Chapter 6

FTSE: The FNP-Like TCAM Searching Engine

6.1 Introduction

As described in [28], most signatures in NIDSes are ASCII codes while network traffic is composed in binary data and signature matches rarely happen in real-world traffic. The more bytes we can skip during searching signatures in packets, the more performance we gain. In this chapter we propose a FNP-based [28] algorithm which utilizes Ternary CAM (Content Addressable Memory) as a pre-filter and achieves multiple gigabit performance in a relative low cost.

General memory components like SDRAM or SSRAM take an address as input and then return corresponding data stored in specified address. On the other hand, CAM components take data as its input and then return the location where the data stored. Besides, CAM simultaneously compares the desired information against an array of data, achieving a search time far less than with RAM-based algorithms. There are two basic forms of CAM: binary and ternary. Binary CAMs support storage and searching of binary bits, zero or one (0,1). Ternary CAMs (TCAM) support storing of zero, one, or don't care bit (0,1,X). Ternary CAMs are presently the dominant CAM since longest-prefix routing is the Internet standard.

Targeting to run in multiple gigabit per second, this design utilizes a small-size TCAM (2.25 Mbits) to be a filter, as FPGA/ASIC to process packets, and a DDR SDRAM to store the whole signature database. Although TCAMs are relatively expensive than SSRAMs and DDR SDRAMs, we will show that the cost/performance ratio is still excellent compared to use a high-end general purpose CPU with software algorithms. In our simulation, the performance of the proposed FTSE engine could be

up to 6Gbps or 8Gbps.

6.2 FTSE Algorithm

The pattern matching algorithm of *FTSE*, as its name, is very like to *FNP*. Both algorithms are designed to find the prefixes of signatures as potential matches. Please note the following used denotations can be referenced from Chapter 4.

G_0	a_0^0	a_1^0	a_2^0	...	a_{w-1}^0
	a_0^1	a_1^1	a_2^1	...	a_{w-1}^1
G_1	*	a_0^0	a_1^0	...	a_{w-2}^0
	*	a_0^1	a_1^1	...	a_{w-2}^1
G_2	*	*	a_0^0	...	a_{w-3}^0
	*	*	a_0^1	...	a_{w-3}^1
...	...				
G_{w-1}	*	*	*	...	a_0^0
	*	*	*	...	a_0^1

Figure 23. An example of signature layout in FTSE TCAM

The TCAM is a perfect component to achieve the goal finding the prefix of a pattern. The value *w* could be set to the width of the TCAM. First we divide the TCAM entries by *w* and therefore the TCAM contains *w* group of entries from G_0 to G_{w-1}. The Group G_0 stores the first *w*-byte prefixes of the ruleset. The Group G_1 stores the conjunction of a “don’t care” byte and *w*-1 bytes prefixes of the ruleset. The Group G_2 stores the conjunction of two “don’t care” bytes and *w*-2 bytes prefixes of the rulset, and so on. Figure 23 shows an example of the internal layout of the TCAM.

The matching procedure of the proposed *FTSE* algorithm is quite simple. Initially *PSW* is aligned with the first byte of the incoming payload. The string within

the *PSW S* ($t_0 \ldots t_{w-1}$) then is fetched and lookup in the TCAM. If *S* matches an entry in G_0, then *S* obviously matches with a *w*-byte prefix of a pattern and an exact match could be performed. On the other hand, if *S* matches an entry in G_x, $1 \leq x \leq$ w-1, the *PSW* will be shifted by *x* bytes for further processing. On the other hand, if there is no match in this round, then *S* can be skipped totally and the *PSW* will be shifted by *w* bytes. Figure 24 shows an example of the matching scenario. Initially the *PSW* is aligned with t_0 and therefore *S* is "123445555". Lookup *S* in the TCAM and we find a match in G_6 so that the GSW is shifted by 6 bytes and *S* is "555576543" then. By repeating the lookup-and-shift operation we find an exact match with only four lookups. However, it needs 29 (the length of *T*) lookups if the brute-force algorithm (continuously lookup and shift one byte) is applied.

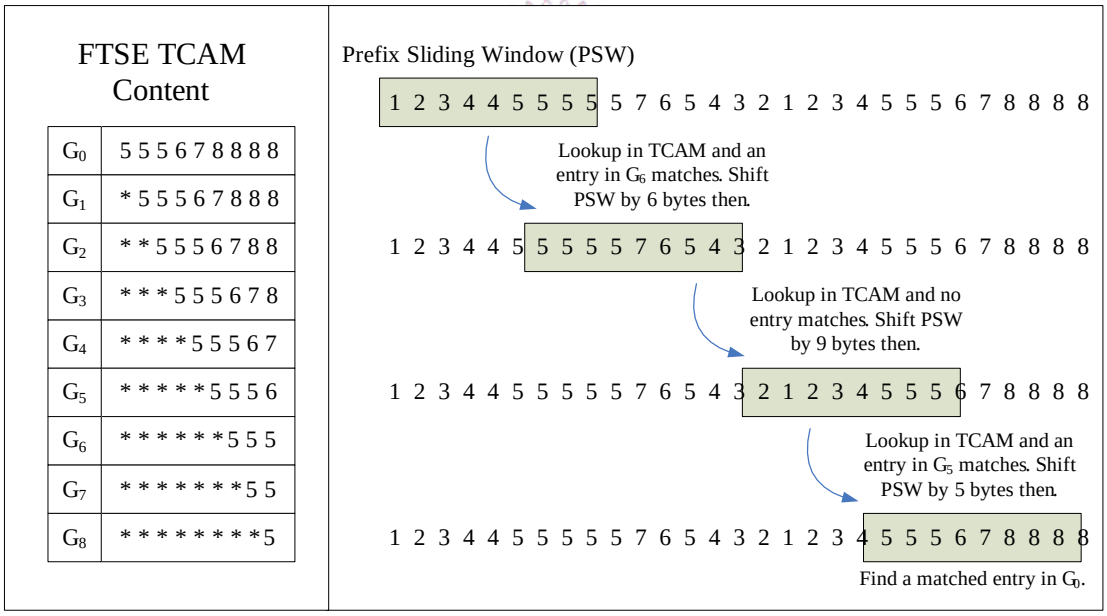

Figure 24. An example of the PSW movement

There're surely patterns whose sizes are greater than *w*. The full signatures are stored in another DDR SDRAM or SDRAM which are cheaper than TCAMs. The FTSE are two-staged. The first stage is the TCAM pre-filter which finds the *w*-byte prefix match and then pass the matched location and pattern ID to the second stage. In second stage an exact match will then be performed to make sure whether or not the

resting part is still matched.

6.3 Proposed Multiple-Pattern Matching Architecture

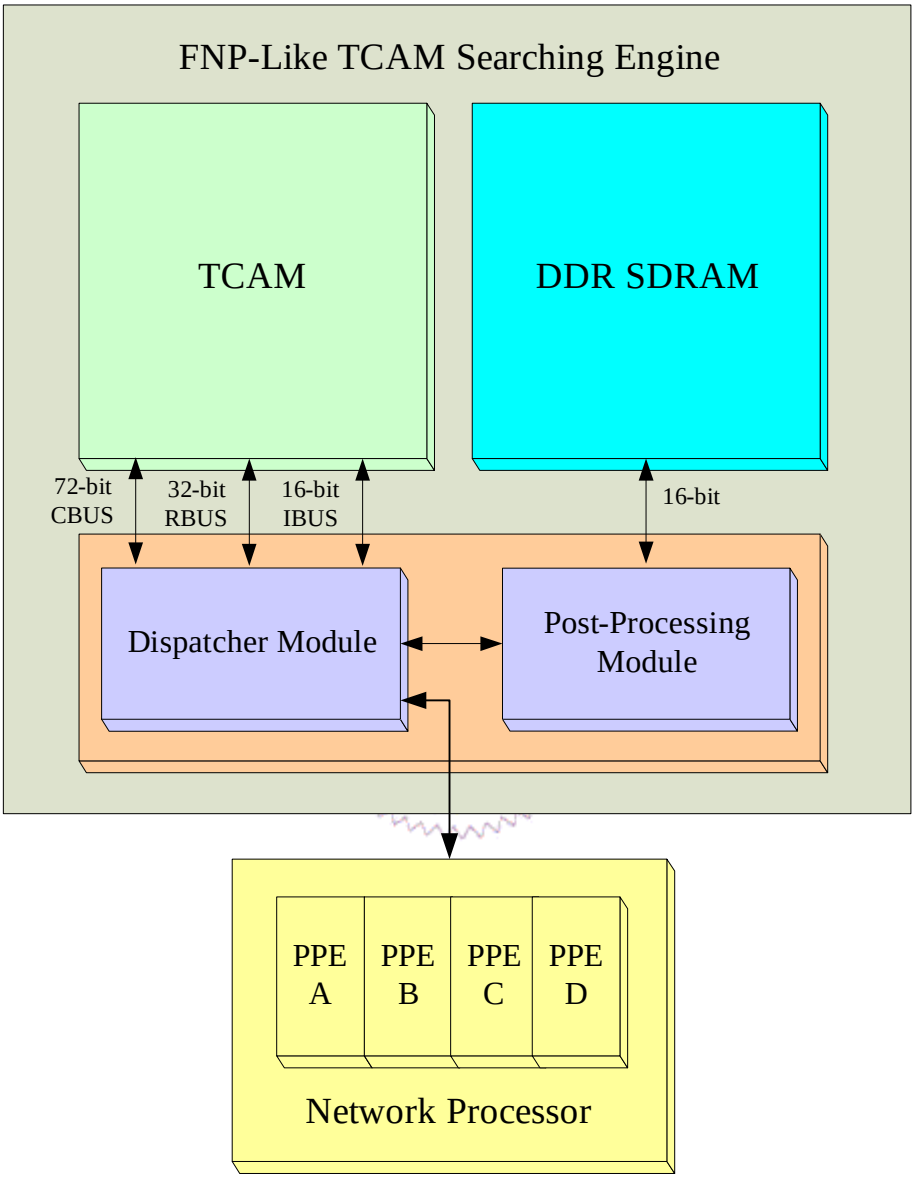

Figure 25. The hardware architecture of FTSE

Figure 25 shows an example of the hardware architecture of FTSE. Our algorithm is implemented into the FTSE Processing Module (FPM) which can be a FPGA or an ASIC. The FPM connects to a Network Processor which has multiple Packet Processing Engines (in this example, four) through a 32-bit SRAM interface or SPI-4 interface. Besides, the FPM attaches a TCAM and a DDR SDRAM for signature

storage.

The TCAM size is commercial-available 2.25Mbit so that the TCAM could be configured into 72-bit * 32K, where 72-bit is the width while 32K is the depth of the TCAM. Under this placement we can handle up to 3,640 signatures (32K/9) and it's quite sufficient since there is only 2K plus rules in current snort ruleset. The TCAM could be configured into different sizes like 144-bit * 16K surely. Since an exact match will be performed if the pre-filter finds the w-byte prefix match, the larger the width, the more accuracy the pre-filter is. However we will show later the 72-bit configuration is accurate enough while 36K depth can accommodate even the largest NIDS ruleset.

The FPM comprises two engines which controls the TCAM module and DDR SDRAM module respectively. The Dispatcher Module in FPM receives the packet payloads from the external network processor and stores the payloads into internal shared packet buffers. The FTSE engine supports multiple requests (packets) in the same time. The Dispatcher Module controls the *PSW* for each packet buffer and performs the lookup operation to the TCAM module. The Dispatcher module monitors the RBUS (Result Bus) of the TCAM after performing the lookup operations, and shifts the *PSW* of each buffer according to the returned results. Please note that the multiple buffer design serves several requests simultaneously and can be used to hide the TCAM latency as well. Figure 26 demonstrates how the Dispatcher Module performs TCAM search by round robin lookup. Dispatcher Module always issues "wrcmp" (write and compare) commands on IBUS (instruction bus) and the *PSW* for each packet is put onto CBUS (Comparand Bus). The lookup result for a packet will be returned on RBUS before its next search so that even the latency of the TCAM is three clocks we can still continuously keep it running. Another advantage of this pipelined process is to serve multiple requests in the same time. This tricky design

makes FTSE preferable for multiple core processors like Network Processors. On the other hand, if a match occurs in G_0, the Dispatcher will send a signal to Post-Processing module. The Post-Processing module performs the exact matching between potential signatures and payloads since the TCAM can guarantee the 9-byte prefix match only. Please note that the lookup-and-shift operation never ends unless it reaches the end of the payload. The Dispatched module and Post-Processing module run in parallel. We will show later that the post comparison happens rarely so that the processing time of post comparison can be ignored in terms of performance.

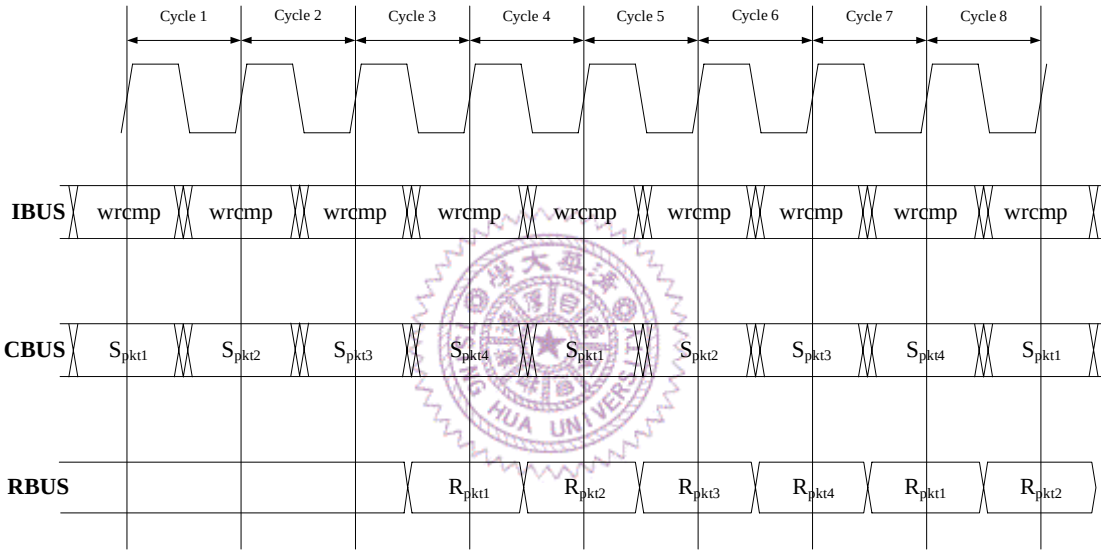

Figure 26. The lookup latency could be hidden by pipelined processing

Furthermore, some commercial TCAMs can be partitioned into different blocks and the TCAM lookup could be performed in specified blocks only and we can make use of this characteristic to improve the performance. We can partition the signatures by their protocols or layer-4 port numbers and therefore we can put signatures into different TCAM blocks accordingly. Every time when FTSE receives a packet, it doesn't have to search the whole TCAM but the corresponding block only. Since only the partial set is searched instead of the whole signature set, the average shift amounts should be increased by this way and therefore the performance is gained. Figure 27 shows an example of the entry layout in an eight-block TCAM. The signatures are

divided into eight blocks while each block still comprises nine groups to perform the lookup-and-shift operation.

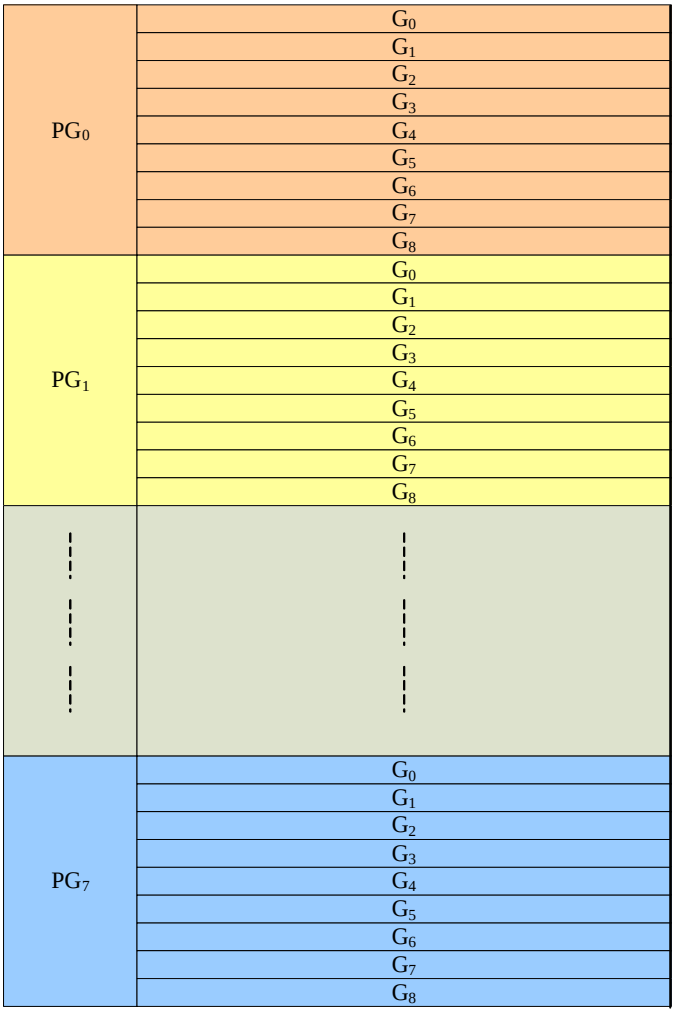

Figure 27. Using port group to enhance throughput.

6.4 Experiments of FTSE

To verify the effectiveness of the proposed *FTSE* algorithm, a simulation program running on a general PC is developed. We observed its efficiency and performance based on the behavior under certain packet traces and ruleset. The current Snort ruleset, containing 2,111 rules, was employed as the default searching pattern. To be simplified we employ all the patterns into a big table without dividing into groups. The full-packet traces can be derived from the “Capture the Capture The

Flag” (CCTF) project held in DEFCON [10] annually. DEFCON holds a “Capture The Flag” contest every year for hackers to compromise the prepared target servers. CCTF is named because DEFCON sniffs and captures the packets during the contest. We believe that the packet traces tend to match more signatures comparing to others by nature.

Table V shows the total payload sizes and number of pattern-matched events of different packet traces. As we mentioned before, matched events rarely happen, and it stands true even in CCTF packet traces. The highest matching rate in our experiments is about the packet trace “defcon_eth2.dump4” where there’re plenty of repetitious identical packets. Even so the matched rate is still only two percent. The assumption is very important because the FTSE comprises two stages and the exact matching operation in DDR SDRAM should be as less as possible so that its latency could be hidden when the lookup-and-shift operation in TCAM keeps running. Therefore the overall performance of the FTSE relates to the lookup-and-shift stage only.

Table V. The matched event rarely happens in our experiments

Trace Name	Total Payload Size	# of Matched Events	Percentage
defcon_eth0.dump2	823104675	2108683	0.25%
defcon_eth0.dump3	807112191	288803	0.04%
defcon_eth2.dump3	166037560	1753897	1.06%
defcon_eth2.dump4	484240159	9674245	2.00%
defcon_eth2.dump5	483417572	1063797	0.22%
defcon_eth6.dump3	419584806	905018	0.22%

Table VI demonstrates the efficiency of the proposed filter. For every packet trace we measure the number of different shifts after TCAM lookups. In this experiment the probability of shifting eight bytes and nine bytes is over 77% and the

overall expected shift value is 7.77. Please note that we believe it can be improved if the packet traces are normal ones instead of CCTF since they tend to match signatures in nature. This experiment demonstrates our approach has significant improvement over traditional TCAM lookup and even the bloom filter since they all need *N* clocks to handle an *N*-byte packet while FTSE only needs *N*/7.77 clocks to search. Modern TCAMs can perform 100M searches per second. In our simulation, if we lookup TCAM every 7.77 bytes, the estimated performance would be (7.77 * 8 * 100) = 6.216 Gbps. If a 133 MSPS (Million Searches Per Second) TCAM is chosen in our design, the FTSE engine is able to achieve more than 8Gbps throughput which surpasses any known software solutions so far.

Table VI. The number of shift bytes are greater than 7 in over 80% cases

	Shift = 1	Shift = 2	Shift = 3	Shift = 4	Shift = 5	Shift = 6	Shift = 7	Shift = 8	Shift = 9
defcon_eth0.dump2	2108683	154456	104394	449579	164982	1766720	4481566	33642128	56021300
defcon_eth0.dump3	288803	18625	491119	76339	44412	35180508	6377031	47577707	18307152
defcon_eth2.dump3	1753897	104576	127674	174747	755641	3521271	2824996	7170874	5603226
defcon_eth2.dump4	9674245	67977	74189	3024168	224238	898825	3138514	23459736	26791804
defcon_eth2.dump5	1063797	77641	85329	220704	186971	2572922	4324869	21396086	28701579
defcon_eth6.dump3	905018	30018	39806	202450	74022	2997209	2092225	16451913	27957458
Total	15794443	453293	922511	4147987	1450266	46937455	23239201	149698444	163382519
Percentage	3.89%	0.11%	0.23%	1.02%	0.36%	11.56%	5.72%	36.87%	40.24%

Table VII demonstrates the accuracy of the lookup-and-shift operation in TCAM as a filter. This table lists the number of nine-byte prefix matches against the number of real matches. It shows that when we have a nine-byte prefix match, we have over 90% probability that it's a real match. This experiment indicates that our "72-bit * 32K" configuration is most suitable for the implementation of an NIDS since it can

accommodate the full ruleset and keep the high accuracy in the same time.

Table VII. The accuracy of the pre-filter is more than 90% in our experiment.

Trace Name	# of prefix matches	# of real matches	Percentage
defcon_eth0.dump2	750842	784201	95.75%
defcon_eth0.dump3	91646	98997	92.57%
defcon_eth2.dump3	1169302	1245920	93.85%
defcon_eth2.dump4	2981885	3277422	90.98%
defcon_eth2.dump5	237316	367953	64.50%
defcon_eth6.dump3	448559	523599	85.67%
Total	5679550	6298092	90.18%

In summary, our experiments showed that the 2nd stage lookup happens rarely so that its latency could be covered by keeping TCAM running in parallel. Our TCAM lookup approach is shown to be very fast and efficient and the performance is 7.77 times faster comparing to bloom filter in our test. With the support of suitable commercial- available TCAM we can achieve 6Gbps to 8Gbps by our design. On the other hand, the accuracy of the FTSE filter is also quite good (over 90%) in the experiment. These characteristics make FTSE a very practical and powerful engine when applying in an NIDS.

FTSE comprises a TCAM to store signature prefix, a DDR SDRAM to store the whole signature database, and an ASIC/FPGA where the main algorithm running. The cost of this engine is relatively lower than high-end CPUs but the gain in performance is better. There are two stages in FTSE, and the first stage is a pre-filter which filter out the strings impossible to match while the second stage performs exact match between a 9-byte matched candidate and the signature. These two processes work in parallel. In our simulation, the probability of passing strings to second stage is only

0.2% to 2%, and the first stage runs seven-time faster than brute-force algorithm. On the other hand, the accuracy of the pre-filter in first stage is over 90% in our test. Furthermore, the FTSE utilizes pipelined processing for multiple packets to hide the TCAM lookup latency and serves multiple PPEs in the same time. These characteristics make FTSE engine very suitable for implementing the high-speed NIDS by co-working with high-end Network Processors.