Hash-Based String Matching Algorithm For Network Intrusion Prevention systems (NIPS)

VINOD. O & B. M. SAGAR
ISE Department, R.V.College of Engineering, Bangalore-560059, INDIA
Email Id :vinod.goutham@gmail.com,sagar.bm@gmail.com

***Abstract* - Network Intrusion Prevention Systems (NIPS) are employed in-line with the network segment that needs to be protected. As the packets within the network passes through the NIPS device, the packets are inspected for the presence of any attacks. Like viruses, most intruder activities have some kind of signatures, hence a NIPS device contains a pattern matching algorithm to match the virus signatures within the rule list with the incoming network packets. When an attack is identified, the NIPS blocks the infected data packet with a unusual signature pattern. The pattern-matching algorithm must be able to operate at network speeds, while simultaneously detecting the main bulk of intrusions. This paper proposes an alternative algorithm using a Hash Function which uses a SRAM that creates fingerprints of the packet payload which are then compared with the patterns signatures. The proposed hash based system consumes around 0.56 times or 56 percent less memory than the memory consumed by the RTCAM method. It can also be observed from the results that as the TCAM width doubles the initial width the memory consumption increases around 1000kb the initial memory consumption value in RTCAM method. But in the case of hash based method as the block size is doubled the memory consumption increases by a small value around 200kb only from the initial memory consumption value. Hence the proposed hash based method is efficient than the RTCAM method in terms of memory consumption. Furthermore, the system is fully compatible with Snort's rules syntax, which is the basic standard followed for intrusion prevention systems.**

Index Terms - Hash Algorithm, NIPS, Padding, RTCAM, Snort Rules, SRAM, TCAM

I. INTRODUCTION

IPS products are basically Intrusion Detection Systems (IDS) that operate in-line and are thus dependent on pattern-matching to recognize malicious content within individual packets (or a group of packets). These systems deploy proactive defense mechanisms designed to detect malicious packets within normal network traffic. Once identified, the malicious traffic is usually blocked. NIPS systems are usually comprised of two major components: a pattern matching engine and a packet classification engine. The pattern matching engine's input is a packet and its output is a set of matched patterns belonging to the set of well known attack's signatures. There are a number of challenges in implementing a NIPS device; These all stem from the fact that a NIPS device is designed to work in-line, thus presenting both a bottleneck and a single point of failure. If a NIPS device struggles to keep up with traffic speeds, it becomes a bottleneck, thus increasing latency and reducing throughput. If the device fails it can impact the availability of entire network. The current trend for integrating security with network switches and routers both at the network edge and at the enterprise gateway, implies that the NIPS device must meet the network performance and reliability requirements [1].

Content addressable memories (CAMs) minimize the number of memory accesses required to locate the entry. Given an input key, the CAM device compares it against all memory words in parallel; hence, a lookup procedure requires one clock cycle. Unlike standard computer memory (RAM) in which the user supplies a memory address and the RAM returns the data word stored at that address, a CAM is designed such that the user supplies a data word and the CAM searches its entire memory to see if that data word is stored anywhere in it. If the data word is found, the CAM returns a list of one or more storage addresses where the word was found (and in some architectures, it also returns the data word, or other associated pieces of data).[1]

TCAM is a type of memory in which each memory cell can store three data states: 0, 1, and X ("Don't Care"). It is suitable for applications such as networking equipment because it achieves deterministic, high-speed searches by using simultaneous parallel operation to

[1] Content addressable memory available at : Http://en.wikipedia.org/wiki/Content-addressable_memory

compare data strings input from an external device with data strings stored in the memory and outputting the matches. This high degree feature comes at the cost of access time, storage density, and power consumption. Since the input key is compared against each memory word, each of the storage bit requires match logic to match a word line which signals a match for the given key. This extra logic and capacitive loading lengthens the access time and increases power consumption [2].

Snort is an open source NIPS that is commonly used in industry. Snort contains a database of rules with several thousands of attack signatures. Each of Snort's rules contains a header and several content fields. The header part consists of a packet identifier (protocol, source / destination IPs and ports), whereas the content part contains one or more patterns that may have some correlation between them. A rule is matched only if all of its patterns are matched with the expected correlation among them. NIPS devices which are compliant with this standard have a great advantage - the same database can be transparently imported from one engine to another.[2]

II. RELATED WORK

High Performance String Matching Algorithm For NIPS [1] which proposes an algorithm that places the set of the attack signatures in the TCAM. It is capable of matching multiple patterns and attains line rate speed and greater accuracy of detection is comparatively easy to analyse and implement. The RTCAM algorithm in this paper forms the base to the Hash algorithm proposed.

Longest Prefix Matching Using Bloom Filters [2]. The algorithm performs parallel queries on Bloom filters, which are an efficient data structure for membership queries, and obtains addresses of prefix membership in sets of prefixes sorted by prefix length. The use of this algorithm for Internet Protocol (IP) routing lookups results in a search engine providing better performance and scalability than TCAM-based approaches. The key feature of this technique is that the performance can be held constant for longer address lengths or additional unique address prefix lengths in the forwarding table given that memory resources scale linearly with the number of prefixes in the forwarding table. The approach is equally attractive for Internet Protocol Version 6 (IPv6) which uses 128-bit destination addresses, four times longer than IPv4.

[2] Snort available at : Yaron weinsberg, Shimirit turz-David, Tal Anker, "High Performance String Matching Algorithm For NIPS", 2005.

Fast Hash Table Lookup Using Extended Bloom Filter: An Aid to Network Processing [3] Hash table is used as one of the fundamental modules in several network processing algorithms and applications such as route lookup, packet classification, per-flow state management and network management. A poorly designed hash table can critically affect the worst case throughput due to multiple memory accesses required for each lookup. Because of which, high throughput requirement in turn underscores the need for a hash table having good and more predictable worst case lookup performance. This paper presents a novel hash table data structure and lookup algorithm which improves the performance of a naive hash table by providing better bounds on the hash collisions and memory accesses per search. The algorithm proposed here extends the multiple-hashing Bloom Filter data structure to support exact matches.

A High Speed Pattern Matching For Network IDS/IPS [4] proposes novel multiple string matching algorithm that can process multiple characters at a time thus achieving multi-gigabit rate search speeds. It also proposes an architecture for an efficient implementation on TCAM based hardware. Additionally it proposes novel optimizations by making use of the properties of TCAMs to significantly reduce the memory requirements of the proposed algorithm. Finally it presents extensive simulation results of network-based virus/worm detection using real signature databases to illustrate the effectiveness of the proposed scheme.

An Efficient TCAM Based Implementation Of Multi Pattern Matching Using Covered State Encoding [5] uses a state encoding scheme called a covered state encoding for the efficient TCAM-based implementation of the Aho-Corasick multi pattern matching algorithm, which is used in network intrusion detection systems. It also proposes constructing the modified Aho-Corasick NFA for multi character processing, which can be implemented on a TCAM using the covered state encoding. The covered state encoding takes advantage of "don't care" feature of TCAMs and information of failure transitions is implicitly captured in the covered state encoding.

III. PROPOSED SOLUTION

A. *Hash-Based Pattern Matching Algorithm*

Since TCAMs consume *150* times more power per bit than SRAM and currently cost about *30* times more per bit of storage. A lookup technique that employs standard SRAM requires less than four memory accesses per lookup and utilizes less than *11* bytes per entry for IPv4 and less than *44* bytes per entry for IPv6. It not only matches TCAM performance and resource utilization, but also provides a significant advantage in terms of cost and power consumption. Because of this reason TCAM can be replaced by an SRAM and for which a Hash based technique is proposed for pattern matching [2].

Due to the high price of TCAM memory and the increasing number of signatures, a TCAM oriented solution may not be acceptable to the industry. In order to elevate the problem, this paper proposes an alternative algorithm using a Hash Function which uses a SRAM that creates fingerprints of the packet payload which are then compared with the patterns signatures (previous works that are using hashing technique can be found in [3] [4] [6] [7] [8]).

The algorithm follows the same logic as the RTCAM algorithm [1]. A string of width 'w' bytes is fetched, the string's rightmost block 'b' bytes are inserted as a key to the shift table and the shift is retrieved. If the shift value N is not equal to 0, the text position is shifted right by N. If it is 0, then possible pattern match is found and it needs to look at the patterns pointers. It follows the hash entry's list, which points to the patterns that have the same key as the matched string.

In the RTCAM algorithm, a shift table is used. Instead of creating a shift table that maps a single character to a shift value, create shift values for a block which contains B characters. Using blocks reduce the amount of false matches. When a packet is processed, a block of characters is extracted from the search window (right to left) which are used as an index to the shift table. The shift value is retrieved from the shift table. If it is zero, then an exact match must be performed, otherwise we can shift the text. In order to minimize the search space, the patterns are linked using a hash value. Then calculate a hash value using the sliding window's text and locate the patterns bucket in the hash-table.

The Proposed hash based pattern matching system works as shown in the fig.1.

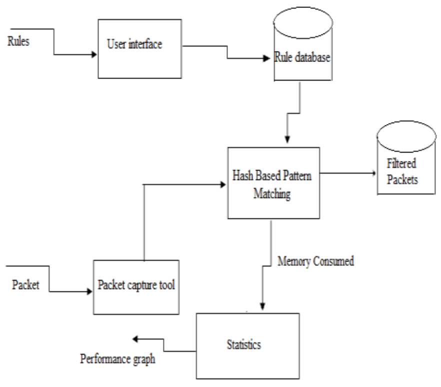

Fig 1 : Proposed Hash based Pattern Matching system.

Algorithm : Hash-Based Pattern Matching

```
1: T(Packet) = {Ti, 1 ≤ i ≤ n}
2: pos ⇐ 1;
3: shift ⇐ 0
4: width, w ⇐ default
5: Block, b ⇐ default
6: while pos ≤ n − width do
7: block ⇐ $T_{[pos+width-B,..,pos+width-1]}$
8: shift ⇐ SHIFT TABLE[block]
9: if shift is 0 then
10: key ⇐ hash($T_{[pos,..,pos+width-1]}$) {construct a fingerprint}
11: Step (2)
12: entry= SRAM.lookup(key)
13: Step (3)
14: shift⇐ entry.shift
15: if shift ≠ 0 then
16: pos⇐pos+shift
17: continue
18: end if
19: Step (6)
20: for all current=entry.key.next ≠ null do
21: if current.len ≤ width {exact match}  OR
checkSubPatterns(current.len ,pos, current.SRAM Ptrs)
                                    = = True then
24: MatchedList.add(current.Id, pos+current.Len)
25: end if
30: end for
31: end while
```

checkSubPatterns(len, pos, SRAM Ptrs)

```
1: while $pos \leq len - width$ do
2 : block ⇐ $T_{[pos+width-B,..,pos+width-1]}$
3: key ⇐ hash($T_{[pos,..,pos+width-1]}$)
4: $entry = SRAM.lookup(key)$
5: if $entry.shift \neq 0$ or $entry.id \notin SRAM\ Ptrs$ then
6: return false
7: end if
```

8: *return true*

9: end while

B. Data Structures that are used by Hash-based Pattern matching Algorithm.

In order to implement the hash-based solution several data-structures are maintained. Sub-Patterns Hash Table (HT) replaces the TCAM in the TCAM based algorithm. The table resides in SRAM and contains the r patterns divided by w which is the search window's width. Entries with less than w bytes are prepended (padding at the prefix) with the suffix of the previous part. A key is calculated on the sub-pattern and the sub-pattern is placed in the table accordingly.

1. Shift Table - A shift table for a block of characters B as in the algorithm presented by Manber in [7] [8] [9]. The block of characters is used during preprocessing the patterns to construct the shifting table. Within the sliding widow, it looks at the text, B characters at a time. For simplicity2 assume that the table size is SB, where S is the alphabet. Each entry corresponds to a distinct substring of length B. Let X = {x1,x2, ..,xB} be a string corresponding to the ith entry of the shift table. There are two cases: either X appear somewhere in one of the patterns or not. If X does not appear in any of the patterns, it stores m−B+1 in the corresponding shift entry, otherwise, it finds the rightmost occurrence of X in any of the patterns that contain it; suppose it is in P_j and that X ends at position q of P_j. Then it stores m−q in the table. If the shift value is greater than zero, it can safely shift. Otherwise, it is possible that the current substring we are looking at in the text matches some pattern in the pattern list. To avoid comparing the substring to every pattern in the pattern list, it uses the previous defined hash table (that minimizes the number of patterns to be compared).

2. Patterns Table - an array of patterns ordered by a patternID.

3. Matched Patterns List - each entry contains the matched patterns and its corresponding end position in the text.

IV. EXPERIMENTAL RESULTS

The proposed Hash Based pattern matching NIPS system and the NIPS system using earlier RTCAM Algorithm was developed on Java platform. The performance comparison was done for the same rule set on different TCAM width for RTCAM and on different block size for Hash algorithm. The signature pattern was obtained from the Snort rule list payload-content in the snort database. Basically these snort signatures are shorter (average is only *12* bytes). For the same signature set the TCAM width and Block size is varied in order to observe the variations in the memory consumptions which is as shown.

It can be noted that the memory consumed by RTCAM Based algorithm is comparatively more than the memory consumed by the Hash based method for the same signature set size. If the TCAM width increases memory consumption also increases considerably to a larger value. This is a major drawback in the RTCAM method as the TCAM width must be carefully chosen as it plays an important role during hardware implementation.

The proposed hash based system in this paper consumes around *0.36* times less memory than the memory used by the RTCAM method for a initial smaller TCAM width and block size. But as the TCAM width and the block size increases the memory consumed by hash method is *0.56* times less than the memory consumed by the RTCAM method. Hence the proposed hash based method is comparatively efficient than the RTCAM method.

1. When the TCAM width is *4* the memory consumed is *1448kb* and when the Block Size is *4* the memory consumed is *900kb*, as illustrated in fig 2.

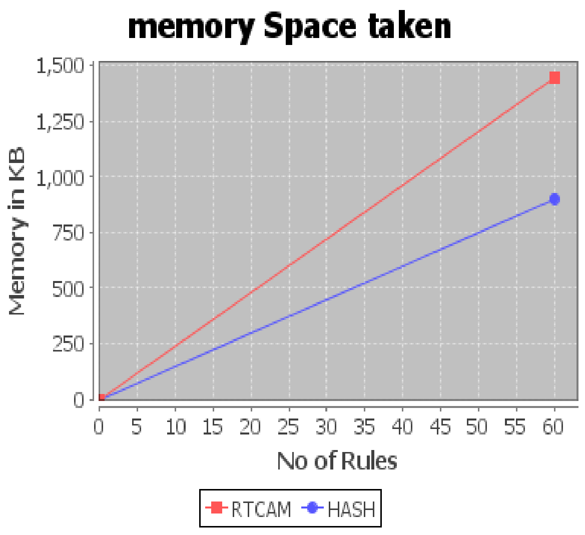

Fig 2 : A graph showing the memory consumed when TCAM width = 4 and Block size = 4.

2. When the TCAM width is *6* the memory consumed is *1782kb* and when the Block Size is *6* the memory consumed is *980kb*, as illustrated in fig 3.

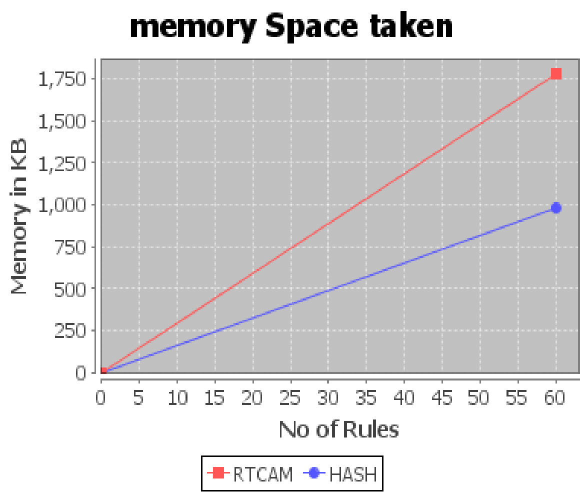

Fig 3: A graph showing the memory consumed when TCAM width = 6 and Block size = 6

3. When the TCAM width is *8* the memory consumed is *2512kb* and when the Block Size is *8* the memory consumed is *1107kb*, as illustrated in fig 4.

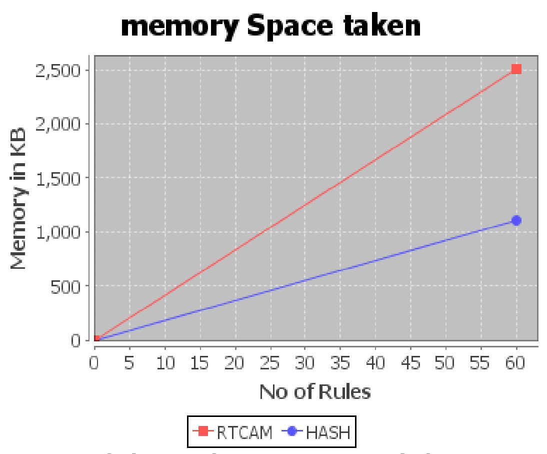

Fig 4: A graph showing the memory consumed when TCAM width = 8 and Block size = 8.

V. CONCLUSION & FUTURE WORK

The paper proposes a NIPS system that is based on a Hash based pattern matching algorithm that uses a static RAM (SRAM) instead of TCAM. This paper presents many advantages over RTCAM NIPS device. First, The proposed hash based system consumes around *0.56* times less memory than the memory consumed by the RTCAM method. Second, this software NIPS device can be implemented easily with less complexity. Third since the TCAM is replaced with the standard SRAM the cost of hardware implementation can be reduced and is economically suitable approach for industrial solution.

Even though the memory consumption rate can be reduced relatively the above solution cannot be directly used in industrial solutions. The reason is because the pattern's length varies quite often depending on source from which the pattern is obtained and also the nature of infection. For eg. the length of the ClamAV signatures are quite long (average of 124 bytes). There is a major difficulty in designing a unified algorithm that deals with long patterns and short ones since the performance is typically influenced by the overhead caused by the short patterns. A Padding technique can be developed to solve the above identified problem.

VI. REFERENCES

[1] Yaron weinsberg, Shimirit turz-David, Tal Anker, "High Performance String Matching Algorithm For NIPS", 2005, unpublished.

[2] Sarang Dharmapurikar, Praveen Krishnamurthy, and David E. Taylor " Longest Prefix Matching Using Bloom Filters", vol.14(2), pp.397-409, April 2006.

[3] Haoyu Song, Sarang Dharmapurikar, Jonathan Turner, John Lockwood " Fast Hash Table Lookup Using Extended Bloom Filter:An Aid to Network Processing", August 26, 2005.

[4] Mansoor Alicheery, Vijay Kumar, Muthuprassana," A High Speed Pattern Matching For Network IDS/IPS", vol.1, pp.187-196,2006.

[5] Sang Yun, " An Efficient TCAM Based Implementation Of Multi Pattern Matching Using Covered State Encoding", vol.62(2), pp.213-221, February 2012.

[6] R. M. Karp and M. O. Rabin, "Efficient Randomized Pattern Matching Algorithms", Technical report TR-31-81, Harvard University, Cambridge, MA, USA, December 1981.

[7] S. Wu and U. Manber, "Fast Text Searching with Errors", Technical Report TR-91-11, University of Arizona, Department of Computer Science, June 1991.

[8] S. Wu and U. Manber. Agrep , "A Fast Approximate Pattern-Matching Tool", In Proceedings USENIX Winter 1992 Technical Conference, pages 153–162, San Francisco, CA, January 1992.

[9] S. Wu and U. Manber, "A fast Algorithm for Multi- Pattern Searching ", Technical Report TR-94-17, Department of Computer Science, University of Arizona, May 1993.

◈◈◈