{i=0}^{N-1}(L_i + W - 1)/W$, the TCAM bits required to be extended for every segment are $W_{ext} = \lceil Log_2(SegNum) \rceil$, and the total TCAM space is $(W_{cont} + W_{ext}) * SegNum$. As mentioned in section 2.2, the parameter W is configurable and smaller W means higher utility ratio of the system. The smaller W is, the larger the segment number will be, the more bits SegNum should be extend. The larger W is, the smaller the segment number will be and fewer bits SegNum should be extend.

Regarding to the rule lengths in Figure 2, we compute corresponding TCAM space size needed. We get the relationship between different segment length and different TCAM space sizes, as shown in Figure 6. Using LSMM, for all those rule lengths, averagely 6% accessional TCAM space is added to solve the long rule problem.

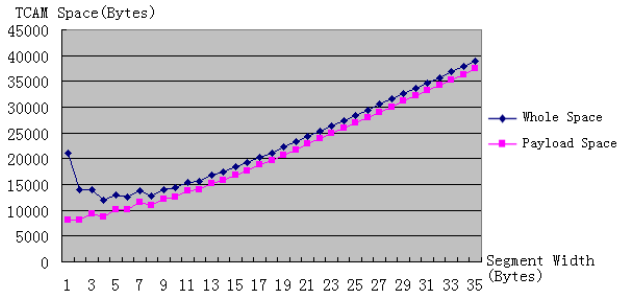

Fig. 6. Relationship between Segment Length and TCAM space size

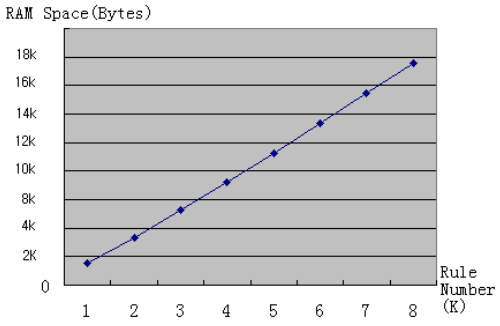

Fig. 7. Relationship between rule number and RAM Space of LSMM

The RAM space size of LSMM can be depicted by the following expression: $\lceil Log_2(N) \rceil * SegNum \sum((|R_i|+1)/W)$. When the distribution of rule length is steady, the size of the RAM space is almost linear with the rule number. Generally speaking, TCAM width is 36, 72, 144 and 288 bits [2], etc. The frequency of TCAM is low if its width is too small or too large. In our system, we consider the typical case (72 bits width, 8 bytes segment width). Figure 7 shows the changing of the space size along with different rule numbers.

3.3 Performance Analysis

After accessing TCAM, if there is a hit, then accessing SRAM is necessary and the access process can be streamlined. The system performance is limited by the access time of TCAM or SRAM. Because at some positions of some packets, if the PHL is not empty and the desired next-hit position of some items equal the current position, we need to access TCAM many times. Since the packet streamlining is limited, so FIFO is needed to smooth the burst.

Considering random packet contents, for any w bytes in the packet, there are 2^{8w} possible values, so the chance of matching one particular pattern is $1/2^{8w}$. There are $\sum_i S(R_i)$ segments (TCAM items) to be matched. So counting in the address pointer of the preceding segment, the hit probability of one pattern is $\sum S(R_i)/2^{8w+\lceil \log_2 \sum S(R_i) \rceil}$, which decreases dramatically when w increases. For instance, supposing we have 1000 rules (each rule is 64bytes) and the segment width is 8 bytes (which is the typical width with best TCAM performance), the hit rate should be 5.29e-20. The average PHL size is too difficult to compute, theoretically speaking it should be less than $w * \sum S(R_i)/2^{8w+\lceil \log_2 \sum S(R_i) \rceil}$. Since the hit of last segments of all rules do not increase the PHL size, so the average PHL size should be less than 5.29e-20.

The size of the FIFO packet queue is determined by the maximum PHL, while not the average PHL. Through capturing packets on an OC48 interface for 60 seconds (the real-time throughput was about 1.8Gbps, all packets whose sizes were less than 62 bytes were ignored, because they did not match our rules. The total memory space of the packets in the 60 seconds time was about 14.4G Bytes. Using the rules mentioned in section 2.2, registers in Matching Engine showed that 9 rules were matched. The max

size of PHL was 3, because the rules we used contained segment inclusive relationships as described in section 2.4. If we modify the rules, the max PHL would be even less.

Table 1 lists the work process during the matching process using LSMM.

Table 1. Match Process

		LSMM
Before matching		Clear out-of-date items of Partial Hitting List, add the preceding field of matched content
Matching process		Send the keywords to be matched into TCAM, two processes is the same.
After matching	Hit	Get hit index and write Partial Hit List Shift bytes
	No Hit	Shift bytes.

LSMM does not induce obvious additional time cost and FPGA logics is comparable to the pure TCAM matching. The addition of preceding field before matching is parallel to the TCAM searching process. If PHL is too long, system may be bottlenecked. In our case, the operating frequency of TCAM is 100MHz and the width of TCAM is 72 bits. For Cypress CYN70256 [2], two time cycles are needed to complete one match. In the real network testing, without using the S-Double technology, CNCFS can obtain throughput about 800Mbps (we had two matching engines work simultaneously, each one achieves 400Mbps throughput), However, after using S-Double 8 times, the throughput of CNCFS can be improved to 6.4 Gbps.

We also test our system using the SPRIENT AX4000, with the following setting of traffic flows to simulate real network traffic:

64 bytes 50% of whole traffic

128 bytes 5% of whole traffic

256 bytes 5% of whole traffic

512 bytes 10% of whole traffic

1024 bytes 10% of whole traffic

1600 bytes 10% of whole traffic

2 OC48 ports of the AX4000 were connected to our system. The throughput ratio is set as 100%, which means the whole traffic is about 5Gbps. In this test, our system detects all packets correctly without any loss. In fact, because the packets headers do not enter the matching engine, the actual payload traffic was less than 5Gbps.

Each time the LSMM updates a rule (delete or add), it needs to reorganize the entries of TCAM. We utilize a batching process to load more than 1070 rules. This process takes about 2 seconds for the loading of rules and less than 1.5 seconds for the writing of TCAM, which is acceptable for real applications.

4 Conclusions

In this paper, a novel content filtering architecture (CNCFS) using LSMM in TCAM is proposed to meet the requirements of the high performance processing for the rapid increasing network bandwidth. Our NCFS implementation combines the software and hardware to complete the process of content filtering. The performance evaluation shows that our system can improve the network throughput up to 5Gbps.

This paper mainly addresses the following three issues:

- A new network content filtering system framework is proposed and implemented.
- The content rule matching is implemented using TCAM, which solves the problem of long rules storage.
- Without changing the structure of TCAM, a new solution is proposed to solve the multi-matching problem by using fewer storage resources, which can improve the matching speed up to 5 Gbps.

5 Acknowledgment

The work described in this paper is partially supported by the project of National Science Foundation of China under grant No. 61202488, 61272482; the National High Technology Research and Development Program of China (863 Program) No. 2012AA01A506, 2013AA013505. This research was also in part supported by Basic Science Research Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning(2014R1A1A1005915).

References

1. A. V. Aho and M. J. Corasick, Efficient string matching: An aid to bibliographic search. Communications of ACM, June 1975, 18(6): 333 - 334.
2. Cypress Semiconductor Corp. Content addressable memory. http://www.cypress.com/.

3. C.J. Coit, S. Staniford, and J. McAlerney, “Towards Faster String Matching for Intrusion Detection or Exceeding the Speed of Snort,” DARPA Information Survivability Conference and Exposition (DISCEX II'01), 2001.
4. M. Fish and G. Varghese, “Fast Content-Based Packet Handling for Intrusion Detection” UCSD technical report CS2001-0670, 2001.
5. S.Antonatos, K.G.Anagnostakis, and E.P. Markatos. Generating realistic workloads for network intrusion detection systems. In Proc. of the 2004 ACM Workshop on Software and Performance.
6. Fang Yu, Randy H.Katz, and T. V. Lakshman. Gigabit Rate Packet Pattern-Matching Using TCAM, In Proc. of the 12th IEEE International Conference on Network Protocols (ICNP'04).
7. K. Lakshminarayanan, A. Rangarajan, and S. Venkatachary, Algorithms for Advanced Packet Classification with Ternary CAMs. In SIGCOMM’05, August 21-26, 2005.
8. G. Peng, Z. Deyun, S. Qindong, Z. Yahui, and L. Wuchun, Multi-Pattern Approximate Matching Algorithm of Network Information Audit System, Journal of Software, 2004, 15(7) :1074 - 1080.
9. S. Hua and D. Yiqi, A New Fast String Matching Algorithm for Content Filtering and Detection, Computer research and development, 2004, 41(6) :940-945.
10. Z. M. Zhong, G. J., and Ding Wei, Study of network flow timeout strategy, JOURNAL ON COMMUNICATIONS, 2005, 26(4):88-93.
11. L. Caviglione, A. Merlo, and M. Migliardi, Green-Aware Security: Towards a new Research Field, the International Journal of Information Assurance and Security (JIAS), Vol. 7, 2012, issue 5, pp. 338-346.
12. M. Migliardi and A. Merlo, Energy Consumption Simulation of Different Distributed Intrusion Detection Approaches, In Proc. of the 27th IEEE International Conference on Advanced Information Networking and Applications (AINA’13), Barcelona, Spain, March 2013.
13. M. Migliardi and A. Merlo, Improving Energy Efficiency in Distributed Intrusion Detection Systems, Journal of High Speed Networks, IoS Press, 19(3): 251-264, 2013
14. B Kim, J Yang, and I. You, A survey of NETLMM in all-IP-based wireless networks, In Proc. of ACM Mobility 2008, I-Lan, Taiwan, September 2008
15. I. You, J.-H. Lee, and K. Sakurai, DSSH: Digital signature based secure handover for network-based mobility management. Computer Systems: Science & Engineering. 27(3), May 2012