A configurable Classification Engine for Polymorphous Chip Architecture

Meeta Yadav, Patrick Hamilton, Rick Sears, Yannis Viniotis,
Thomas M Conte, Paul D Franzon
Department of Electrical and Computer Engineering
North Carolina State University, Raleigh, NC 27695
{myadav, phamilt, rnsears, candice, conte, paulf}@ncsu.edu

Abstract

The ever-increasing demands for bandwidth requirement, faster IP forwarding, efficient and effective firewall and flexible differentiated services has resulted in the evolution of sophisticated Network Processor Units (NPUs). We describe a novel approach to implement a pipelined, configurable IPv6 and IPv4 co-processor for a Network Processor Unit. The co-processor is capable of providing Forwarding, Firewall and Differentiated Services. We use trie based algorithmic approach for the implementation of the design and extend it by using a morphable data path. The pipelined architecture provides significant improvement in the lookups and updates. The SRAM contains preprocessed IP addresses for forwarding and rules for Diffserv and Firewall engines. Our design scales well with the number entries. The SRAM with preprocessed rules is coupled to the engine with a bus and results in a throughput of 28 Million look up per second for an ASIC implementation and an update time of 8ms, this results in an update rate of 125 entries per second.

1. Introduction

The increase in the feature requirements of the Network Processor Unit has brought upon the need for sophistication in co-processor hardware. The basic classification functions are

1) Forwarding,
2) Differentiated services, and
3) Firewall.

We propose a pipelined configurable co-processor capable of performing firewall, diffserv and forwarding for IPv4 and IPv6. The engine provides faster lookup, faster updates and memory compaction. The engine scales with the number of IP addresses, firewall rules and diffserv rules. The basic goals of our design are

1) To perform fast next hop address lookup and packet classification based on an incoming packet,
2) For the design to be morphable by software,
3) To allow for faster updates, and
4) To have memory optimizations with configurable boundaries.

In this paper we use an algorithmic approach to IP forwarding, originally developed by Mehotra et al. [8], we extend it by using a morphable data path, to also provide high performance Diffserv and firewall. We show how this algorithmic approach can achieve a performance of over 28 million lookups per second for these tasks. We also address the critical table build up problem, outlining a solution that can sustain 125 upates per second, and takes less than 8 ms to complete updates. The solution described herein outperforms CAM based solutions in terms of power consumption, area, and cost while remaining competitive in terms of throughput and update times.

Packet forwarding and classification are memory and computation intensive tasks. In our design we propose a strategy for optimizing IP address and rule storage by preprocessing the entries to reduce the best-case complexity for rule lookup. We employ the trie based scheme that enables SRAM compaction. The design's pipelining enables faster lookups and increased throughput and the memory and stage design allow for faster updating

The rest of the paper is organized into the following five sections. In Section 2 we discuss some of the related work and motivation. Section 3 describes the algorithms used. Section 4 outlines our design and hardware implementation In section 5 we present our results for area and speed for an ASIC implementation and include memory sizes. We conclude in Section 6.

2. Related Work

Various designs for fast classification engines exist today whose underlying architecture is based on CAM or trie data structures to partition the routing/rule tables.

Girija et al [9] proposed a CAM based designs. CAM engines, despite their increased power requirements, cost and board area are used in several coprocessor design. CAMs are inefficient in representing filters with port ranges. There are several other algorithmic alternatives that, either consume more memory, or have increased memory accesses or suffer from scalability issues.

Existing trie-based schemes include direct and indirect lookups. These schemes require large amounts of memory to store the forwarding tables. The number of lookups is small (1–2) for these schemes however they do not scale well with number of entries. Binary tries, store data fairly efficiently. However, they require a large number of memory accesses compared to the direct or indirect lookup schemes. Variations of the basic binary trie such as Patricia [2] and LC tries [3] improve performance to some extent, but the average number of memory accesses is still fairly large.

Basu et al [4] use pipelined Forwarding engines with fast incremental updates that balance memory utilization across multiple pipeline stages and minimize disruption to the forwarding process caused by route updates. This system however suffers from scalability issues, as the memory requirements per stage make the implementation of an on-chip memory difficult. For a million entries in the forwarding table their scheme has memory increased by a factor of 5 over our proposed design. Their design also supports only IPv4 and also lacks the configurability feature which is key in configuring the memories and altering the boundaries for better trie distribution.

Sawhney [13], in his thesis focuses on a forwarding engine for a million entries IPv6 routing tables. Memory requirements are analyzed for a trie-based scheme and a binary search scheme for IP address lookup. The hardware described however is not pipelined and does not support IPv6

3. Algorithm

Mehrotra et al [8] propose an algorithm that compacts the trie data structure to easily fit it on a on-chip SRAM. The row of the DRAM is calculated by the SRAM and the next hop address is read from the DRAM. This work is the basis for our design. We alter the memory to allow faster updates and pipeline the design to handle faster lookups as well as support for IPv6.

3.1 Tries

The SRAM and DRAM databases are built from the conventional multiway trie structure. The SRAM database contains information that represents the topology of the trie, while the DRAM contains the next-hop addresses corresponding to the leaves of the trie. In addition to the SRAM and DRAM databases, an array (*Level*) is also maintained in the SRAM. The route lookup is done in two stages. In the first stage the SRAM is used to traverse to the longest matching leaf node in the trie, while in the second stage the DRAM is read to get the next-hop address.

3.1.1 Trie Buildup

The data structure to be stored in the SRAM and DRAM are built from the corresponding multiway trie. Figure 1 below shows a 4 way trie.

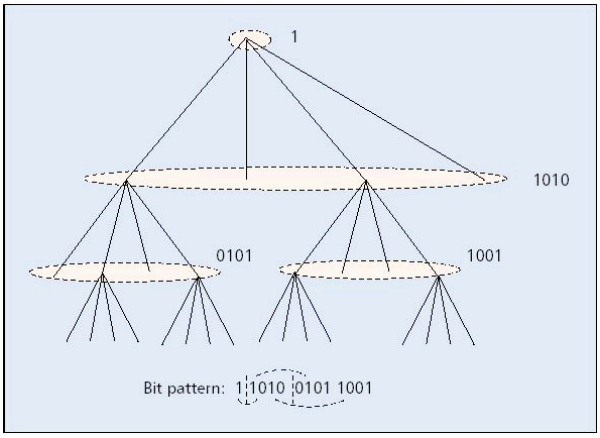

Figure 1. Sample four way trie and its bit pattern

We describe the implementation using a 16-way trie, although any degree of trie can be built. The trie is built as follows:
Step 1: Read each entry from the routing table and store it in a list. Sort the list in ascending order. For prefixes of differing lengths where one prefix forms the beginning of the other, the prefix with fewer prefix character is considered to be shorter For example, 10* is considered smaller than 100*. This ensures that while building the trie, parent nodes are processed before child nodes.

Step 2: Create the root node of the trie and initialize the child node pointers to NULL.

Step 3: Read each entry from the list and expand if necessary to complete the trie (to make sure that every internal node has X children, where X is the trie degree). Add appropriate nodes to the trie along with their next-hop addresses.

Step 4: Once the trie is built, construct the SRAM and DRAM data structures and the array Level via a breadth-first traversal of the trie. The SRAM is built by writing a 1 for every internal node and a 0 for leaf nodes, as shown in Figure 1. When constructing the SRAM data, we assume the existence of the first "1" which represents the root node and hence it does not need to be stored. The DRAM is built by writing an entry for every node in the trie in a breadth-first order.
The Trie depth is given by the formula.

$$Trie\ depth = No.\ of\ Address\ bits/log2X$$

where X - is the degree of the trie.

Trie depth is also the number of lookups required during insertion of an entry into the trie in the worst case. Since building the trie requires inserting N entries, where N is the total number of entries in the routing table, the total number of memory lookups while building the trie is

$$Memory\ Lookups = N * D$$

N - total number of entries in the routing table.
D - is the depth of the trie.

3.1.3 SRAM Compaction

Mehrotra's [8] trie-based approach is a novel method to compress the forwarding table information by reducing the trie path-information. The required SRAM is small enough (about 35KB for a routing database 30,000 entries) to easily fit on a chip. This is significantly important especially when moving to IPv6 where larger routing tables or multiple tables for different hierarchies are used. The data for our case, using this scheme is compacted to approximately 2 bytes for every entry in the routing table (for a 16-way trie constructed with a million entries). Also, the overall memory consumption (SRAM and DRAM) using this scheme is almost half that required in conventional implementations.

The amount of compaction achieved is much higher than other existing schemes, making a hardware implementation feasible. The compacted information can be stored in an on-chip SRAM and the final next-hop addresses are stored in an off-chip DRAM. To perform a route lookup, trie traversal is done in the SRAM and a final DRAM access is required to determine the next hop address.

3.2 Array of Tries

The Diffserv and Firewall engines are built based on an Array of Tries. To start constructing this array the IP address field (source/destination) is utilized since it guarantees enough uniqueness.

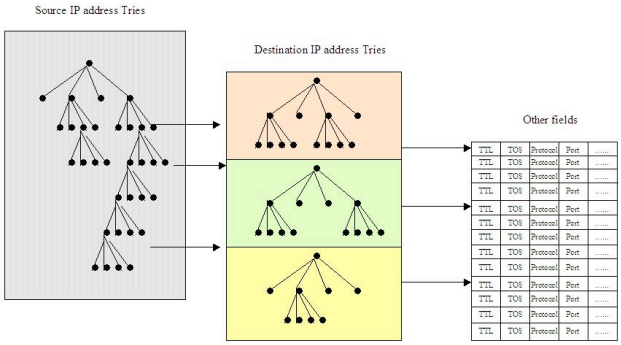

Figure 2: Array of Tries.

The first trie is built on the source IP address and the second corresponding trie is built for destination IP addresses that share this source IP address. Based on information from the current firewall and Diffserv rule sets we infer that there can be packets originating from a single source to multiple destinations. Thus the search for the destination address is narrowed down. The result of traversing the destination trie generates pointers to a memory populated with the remaining fields, where a direct comparison can be made to determine the action.

3.3 Range lookups

Port number ranges and IP address ranges are a common occurrence in rule sets. Srinivasan et al [10] propose a simple mechanism of converting ranges to prefixes. The ranges are thus pre-processed to prefixes and are translated into part of the tries. Gupta et al [5] further define that a range of width W can be represented by at the most 2W-2 prefixes.

4. Design and Implementation

IP address and rule information can be maintained as a trie for fast address lookup and search to determine membership of the incoming packet. We use the trie approach for IP forwarding and an Array of tries for Differentiated services and Firewall.

Some of the important design decisions for a Trie are
1) The number of entries in the trie,
2) The IP version, and
3) The SRAM compaction.

The design has four stages each of which pipelined internally. The memory for each of these stages is partitioned based on the address spaces and each partition is further divided based on the levels they comprise of.

4.1 Block Diagram

We discuss the block architectures of the forwarding engine, firewall engine and differentiated services engine in this section.

4.1.1 Pipeline stage.

The forwarding engine consists of four stages. The stages are split on the basis of levels of the Trie. For our implementation, we chose the order of the trie to be 16 since the IPv4 address is 32 bits long the depth of the trie is 8. Similarly the depth of 128 bit long IPv6 address is 32. In Figure 3 below we show the different stages connected together.

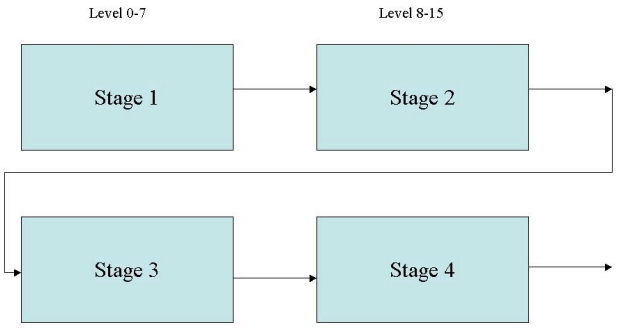

Figure 3: Pipeline Stages.

Figure 4 below shows the internal blocks of a single pipeline.

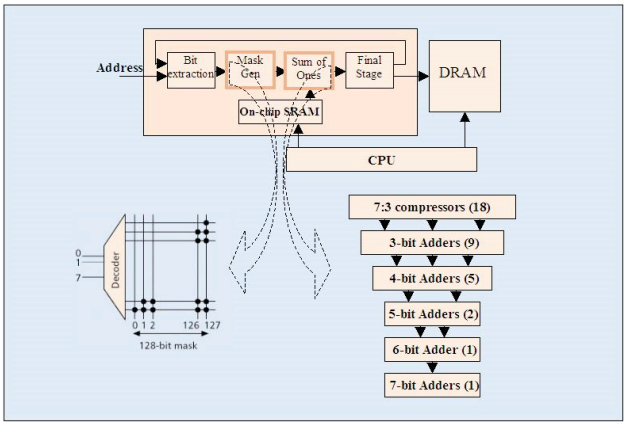

Figure 4 : Logic Blocks

The four basic building blocks of a single pipeline are:

1) SRAM Access,
2) Mask Generation,
3) Sum of 1's, and
4) Final state logic.

Memory for Stage 1 consists of the trie from level 0-7. Since IPv4 consists of only 8 levels, the lookup or classification for IPv4 ends in Stage 1 and does not traverse through the rest of the stages. Stage 2 consists of the trie from levels 8-15, Stage 3 consists of trie from level 16-23 and Stage 4 consists of trie from level 24-31. Typically IPv6 prefixes are 64 bits wide hence the output can be obtained by traversing through stages 1 and 2. We do not maintain a constant look up time hence the next address is obtained when the trie traversal ends.

Each pipeline cycles through a stage 8 times to traverse the 8 levels of the trie. The pipelines have been designed to avoid memory contention.

Figure 5 below depicts the pipeline structure within each stage. Incoming packets are fed into a stage at the rate of one packet every 9 cycles.

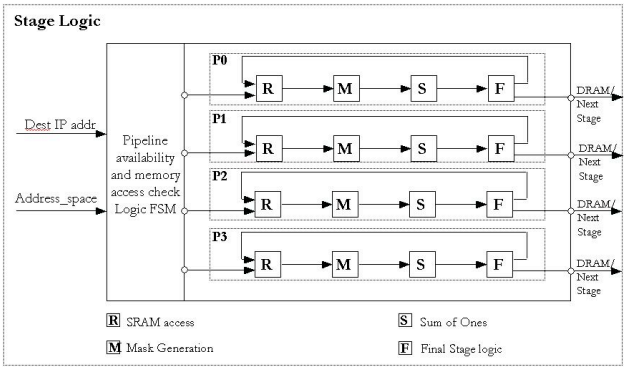

Figure 5: Forwarding Engine stage diagram

Figure 6 shows the components external to the lookup engine. Each stage has its dedicated memory that is populated with corresponding trie data.

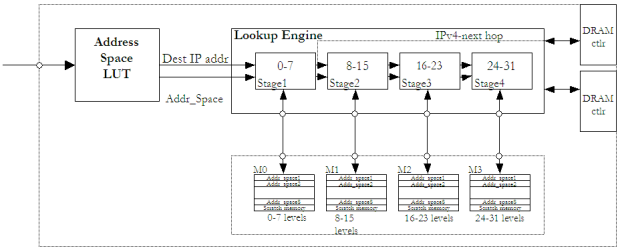

Figure 6 : Stage Pipeline with memories

When a packet is received at stage 1 the trie from level 0-7 are traversed based on their source IP address. If the prefix to be matched is longer than 32 bytes then the packet is passed to the next stage. For an IPv4 packet stage 1 points to the DRAM to look up the next hop address but for a firewall and Diffserv it forwards the packet to traverse the tries of the destination addresses,

the result of which directs the packet to the corresponding fields in the memory that need to be compared. This is achieved by creating filters in software.

The filter could also be created based on the destination IP addresses, which in turn point to a subset of source IP addresses. The uniqueness of the IP addresses gives the advantage of smaller search space. There is no substantial degradation in performance seen in the event such uniqueness does not exist.

4.1.2 Memory Structure for Forwarding Engine.

The memory structure as depicted in Figure 7 is one of the driving forces in the efficiency of the design. The address space of the IP addresses are split based on the first octet of the IP address; the different ranges are then created for the IP addresses. This feature in the design is configured by software. The memory is further partitioned into different levels. This enables different stages to access the memory without memory contention. In the event of an update a new trie is constructed in software and loaded onto a spare memory for swapping. Each address space occupies typically 288 KB. Software prescribes the number of address spaces and their boundaries.

4.1.3 Memory Structure for Firewalling and Diffserv engine

The memory structure for the Firewalling and Diffserv is similar to the Forwarding engine but contains additional memories for secondary trie and multiple fields of the Differentiated services rules.

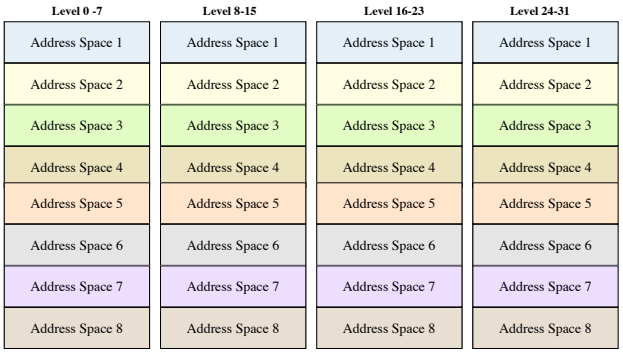

Figure 7: Memory Structure for the different stages.

The memories are separated based on the protocol field and then based on the source or the destination IP addresses. Hence the memories are indexed using the Protocol field and then use the source and destination addresses. We assume the Differentiated Services to be provided by the ISPs on the basis of rules, which would classify packets based on the following fields

1) Protocols,
2) Ports (including ranges),
3) Source IP address (including ranges),
4) Destination IP addresses (including ranges),
5) type of Service field,
6) DSCP value and
7) Flags (e.g. STN, FIN etc.)

5. Results

5.1 Performance

We present our performance results for an ASIC based implementation. For the ASIC implementation the design was synthesized using the Virginia Tech 0.25um library. The timing analyses were done with a clock skew of 300ps. The synthesis provides a cycle time of 2 ns for the design with a total cell area of 4.8 sq. mm. It takes typically 32 cycles to process IPv4 packets and 64 cycles to process IPv6 packet (with typical prefix length of 64 bits). We achieve a throughput of 28 Million lookups per second.

5.2 Memory Requirements

The required SRAM is small enough to easily fit on a chip. The data in our case is compacted to approximately 2 bytes per entry as per the formula in [11] for a million entry routing table. We calculated the maximum SRAM requirement and the expected SRAM requirement for the forwarding engine. The maximum SRAM requirement arise from extreme cases that are not observed in present day routing tables. It is further observed that the expected SRAM requirement is less than that required for the actual routing tables. Therefore we use a scaling factor to allocate sufficient SRAM for the desired routing table size.

The theorem explained in [11] shows that the expected SRAM memory (bits/entry) for *n* random uniformly distributed routing table entries is given by:

$$E(Mem(bits/entry)) = M/\ln(M),$$

where M is the degree of the trie.

In our case for a 16 degree trie the memory requirements (using a scaling factor of 3) are as follows:

$$SRAM = 6Mbit * 3 = 18Mbits$$

Assuming a byte to store the port numbers, the memory requirement for the DRAM would be:

*DRAM = 18Mbit *8 = 144Mbits*

The four memories (M0-M3) as explained in section 4.1.2, are split across the 4 stages and serve separate levels. From the prefix distribution of IP addresses described in [12], it is observed that the 24-bit prefixes (Level 6) are most dominant. For IPv6 packets the 64-bit prefixes are found to be the most dominant. Based on this information the memories M0-M3 have been partitioned.

For a DiffServ and Firewall engine the typical memory requirements are likely to be less than the above since the number of entries (rules) are less. The number of rules for a DiffServ and Firewall are typically 20,000 and 10,000 respectively hence the trie memories for Diffserv is approximately 90 KB and for Firewall it is 45 KB.

6. Conclusion

Our design achieves a throughput of 28 Million lookups per second. We parallelized the design by creating deep pipelines. SRAM compaction results in reduced memory requirements as compared to other designs. The area for our design is 4.8 sq mm. in a 0.25um technology.

The functionality of forwarding engines has now grown to encompass multi-field classifications. There is a need for a system that can efficiently perform the functions of Firewall Differentiated Services and Forwarding. These engines should also not suffer from constraints such as excessive memory requirements and slow updates. The design we propose uses SRAM compaction with a memory of 2.25 MB for typical cases and performs updates in 8 ms. The pipeline makes the design faster and and increases the throughput. We show that with our algorithmic approach we can achieve a performance of over 28 million lookups per second for these tasks. We also address the critical table build up problem, outlining a solution that can sustain 125 updates per second, update takes less than 8 ms to complete.

8. Acknowledgement

The authors would like to acknowledge the contributions of Dr. Pronita Mehrotra and Ishdeep Sawhney.

9. References

[1] P. Gupta, S. Lin, and N. McKeown, “Routing Lookups in Hardware at Memory Access Speeds,” Proc. IEEE INFOCOM ’98, San Francisco, CA, 1998, pp. 1382–91.

[2] K. Sklower, “A Tree-Based Routing Table for Berkeley Unix,” Tech. rep., UC Berkeley.

[3] S. Nilsson and G. Karlsson, “IP-Address Lookup Using LC-Tries,” IEEE JSAC, vol. 17, June 1999, pp. 1083–92.

[4] Anindya Basu, Girija Narlikar, "Fast Incremental Updates for Pipelined Forwarding Engines", INFOCOM 2003

[5] Gupta, McKeown. “Alogrithms for Packet classification”. Computer systems Laboratory, Stanford.

[6] J. Xu and al., “A novel cache architecture to support layer-four packet classification at memory access speeds,” in Proc. of Infocom, mar. 1999.

[7] Florin Baboescu, Sumeet Singh, George Varghese, "Packet Classification for Core Routers: Is there an alternative to CAMs?",INFOCOM 2003

[8] P. Mehortra, P. Franzon, Novel Hardware Implementation for Fast Address Lookups, 2002 Workshop on High Performance Switching and Routing

[9] Girija Narlikar, Anindya Basu, Francis Zane, "CoolCAMs: Power-Efficient TCAMs for Forwarding Engines", INFOCOM 2003.

[10] V. Srinivasan, G.Varghese, S.Suri, M.Waldvogel. “Fast and Scalable Layer four Switching”. SIGCOMM’ 98 vancouver.

[11] Pronita Mehrotra, “Memory Intensive Architectures for DSP and Data Communication”, Ph.D Dissertation, NCSU, 2002

[12] Ruiz-Sanchez, M.A.; Biersack, E.W.; Dabbous, W., Survey and taxonomy of IP address lookup algorithms, IEEE Network April 2001

[13] SAWHNEY, ISHDEEP SINGH, Forwarding Engine for IPv6, Masters Thesis, North Carolina State University.