This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2005 proceedings.

A Multi-gigabit Rate Deep Packet Inspection Algorithm using TCAM

Jung-Sik Sung*, Seok-Min Kang†, Youngseok Lee†, Taeck-Geun Kwon†, and Bong-Tae Kim*

* ETRI,
161 Gajeong-dong, Yuseong-gu, Daejeon, 305-350, Korea
{jssung, bkim}@etri.re.kr

† Chungnam National University
220 Gung-dong, Yuseong-gu, Daejeon, 305-764, Korea
{esemkang, lee, tgkwon}@cnu.ac.kr

***Abstract*- With the increasing importance of network protection from cyber threats, it is requested to develop a multi-gigabit rate pattern-matching method for protecting against malicious attacks in high-speed network. This paper devises a high-speed deep packet inspection algorithm with TCAM by using an *m*-byte jumping window pattern-matching scheme. The proposed algorithm significantly reduces the number of TCAM lookups per payload by *m* times with the marginally enlarged TCAM size which can be implemented by cascading multiple TCAMs. Due to the reduced number of TCAM lookups, we can easily achieve multi-gigabit rate for scanning the packet payload. It is shown by simulation that for the Snort rule with 2,247 patterns, our proposed algorithm supports more than 10 Gbps rate with a 9 Mbit TCAM.**

I. INTRODUCTION

Recently, a lot of various cyber threats such as worms, viruses, spam-mails, and hacking have appeared with the explosive increase of Internet usage. When viruses hidden in the e-mail propagate quickly themselves, they will cause various kinds of troubles to users, hosts, and networks. A new type of dangerous threats to security today is the worm. For instance, the SQL Slammer, one of the first flash worms unleashed on the Internet in January 2003, caused significant disruption to networks around the world [1,2]. Hence, the importance of developing network security technologies and various secure solutions that protect data, system, and networks from these cyber attacks has become invaluable. Network Intrusion Detection Systems (NIDSs) monitor every packet in the network to detect malicious attacks. In a high-speed network, an NIDS may be overloaded as the packet arrival rate becomes high. Hence, the hardware-based approach of implementing the NIDS will be appropriate in order to support the high-speed network. The functions of the NIDS are often performed by secure routers in these days. The secure router usually supports multi-gigabit rate such as 10 Gbps Ethernet and OC-192. Therefore, multi-gigabit rate secure routers need 10Gbps scan rate for detecting malicious signatures from the packets.

In this paper, we suggest a 10Gbps deep packet inspection algorithm that can be used with Ternary Contents Addressable Memory (TCAM). Usually, the search capability of TCAM outperforms general purpose memories. TCAM can provide an answer for searching a packet of length *n*, in a deterministic time of *O(n)* TCAM lookups, because one TCAM lookup is needed for every byte position in the packet [3,4]. Since the TCAM lookup time is known and fixed, we need to minimize the number of TCAM lookups per packet to support the multi-gigabit rate secure router. We suppose a TCAM-based pattern- matching algorithm which decreases the number of TCAM lookups per packet by means of performing a TCAM lookup operation per multi-bytes in a packet payload. We call this algorithm as *the jumping window pattern-matching scheme*. One pattern will generate multiple TCAM entries shifted from 0 to *m*–1, if the size of the jumping window is *m*. In our algorithm, TCAM lookups for searching a packet of length *n*, is *O(n/m)*. Therefore, by reducing the number of TCAM lookups with the proposed jumping window scheme, multi-gigabit rate can be supported for secure routers.

The remainder of the paper is organized as follows. We review related work in Section 2, and summarize the characteristics of TCAM in Section 3. Section 4 presents algorithms to map the multiple patterns into TCAM and efficiently scan packets at high speeds. Section 5 describes the analysis and the simulation results of our algorithm. Finally, we conclude the paper in Section 6.

II. RELATED WORK

Signature-based intrusion detection schemes are used to detect malicious signatures which may appear anywhere in the packet payload by scanning the packet payload. These signatures are stored in the searching table as multiple patterns. References [5] and [6] provide fast algorithms to search multiple patterns. Reference [5] suggests a multiple-pattern search algorithm, a set-wise Boyer-Moore-Horspool (SBMH) algorithm, which combines the one-pass approach of Aho-Corasick [7] with the skipping feature of Boyer-Moore [8] as optimized for the average case by Horspool [9]. Reference [6] suggests a fast algorithm for multi-pattern searching. It builds three tables at the preprocessing stage: a SHIFT table, a HASH table, and a PREFIX table. The SHIFT table is similar, but not exactly the same, to the regular shift table in the Boyer-Moore type algorithm. It is used to determine how many characters in the text can be shifted (skipped) when the text is scanned. The HASH and PREFIX tables are used when the shift value is 0. Besides [5] and [6], [10] and [11] provide multiple pattern-matching algorithms for the payload-sensitive packet-filtering system. The multiple pattern-matching algorithms [5][6][10][11] use software approaches. However, software-based pattern-matching is not able to inspect all packets in the high-speed network [12].

° This research was supported in part by ITRC program of the Ministry of Information and Communication, Korea.

Reference [13] proposed a hardware-based technique using parallel bloom filters which could detect strings in streaming data. The proposed scheme builds a bloom filter for each possible pattern length. Each bloom filter scans the streaming data and checks the strings of corresponing length. This could impose parallelism limits in some virus databases because pattern lengths vary from tens to thousands of bytes and there are hundreds of possible patterns lengths[4]. Parallel bloom filters are implemented on FPGA which cannot realize large-scale rule database.

References [3] and [4] provide TCAM-based pattern match algorithms that can be used with TCAM. Reference [3] achieves optimal functionality and efficiency for deep packet filtering with assistance of the "self-study" table which is a preprocessing method similar to cache. Reference [3] supports indefinite length pattern match, because payload fields may be indefinite within the limitation of TCAM width. The system has to match data contents with a sliding window. The width of the sliding window is determined by the width of the field in a rule database, thus it changes dynamically. Since the location of pattern within the payload is not known, the sliding window must forward one byte per clock exactly so as not to miss any matching opportunity. Reference [4] presents a TCAM-based pattern-matching algorithm for handling both short patterns less than the TCAM width and long patterns. In case of the short pattern, one TCAM lookup is needed for every byte position in the payload like [3]. Given the limited TCAM width, a pattern longer than the TCAM width will split into several sub-patterns: the first TCAM width prefix patterns and the remaining suffix patterns. Assuming TCAM lookup time is 4ns, [4] can support a deterministic scan rate of 8bits/4ns = 2Gbps in case of short patterns. For long patterns, the speed of pattern-matching will be dominated by TCAM lookup time if the TCAM hit rate is low and the size of partial hit list table is small. In this case, the total time to scan an n-byte packet is $4n$ ns and the matching speed is $8n$ /$4n$ = 2Gbps [4]. Since 2Gbps will not be suitable for multi-gigabit secure routers or IDSs in the high-speed network, the capability of supporting multi-gigabit scan rate will be essential. Hence, we propose a high-speed deep packet inspection algorithm that employs one TCAM lookup per multi-bytes of payload by using *the variable size jumping window* instead of 'the sliding window scheme'[1]. Fig. 1 illustrates how to find the pattern "GATT" in the payload; the example shows the 6-th segment of the packet payload matches the TCAM entry.

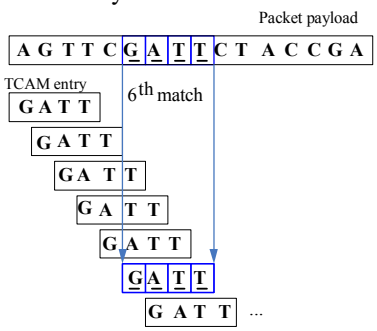

Fig. 1. TCAM-based packet inspection with "sliding window"

[1] In this paper, the scheme in [3][4] is referred to a 'sliding window'.

III. TCAM CHARACTERISTICS

TCAM is a type of memory that can search multiple items simultaneously at a high-speed rate. Each cell in a TCAM may have one of three states (0, 1, or 'don't care'); a binary CAM has only two states (0 or 1). One input may be to search multiple TCAM entries because of the 'don't care' state. If multiple matches are given, TCAM will return the index of the first hit or the indices of multiple hits. Two storage locations, data (or key) and mask cells are necessary to save three possible states. The mask cell specifies which bits in the entry are active, thereby specifying the variable-length prefix. TCAM entries are the logical records stored in the data and mask cells. The TCAM entry may have associated data stored in SRAM attached to the network search machine (NSE) or in a separately accessible external memory (e.g., SRAM or DRAM). The associated data may be returned as the result of a TCAM lookup operation.

A TCAM-based NSE chip can support the packet-lookup speed of 100 ~ 250 million searches per second (MSPS) according to its family. It means that the TCAM lookup time is about 4 ~ 10 ns. One feature of the NSE chip is to support variable word-width searches of 36-, 72-, 144-, and 288-bit wide words called the TCAM width. A 9Mbit TCAM has 72-bit wide 128K entries or 144-bit wide 64K entries. Each of NSE devices may be cascaded to extend TCAM spaces. For example, IDT75K62134 [14] supports up to eight devices through point-to-point cascading. Point-to-point cascading allows databases to contain up to 1 million 72-bit entries.

IV. THE PROPOSED ALGORITHM

A. Algorithm

As stated above, the sliding window scheme supports one TCAM lookup by one shift; this increases the number of TCAM lookups. In order to achieve parallelism for payload inspection, we devise an 'm-byte jumping window' scheme that matches m-byte payload segment with every m-th jumping windows as illustrated in Fig. 2. We guess that the scanning time of the jumping window is superior to that of sliding window. We describe the creation of TCAM entries from a pattern in order to perform the jumping-window scheme as follows.

Let w be the TCAM width and let - be 'don't care' state of TCAM. Given the pattern of "GATT" the position of the pattern in the payload is one of "GATT," "-GATT," "--

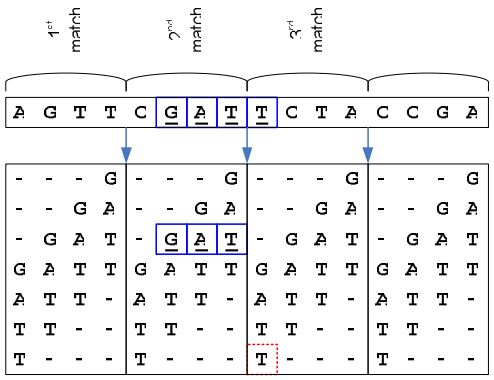

Fig. 2. Example of jumping window scheme (m = 4)

GATT," …, "(*w* -1)-GATT." When *w* is 4, the pattern may be found at the different position of the payload such as "GATT," "-GATT," "--GATT", or "---GATT." (shown in Fig. 3) We put the above derived patterns into the TCAM table. Then, a TCAM lookup operation is carried out for every segment of *w* bytes called a jumping window for a packet payload. Usually, the width of the TCAM, which will be used for matching the pattern in a parallel way, is fixed. Therefore, if the TCAM width is smaller than the pattern, we have to split a long pattern into shorter sub-patterns with the same length of the TCAM width.

If one pattern splits into several sub-patterns, a pattern-matching operation will be completed when all sub-patterns are matched to the TCAM entries in series. Hence, for the matching operation of multiple sub-patterns, a sub-pattern matching function requires the result of the previous sub-pattern matching operation. To increase the speed of searching, we employ a hash function to find the result of the previous sub-pattern matching operation. The value of the hashing function will be stored in the associated data. The hash value of associated data is used as a key for TCAM lookup of the next sub-pattern. The key for TCAM lookup is the combination of the previous hash value and the sub-pattern. The TCAM window, *m*, is the length of sub-pattern. In other words, we compute *m* by subtracting the length of the hash value from *w*. Fig. 4 shows continuity of sub-patterns with hash. A pattern-matching operation should be done, because all sub-patterns are matched to TCAM entries in series when the last hash value of associated data is zero. We set the hash value of key for TCAM lookup of the first sub-pattern to zero. For example, <hash0> in Fig. 4 is zero, because it is the hash value of first sub-pattern. <hash1> of associated data is obtained by hashing with the sub-pattern, "-GAT". It is used as a key for TCAM lookup of the next sub-pattern, "T---." Therefore, the key of the sub-pattern "T---" following "-GAT" is combination of the previous hash value, <hash1>, and the sub-pattern, "T---."

Our algorithm consists of two phases of a preprocessing stage and a scanning stage. At the first stage, the set of patterns are preprocessed to create TCAM entries. Then, at the second stage, payloads are scanned to detect malicious attacks.

B. The Preprocessing Stage

We create TCAM entries from the set of patterns in the preprocessing stage. The algorithm uses symbol notation of Table 1 in the preprocessing stage. The algorithm picks out the pattern P_i from the signature rule to create TCAM entries in order to match the TCAM in parallel. Algorithm 1 shows the algorithm of TCAM entry creation from P_i.

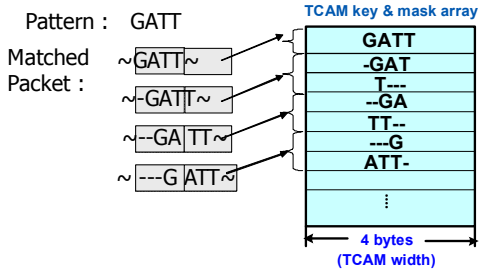

Fig. 3. TCAM entry creation through (0 ~ *w*-1) shift-right operations

Key & Mask		Associated data
Hash	Sub_pattern	hash
<hash0>	GATT	0
<hash0>	– GAT	<hash1>=hash(<hash0>, "GAT")
<hash1>	T – – –	0
<hash0>	– – GA	<hash2>=hash(<hash0>, "GA")
<hash2>	TT– –	0
<hash0>	– – – G	<hash3>=hash(<hash0>, "G")
<hash3>	ATT–	0
…	…	…

Fig. 4. Continuity of sub patterns with hash

TABLE 1. Notations used in the preprocessing stage

w	TCAM width
m	Window size
H_0	Initial hash value, 0
P_i	i-right shifted pattern ($0 \bullet i \bullet m-1$)
S_j	Sub-patterns which are derived from P_j, the size of S_j is equal to m ($j \bullet 1$)
H_j	Hashing with concatenation of H_{j-1} S_j hash($H_{j-1}S_j$)
K_j	key which is derived from S_j (= $H_{j-1}S_j$)

Algorithm 1. TCAM entry creation algorithm for a pattern

```
for each pattern
  for each i-shifted pattern P_i
    if(length (P_i) <= m) {
      S_1 = P_i(m - length (P_i)) ;
      create key H_0S_1 and associated hash 0; }
    else {
      left_length = length (P_j), j = 1;
      while(left_length > m) {
        fetch m-byte S_j ;
        calculate hash H_j = hash(H_j-1, S_k);
        create key H_j-1S_j and associated hash H_j ;
        left_length = left_length – m; j ++;   }
      calculate hash H_j = hash(H_j-1S_j) ;
      create key H_j-1S_j and associated hash 0
```

C. The Scanning Stage

We now describe the scanning stage in more detail and give algorithm for it. The algorithm uses the symbol notation of Table 2. The *m*-byte segment of an input packet payload is denoted as *T[i..i+m–1]* (*i* • 1). A key of TCAM lookup for the payload, *T[i..i+m–1]*, is the combination of the previous hash value and *T[i..i+m–1]*. The previous hash value is hashed with *T [0..i–1]* or the initial hash value if there is no previous sub-pattern matching. When *T[i..i+m–1]* matches to one of TCAM entries, we combine the associated hash value of *T[i..i+m–1]* with the next sub-pattern, *T[i+m..i+2m–1*, in order to create a key for the TCAM lookup of the payload, *T[i+m..i+2m–1]*. If *T[i+(j–1)m..i+jm–1]* matches to one of TCAM entries and the associated hash value is zero, finding the pattern will be completed at the position of a packet payload, *T[i..i+jm–1]*.

Let us illustrate an example for explaining the jumping-window pattern-matching scheme. We can create shifted sub-patterns from a pattern, "GATT" and put them into the TCAM table as shown in Fig. 5. When *m* is 4, we fetch a 4-byte character with 4-byte jumping window from a packet pa-

TABLE 2. Notations used in scanning stage

T	The total payload of a packet
T[i..j]	A continuous byte segments belong to a single packet
n	Length of T
H[]	Hash list
NH[]	Next hash list
assoc_hash	Hash from associated data when TCAM lookup is successful

Algorithm 2. Scanning algorithm

```
left_length = n; i = 1;
H[0] = 0; NH[0] = 0;
while(left_length > 0) {
    fetch m-byte T[i..i+m-1];
    j = 0; k=1;
    while(H[j] != NULL) {
        lookup_tcam(H[j], T[i..i+m-1]);
        if(match) {
            if(associ_hash == 0)
                pattern-matching successful
            else        /* need continuous match */
                NH[k++] = associ_hash ;
        }
        j++ ;
    }
    copy(H, NH); reset(NH);
    i = i+m; left_length = left_length – m;
}
```

yload. Next, we make a key for TCAM lookup concatenating the initial hash value, H_0, and fetched characters. The associated hash value H_1, which is the result of hashing H_0 and shifted sub-pattern ,"-GAT," is returned, when the key created from the third 4-byte jumping window, "CGAT" matches to one of TCAM entries. Now, we form the key with H_1 and the next 4-byte jumping window, "TCTA," in order to search the "-GATTCTA" string from the TCAM. It matches to the TCAM entry which is the last sub-pattern of "-GATT---." Therefore, we obtain the zero value as the result of TCAM lookup, which means that the input packet payload includes the pattern, "GATT."

V. Performace Analysis And Simulation

A. Analysis of TCAM Lookup Time

Let *t* be the TCAM lookup time, *L* be the average length of payloads, and *l* be the length of pattern. The scanning time of the sliding window scheme is equal to the product of the number of TCAM lookups and the TCAM lookup time, i.e., *Lt*. Our algorithm uses the key of TCAM lookup as the combination of the hash value and *m*-byte jumping window from the payload. Namely, it processes *m* characters of the payload per one TCAM lookup operation. Therefore, the scanning time can be derived as *Lt/m*.

It is assumed that the payload length of an IP packet, *L* varies from zero to 1,460 bytes. In addition, the TCAM lookup time, *t*, is assumed to be a constant. Then, *m*-byte jumping window will decide the scanning time. If the value of *m* is large enough, the scanning time will be decreased to provide multi-gigabit payload inspection. However, the number of TCAM entries will be increased. The number of TCAM entries, namely *e*, created from one pattern is given as follow:

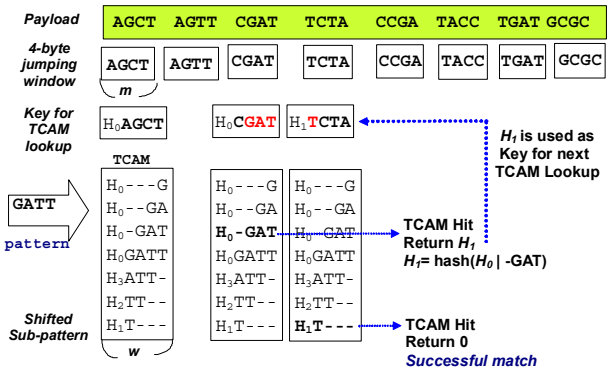

Fig. 5. Jumping-window with hash values

$$e = m\,(l / m + 1) + (l \,\%\, m - 1)$$

Fig. 6 compares the throughput of the proposed algorithm with the sliding window scheme when the length of a packet is 1518 bytes. The maximum-length packet is assumed because it requires the worst case of TCAM lookups for scanning the payload. 0.81 million packets per second (Mpps) throughput is required to support 10Gbps. The sliding window scheme performs 0.17Mpps throughput, but it cannot achieve the performance of 10Gbps. Its throughput does not change as *m* increase. In contrast, our algorithm, the jumping window scheme can support the throughput of 10Gbps. The throughput in our algorithm will be increased as *m* becomes high because the scanning time can be decreased. Fig. 7 compares the number of TCAM lookups of the proposed algorithm with sliding window scheme when *m* is 7. The number of TCAM lookups of sliding window increases rapidly as the packet length increases, while that of our algorithm increases slowly.

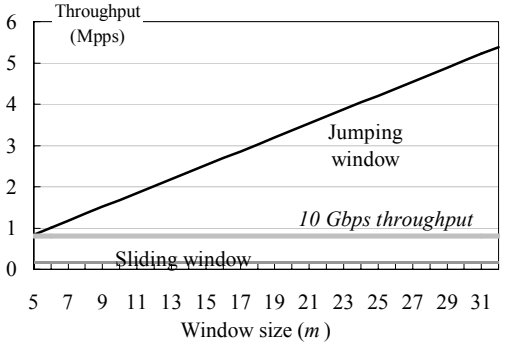

Fig. 6. Packet inspection throughput of jumping and sliding windows

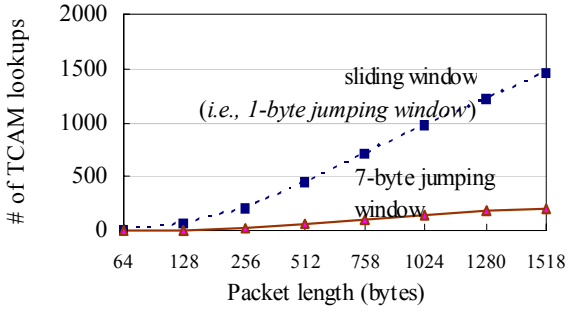

Fig. 7. TCAM accesses of jumping and sliding windows

B. Simulation Results

We considered signature sets from Snort 2.1.0 [15] and created TCAM entries from them. The signature sets have 2,394 rules and 2,247 patterns. We used real packet traces with libpcap [16]. The number of total packets is 10,000 and the average packet payload length is 690 bytes. Fig. 8 shows the number of TCAM lookups and TCAM sizes accommodate all the patterns under different *m*-byte jumping window settings. We observed about 690/*m* TCAM lookups per one packet in Fig. 8, because the average length of packet payload is 690 bytes. As *m* increases, the number of TCAM lookups has been reduced, because one TCAM lookup has been performed for each *m*–byte jumping window. However, as *m* increases, the number of TCAM entries becomes high because of creation of staggered sub-patterns from a pattern. Since TCAM supports several TCAM width for example 36-, 72-, 144-, and 288-bit wide words, TCAM sizes increase rapidly when the size of jumping window is 8 and 17. Our algorithm can achieve maximum performance with 16-byte jumping window where TCAM size is 9Mbit.

We measured the number of TCAM lookups of sliding window and 7-byte jumping window for each real packet as shown in Fig. 9. The number of TCAM lookups of sliding window for each packet is shown in Fig. 7.

VI. CONCLUSION

In this study, we have presented a multi-gigabit pattern-matching algorithm for the high-speed network. The TCAM-based deep packet inspection algorithm developed in this pap er uses a jumping window scheme, which is supported by staggered sub-patterns and hash function to reduce the number of TCAM lookups. As shown in the simulation results, the number of TCAM lookups was decreased by *m* times using *m*–byte jumping window. Our proposed scheme can scan thousands of patterns simultaneously at the high performance. We evaluated its performance using real packet traces and it was shown that our method is suitable for a multi-gigabit secure router that provides network intrusion detection functions.

In this paper, we described the algorithm for multiple pattern-matching. Since the signature such as Snort rule contains multiple patterns, we must match multiple patterns to detect one signature for a packet. In order to detect multiple patterns

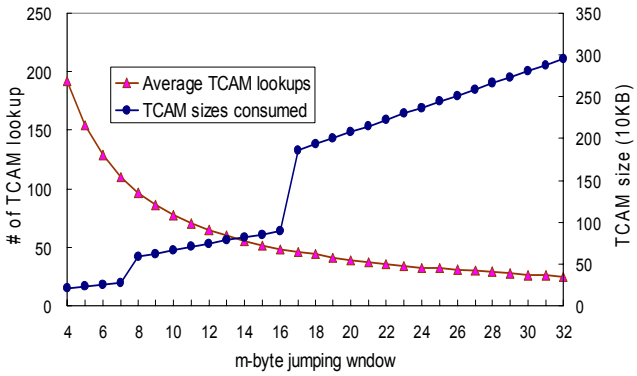

Fig. 8. The number of TCAM lookups and the size of TCAM in *m*-byte jumping window

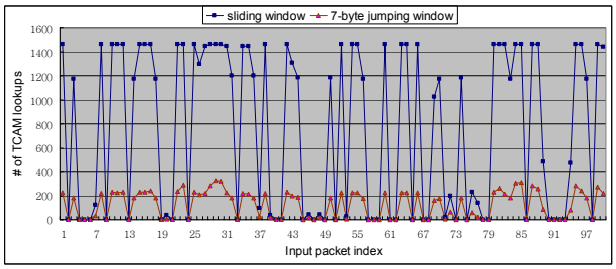

Fig. 9. The simulation result of TCAM lookups for packet trace

belonging to one signature, our algorithm can be extended by creating a rule table and a matching table We have implemented the proposed algorithm on the intel IXDP28xx platform [17]. Our proposed alogirthm scanned for packet payload at 1Gbps with one micro-engine that has single thread. We expect 10Gbps rate through code optimization and multiple micro-engines that have multiple threads. We've proven the feaiblity of the proposed algorithm with our experimental implementation that runs on the IXDP28xx platform.

REFERENCE

[1] D. Moore, et al., "The spread of the sapphire/slammer worm," *Tech. Report*, Caida, 2003; http://www.caida.org/outreach/papers/2003/sapphire/sapphire.html

[2] P. Jungck and S. S.Y. Shim, "Issues in high-speed internet security," *IEEE Computer Magazine*, vol. 37, no. 7, pp. 22-28, July 2004.

[3] J. Bo and L. Bin, "High-speed discrete content Sensitive pattern match algorithm for deep packet filtering," *Int'l Conf on Computer Networks and Mobile Computing*, 2003.

[4] F. Yu, R. H. Katz and T. V. Lakshman, "Gigabit rate packet pattern-matching using TCAM," *IEEE Int'l Conf on Network Protocols*, pp.174-183, Oct. 2004.

[5] M. Fisk and G. Varghese, "Fast content-based packet handling for intrusion detection," *Tech. Report CS2001-0670*, UCSD, May 2001.

[6] S. Wu and U. Manber, "A fast algorithm for multi-pattern searching," *Tech. Report, TR94-17*, U of Arizona, May 1994.

[7] A. Aho and M. Corasick, "Efficient string matching: An aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp.333-343, June 1975.

[8] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Communications of the ACM*, vol. 20, no 10, pp. 762-772, Oct. 1977.

[9] R. N. Horspool, "Practical fast searching in strings," *Software Practice and Experience*, vol. 10, no 6, pp. 501-506, 1980.

[10] Y. Huang, P. Zhang, S. Li, Y. Chen, and D. Zhang, "Research on distributed real time network information auditing system," *Int'l Conf on Information, Comm. & Signal Processing*, 2001.

[11] eSafe Gateway, http://www.eAladdin.com/eSafe

[12] Y. H. Cho, S. Navab, and W. H. Mangione-Smith, "Specialized hardware for deep network packet filtering", *Proceedings of FPL 2002*, pp. 452-461, Sep. 2002.

[13] S. Dharmapurikar, P. Krishnamurthy, T. S. Sproull and J. W. Lockwood, "Deep Packet Inspection using Parallel Bloom Filters", *IEEE Micro*, vol. 24, no. 1, pp. 52-61, Jan. 2004.

[14] IDT, *Integrated IP Co-Processor (IIPC) with QDR Interface, IDT75K52134/ IDT75K62134 User Manual*, Sep. 2002.

[15] Snort.org, http://www.snort.org/

[16] libpcap, http://ee.lbl.gov/

[17] Intel, *Intel 2800 Network Processor, Hardware Reference Manual*, Jan. 2004.