

# Routing, L2 Addressing, and Packet Filtering in a Hardware Engine

David Antoš<sup>1,2,3</sup> and Vojtěch Řehák<sup>1</sup>

<sup>1</sup> Faculty of Informatics,  
Masaryk University,  
Botanická 68a, Brno 602 00, Czech Republic

<sup>2</sup> CESNET, z. s. p. o.,  
Žitkova 4, Prague 160 00, Czech Republic

<sup>3</sup> Institute of Computer Science,  
Masaryk University,  
Botanická 68a, Brno 602 00, Czech Republic  
`antos@ics.muni.cz`, `rehak@fi.muni.cz`

**Abstract.** To improve throughput of personal computers used as Internet routers, hardware acceleration can be used. Packet classification unit employed in the design utilizes content addressable memory combined with comparison instructions. Routing, link layer addressing, and packet filtering has to be performed in a single operation. We have developed a representation of the first two called routing-ARP table, and representation of filters based on decision diagrams.

In this paper, we describe a method to combine them all together and convert the resulting structure into the hardware device. As a special case, the algorithm converts a decision diagram into a first-match structure. Dealing with implementation and limited hardware resources is mentioned.

## 1 Introduction

Throughput of personal computers used as network routers is comparable with middle-class commercial boxes. Main limitation of their performance is however the throughput of system bus [1]. Offloading the traffic to an acceleration card can mitigate the bottleneck, avoiding common buses in the computer. *Packet classification* is the central part of the accelerator. While operating systems classify packets in discrete stages, our target hardware architecture has a single-run engine called Lookup Processor (LUP).

The LUP consists of two units: (1) a Content Addressable Memory (CAM) that searches parts of packet headers in parallel and acts as a “big case statement” returning the first matching record and (2) comparison instructions that finish the search. The OS classification setting must therefore be combined into a single operation and converted to a format of CAM and comparison instructions. The conversion is performed in stages. First, routing table and link layer addressing information is compiled to a structure called routing-ARP (RA) table.

Packet filter is represented by a Filtering Decision Diagram (FDD). We summarize those structures in Section 3. In this paper, the final stage is presented, i.e., combining the structures together by distributing packet filters based on destination address space (Section 4) and transforming the structure to the target architecture (Section 5). Interesting properties of the design are summarised in Section 6.

## 2 Related Work

A plethora of methods to store routing tables and packet filters efficiently has been studied. We refer to the survey [2] for routing table representations.

Packet filters can be understood as “cutting”  $k$ -dimensional space, classifying packets taken as points in the space. Grid-of-tries [3], Hierarchical Intelligent Cuttings [4], Fat Inverted Segment trees [5], and HyperCuts [6] are typical examples of cutting algorithms.

Decision diagrams are graphs searching through a series of tests. Binary Decision Diagrams (BDD) [7] use a single boolean variable at each node. [8, 9] give a method to convert a packet filter into a BDD. [10] describe an approach to convert BDD filter representation into a circuit design for programmable chips (FPGA). Filters can be also converted into Interval Decision Diagrams [11] with integer variables in nodes [12].

## 3 RA tables and FDDs

Routing-ARP (RA) table is a combination of routing table and L3-to-L2 translation table (called ARP for brevity) into a single structure. RA table construction and properties are described in depth in [13], we give a short summary here.

RA table  $RA$  consists of records  $RA_i$  for  $1 \leq i \leq \text{Size}(RA)$ . Each record contains an IP prefix; its outcome is either an L2 address and interface of the next hop or symbol  $SW$  denoting the packet cannot be processed by the accelerator and must be sent to the host operating system. We suppose the table contains the default record, i.e., the void prefix. The table is searched for the first matching record for a particular address. The records are sorted in non-increasing prefix lengths and prefixes of the same lengths are disjoint. With those properties, RA table can be stored into a trie structure searched for the longest matching prefix. On the opposite, dumping a trie in non-increasing prefix lengths, RA table is obtained. We will use both approaches to describe the structures interchangeably.

Packet filter  $F$  is a list of rules denoted by  $F_i$ . Each rule contains a condition over packet header fields and an action. The first matching rule is used for a packet (see [14] for a discussion that this is general enough to model the real world). We require filters to be total, i.e., to decide how to handle any possible packet.

In [14], we have developed a representation of packet filters called Filtering Decision Diagram (FDD). By *FDD variables* we understand all possible terms

```

for  $i = 1$  to Size( $F$ ) do
  if rule  $F_i$  contains daddr
    then
      let  $p$  be the address prefix from  $F_i$  daddr
      let  $f$  be  $F_i$  with daddr removed
    else
      let  $p$  be the default route (i.e., the RA trie root)
       $f = F_i$ 
  fi
   $f' = \text{FDDConvertRule}(f)$ 
  PrepareSubtrie( $p$ )
  ApplyRuleToSubtrie( $f', p$ )
done

```

**Fig. 1.** Main loop of RAF computation

of the filtering language, e.g., `dst port 1024-65535` or `src addr 1.2.3.0/24`. *Class of FDD variables* is a set of variables testing the same header field. Note that `dport 1024-65535` and `dport 1025-65535` are distinct variables, they only belong to the same class. *FDD* is a multi-terminal binary decision diagram [15] over FDD variables; its terminals are actions of the filter. Paper [14] gives procedures to build FDDs, an algorithm to convert a filter to its FDD representation, and discusses implementation and complexity.

## 4 Combining RA and Filtering

To combine routing-ARP structure with filtering, the basic idea is to *distribute the packet filter into the RA table* according to destination address space specified in filtering rules. The essential observation is the following. Let  $F$  be a packet filter. For a destination address  $p \in IP$ , let  $F'$  be the filter containing only rules from  $F$  that affect the destination address  $p$  (i.e., filters specifying disjoint address spaces are omitted). Then filters  $F$  and  $F'$  give identical results for packets destined in  $p$ .

Although maintaining an FDD representation of the filter *for each prefix* in the RA table may seem as an extreme waste of memory, we argue that all the FDDs are stored in a single hash table and common parts of them are shared. For a routing-ARP record  $RA_i$ , we denote the connected FDD by  $\text{FDD}(RA_i)$ .

To distribute a packet filter  $F$  to a routing-ARP table  $RA$ , we start with all  $\text{FDD}(RA_i)$  values undefined. The main computation loop of applying the filter on the RA table is shown in Figure 1. The algorithm first determines the destination address space of the filtering rule (denoted by **daddr**) and removes it from the rule. Function  $\text{FDDConvertRule}(f)$  creates an FDD representation of the rule [14].

Function  $\text{PrepareSubtrie}(p)$  searches for the prefix  $p$  in the RA trie. If  $p$  is present in the RA trie, no changes are necessary. Otherwise, let  $p'$  be the longest prefix of  $p$  that is present in the trie. Then a new entry for  $p$  is created in the

trie and the outcome from  $p'$  is copied into the  $p$  entry. Such  $p'$  always exists as we require the RA table to contain at least the default RA record. Meaning of the RA table will not be changed by this modification [16].

Finally, function `ApplyRuleToSubtrie( $f'$ ,  $p$ )` traverses the whole subtrie denoted by prefix  $p$  (and including the prefix) and appends the rule  $f$ —calling `FDDAppend( $u$ ,  $f'$ )` [14] for the filter  $u$  that was originally in the trie—to all filters associated with prefixes of the subtrie. The resulting structure is called Routing-ARP-Filtering (RAF) table.

From implementation point of view, it is likely that a large number of filters in subtrees is identical. We can use dynamic programming: a cache of results of applying rule  $f$  to filters from the subtrie, avoiding to recompute them.

## 5 Rewriting the RAF Structure to LUP

We will describe a procedure to rewrite the obtained RA/FDD representation into a search that is partly performed by a first-match structure—CAM, and by comparison instructions.

We suppose following properties of the target hardware:

- The initial RA trie (i.e., address width) fits into CAM. (This is reasonable as commercially available CAMs are capable to hold the addresses.)
- A list *CAMList* prescribes *classes of variables* tested by columns of the CAM. We refer to fields of the list as *CAMList<sub>i</sub>*. An example of the *CAMList* may be `src addr, src iface, proto`.
- The lookup instructions may access any field of the headers. The order of testing header fields by the instructions is arbitrary.

We describe a *CAM row* by the *CAMRow* symbol. Technically, *CAMRow* is a ternary string of values 0, 1, and don't-care. It contains destination address prefixes and values of header fields prescribed by *CAMList*. Resulting values are attached to the *CAMRow*: MAC address of the next hop, interface to reach the next hop, and FDD representation of the filter.

### 5.1 Rewriting the RA Part

The conversion starts dumping the RA records in non-increasing prefix lengths into CAM (Figure 2) and expanding the decision tree “as much as possible.” The FDD associated with the CAM row represents the rest of the filter to be performed to resolve packets matching the row. Function `LUPInsertCAMFilter( $i$ , CAMRow)` is described in the following section.

### 5.2 Converting FDD to First-Match CAM

Function `LUPInsertCAMFilter( $i$ , CAMRow)` (Figure 3) expands variables belonging to classes prescribed by *CAMList* into a first-match structure and connects the lines of the structure to appropriately reduced FDDs that finish the

```

initialise CAMRow to don't-care values
/* traverse the RA trie in non-increasing prefix lengths */
for i = 1 to Size(RA) do
    insert RAi prefix into CAMRow
    insert RAi outcome (next hop MAC and interface) into CAMRow
    FDD(CAMRow) = FDD(RAi)
    LUPInsertCAMFilter(0, CAMRow)
done

```

**Fig. 2.** Converting RAF structure into LUP

```

function LUPInsertCAMFilter(i, CAMRow)
    i = i + 1
    if i ≤ Size(CAMList) then /* next CAMList field shall be prepared */
        choose an FDD variable j belonging to CAMListi
        if no such j exists in FDD(CAMRow) then
            LUPInsertCAMFilter(i, CAMRow)
        else /* prepare CAM rows for both possible j values */
            CAMRowh = CAMRowl = CAMRow /* incl. output values */
            FDD(CAMRowh) = FDDRestrict(FDD(CAMRow), j, high)
            FDD(CAMRowl) = FDDRestrict(FDD(CAMRow), j, low)
            insert value of j into CAMRowh /* keep don't-care values in CAMRowl */
            /* recursion: insert subsequent fields */
            LUPInsertCAMFilter(i, CAMRowh)
            LUPInsertCAMFilter(i - 1, CAMRowl)
        fi
    else /* terminal cases */
        write CAM row CAMRow to the output
    fi
end function

```

**Fig. 3.** LUPInsertCAMFilter(*i*, *CAMRow*): placing FDD into CAM

classification. Parameter *i* is the index of so-far-processed *CAMList* entries, *CAMRow* is the content of currently processed CAM row. The algorithm emits CAM rows one-by-one in first-match order.

In Figure 3, we first check if the CAM row has been finished. Then, we just write the *CAMRow* to the output. If it is not the case, we continue the recursion splitting the search on a suitable variable if possible. We will discuss this part of the algorithm in detail.

- First, we try to find a variable that can be resolved in currently processed CAM column (i.e., belonging to *CAMList<sub>i</sub>* class). We should choose a variable that really appears in the FDD(*CAMRow*) in order to allow the CAM row to be split into two (hence not wasting the CAM space). We may, e.g., traverse the FDD until we find a variable belonging to *CAMList<sub>i</sub>* class.
- If no such variable is found, we keep don't-care values in appropriate positions of the CAM row and continue to the next *CAMList* field.

- If a suitable variable  $j$  belonging to  $CAMList_i$  has been found, we prepare CAM rows for both values of  $j$ ;  $CAMRow^h$  will be used for the case that variable  $j$  is satisfied and  $CAMRow^l$  for the opposite. Appropriate restrictions of the FDD are attached to the rows. Intuitively, the  $FDDRestrict(u, j, v)$  function prunes the FDD  $u$  based on the fact that value of variable  $j$  is known to be  $v$ . It is similar to usual BDD restriction, moreover semantical relationship among variables is taken into account. See [14] for technical details.

We insert the test of variable  $j$  into appropriate position of  $CAMRow^h$ . Don't-care symbols stay unchanged in  $CAMRow^l$ . Finally, we continue the recursion for the rows split depending on value of variable  $j$ .

In the case of  $CAMRow^l$ , the block originating from this expansion is evaluated when variable  $j$  does not hold. We can use the same CAM column to insert another possible variable from the same class to test, therefore we call the recursion for  $i - 1$ . Note that when no suitable variable can be found, don't-care symbols will be preserved here.

This algorithm is also a general method to rewrite a decision diagram into a first-match hardware unit.

A detailed proof that the RAF table classification result is the same as with the source tables can be found in [16].

### 5.3 Query Types in CAM

We presented Algorithm in Fig. 3 hiding the problem that various types of queries differ in price paid in CAM. CAM is suitable to perform exact match and prefix match queries, a test of this type can be performed by a single CAM row, therefore the algorithm works exactly as we presented it.

On the other hand, range queries must be converted into prefixes, i.e., expanded into several CAM rows [17, 18]. We can understand creating CAM rows in Algorithm 3 as creating *blocks* of CAM rows that cover the range with prefixes and all the rows in the block are handled as a single unit in the algorithm.

### 5.4 Creating Lookup Instructions

Each CAM row we produced in Section 5.2 has an associated FDD that finishes classification of packets matched there. The tests in the FDD can be directly rewritten into SRAM instructions. We store all FDDs in a single structure—a hash table keeping their nodes. We can rewrite the hash table into SRAM instructions. For each CAM row  $CAMRow$ , we only arrange the computation to jump to appropriate root of  $FDD(CAMRow)$ .

This approach does not insert unreachable parts of the structure into the instructions. To show this, we have to indicate how node storage is implemented. FDD nodes are stored in a hash table and each node has an associated value denoting “how many times” the node is used in all structures represented by the hash table. We only have to delete FDDs that are not used anymore. This

is done decreasing the usage counter of appropriate nodes. When the counter reaches zero, the node can be removed. In Algorithm Fig. 3, this situation occurs when the restricted FDDs for  $CAMRow^h$  and  $CAMRow^l$  are computed. Then the FDD associated with the original  $CAMRow$  can be deleted, so parts of the structure that became unreachable are removed.

## 6 Conclusion

We have described a method to convert packet classification setting performed in discrete steps into a hardware lookup device used in the design of an accelerated router. The method is based on structures combining routing and level-3-to-level-2 addressing (RA tables) and representation of packet filters by Filtering Decision Diagrams. The structures are combined by distributing the filters to relevant portions of destination address space. The result is then transformed the lookup engine employing first-match Content Addressable Memory accompanied by comparison instructions that finish the search. A special case of this algorithm covers a general method to convert a decision diagram into a first-match structure.

One of the most beautiful properties of the algorithm is adaptability. Expansion of each CAM row can be regulated separately preserving correctness of the whole system, e.g., preferring more frequently used entries over less used ones. It can adapt to limited hardware resources, using the operating system to handle part of the traffic.

The method has been implemented [19] and its performance has been evaluated [16]. Experiments confirm that the method is extremely sensitive to variable class ordering. In the future, variable ordering together with adaptation mechanisms considering limited resources and current traffic to obtain optimal performance shall be studied.

## Acknowledgements

This project has been kindly supported by research intents “Optical Network of National Research and Its New Applications” (MŠM 6383917201) and “Parallel and Distributed Systems” (MŠM 0021622419).

## References

1. Novotný, J., Fučík, O., Antoš, D.: Project of IPv6 Router with FPGA Hardware Accelerator. In Cheung, P.Y., Constantinides, G.A., de Sousa, J.T., eds.: Field-Programmable Logic and Applications, 13th International Conference FPL 2003. Volume 2778., Springer Verlag (2003) 964–967
2. Antoš, D.: Overview of Data Structures in IP Lookups. Technical Report 9/2002, CESNET (2002)

3. Gupta, P., McKeown, N.: Packet classification on multiple fields. In: Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication, Cambridge, Massachusetts, United States, ACM Press, New York, NY, USA (1999) 147–160 ISBN 1-58113-135-6.
4. Gupta, P., McKeown, N.: Classifying packets with hierarchical intelligent cuttings. *IEEE Micro* **20**(1) (2000) 34–41
5. Feldmann, A., Muthukrishnan, S.: Tradeoffs for Packet Classification. In: Proceedings of INFOCOM. Volume 3., IEEE (2000) 1193–1202
6. Singh, S., Baboescu, F., Varghese, G., Wang, J.: Packet Classification Using Multidimensional Cutting. In: SIGCOMM'03: Proceedings of the Applications, Technologies, Architectures, and Protocols for Computer Communication Conference, Karlsruhe, Germany, ACM Press, New York, NY, USA (2003) 213–224
7. Bryant, R.E.: Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers* **35**(8) (1986) 677–691 ISSN 0018-9340.
8. Hazelhurst, S., Fatti, A., Henwood, A.: Binary Decision Diagram Representations of Firewall and Router Access Lists. Technical Report TR-Wits-CS-1998-3, University of the Witwatersrand, Johannesburg, South Africa (1998)
9. Hazelhurst, S., Attar, A., Sinnappan, R.: Algorithms for Improving the Dependability of Firewall and Filter Rule Lists. In: DSN '00: Proceedings of the 2000 International Conference on Dependable Systems and Networks (formerly FTCS-30 and DCCA-8), Washington, DC, USA, IEEE Computer Society (2000) 576–585 ISBN 0-7695-0707-7.
10. Sinnappan, R.A.: A Reconfigurable Approach to TCP/IP Packet Filtering. Master's thesis, Faculty of Science, University of the Witwatersrand, Johannesburg (2001)
11. Christiansen, M., Fleury, E.: An Interval Decision Diagram Based Firewall. In: Proceedings of 3rd IEEE International Conference on Networking (ICN '04), Gosier, Guadeloupe, French Caribbean, University of Haute Alsace, Colmar, France (2004) ISBN 0-86341-325-0.
12. Strehl, K., Thiele, L.: Symbolic model checking of process networks using interval diagram techniques. In: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design, San Jose, California, United States, ACM Press, New York, NY, USA (1998) 686–692 ISBN 1-58113-008-2.
13. Antoš, D., Řehák, V.: Routing and Level 2 Addressing in a Hardware Accelerator for Network Applications. In: ICT 2006, 13th International Conference on Telecommunications, Funchal, Madeira, University of Aveiro, Portugal (2006) 1–4 ISBN 972-98368-4-1.
14. Antoš, D., Řehák, V., Holub, P.: Packet Filtering for FPGA-Based Routing Accelerator. In: CESNET Conference 2006 Proceedings, Prague, Czech Republic, CESNET, z. s. p. o. (2006) 161–173 ISBN 80-239-6533-6.
15. Baier, C., Clarke, E.: The Algebraic Mu-Calculus and MTBDDs. In: 5th Workshop on Logic, Language, Information and Computation (WoLLIC'98). (1998) 27–38
16. Antoš, D.: Hardware-constrained Packet Classification. PhD thesis, Faculty of Informatics, Masaryk University, Brno, Czech Republic (2006) Submitted.
17. Spitznagel, E., Taylor, D., Turner, J.: Packet Classification Using Extended TCAMs. In: Proceedings of ICNP. (2003)
18. Taylor, D.E., Spitznagel, E.W.: On using content addressable memory for packet classification. Technical Report WUCSE-2005-9, Washington University in St. Louis (2005)
19. Ševčík, J.: Generator of Lookup Structures for Hardware-accelerated Routing and Packet Filtering. Bc. thesis, Faculty of Informatics, Masaryk University (2006)