Fast Routing Table Lookup Using CAMs

Anthony J. McAuley & Paul Francis[1]

Bellcore, 445 South Street, Morristown, NJ 07962-1910, USA

(mcauley@bellcore.com tsuchiya@bellcore.com)

Abstract

This paper investigates fast routing table lookup techniques, where the table is composed of hierarchical addresses such as those found in a national telephone network. The hierachical addresses provide important benefits in large networks; but existing fast routing table lookup technique, based on hardware such as Content Addressable Memory (CAM), work only with flat addresses. We present several fast routing table lookup solutions for hierarchical address based on binary- and ternary-CAMs and analyze the pros and cons of each.

1 Introduction

The central function of any communications switch is to route a call or packet to the appropriate destination. Simply put, this involves searching a routing table—a table composed of <*address, associated information*> entries—for the information needed to route the packet to the appropriate output port (or ports, in the case of multi-cast addresses). An entry in the routing table corresponding to a certain *address* tells the switch some *associated information* for deciding how to route the packet.

Ideally, routing table lookup should complete in the time it takes to read the packet off the link, or if cut-through switching is being performed (where the head of the packet is routed out before the tail arrives), the time it takes to read the address off the incoming link. As bandwidth and switching speeds increase, the time allotted to do the lookup decreases to the point where a software/RAM-based approach is not fast enough. A hardware structure called tries has been suggested [1]; but, for many lookup applications, it can be inefficient in memory or too slow. Exploiting the inherent parallelism of Content Addressable Memory (CAM) is attractive; but existing solutions [2] [3] [4] are limited to non-hierarchical addresses. This paper describes CAM-based architectures that provide low latency over a wide variety of address structures.

Sections 2 and 3 review the lookup function and address types. Section 4 categorizes existing lookup algorithms based on various memory hardware. Sections 5 and 6 describe six methods of using binary- and ternary-CAMs for routing table lookup. We conclude with a table for deciding between the solutions based on the address size and format.

2 The Lookup Function

This section describes the routing table lookup function that is executed every time a packet arrives at a switch.

2.1: Simplified Lookup Function

If the switch is connection-less (e.g. an IP network), then each packet header contains the complete addressing information needed to do the lookup function.

If the switch is connection-oriented (e.g. an ATM network), then most data packets need contain only a connection identifier. The complete addressing information (e.g. E.164 or IP address) is contained only in the first packet (e.g. SMDS) or in a special packet carried on the signalling channel (e.g. basic ATM). The connection identifier can be assigned by the sender or the switch; but, in either case, the connection identifier is a mnemonic representing all the address information.

The purpose of the connection identifier is increased efficiency. The connection identifier comes from a small number space compared to the address information. As a result, most of the packets of a connection do not need to carry all the addressing information, thus shrinking packet size. Further, the lookup function is simplified, because the size of the search field is reduced.

The connection identifier lookup function is a simplified case of the more general routing table lookup function. As such, any solution to the more general routing table lookup function, with its sparse address space, can be applied to the connection identifier lookup function (although it may not be the most cost-effective solution). We therefore focuses on the general routing table lookup function.

1. Recently published under the name of Paul Tsuchiya.

2.2: The General Routing Table Lookup Function

Up to this point, we have oversimplified by describing the input to the lookup function as being a simple address. However, the input can be both source and destination address, and other information that can collectively be called Quality-of-Service (QOS) information. QOS includes such *implicit* path information as low cost, low delay, high throughput, low error rate, and so on. This information is combined with the address for the lookup function. QOS information may also be *explicit*, such as an long-distance Inter-exchange Carrier (IEC) indication. In this case, the QOS information may take priority over the addressing information altogether, for instance because a switch is only concerned about getting the packet to the appropriate IEC. This priority effect also exists with hierarchically structured addresses. Here, one part of an address takes priority over another part in the routing decision.

For the remainder of this paper, we consider an *address* to potentially include both source and destination address and the QOS information described above. The way the addresses are assigned in a network affect the routing table lookup function; therefore we next look at three general address structures. This covers the full range of address types that one is likely to see in practice. In particular, it covers a wider range than previous work on hardware-assisted routing table lookup [1].

3 Three Types of Address

This section considers two addressing structures (flat and hierarchical) and two hierarchical address assignment algorithms (fixed-position and variable-position).

3.1: Flat Addresses

The simplest addressing structure is a flat address space, where we simply assign each destination a unique address chosen anywhere from the address space. This is seen, for example, in ethernet addresses and for local connection identifiers. This method has the advantage of simplicity; but is limited to small networks, where routing table size is manageable.

3.2: Fixed-position Hierarchical Addresses

Hierarchical addresses, such as those in the telephone system, greatly reduce routing table size. Figure 1 shows a small example, representing a subset of seven nodes (ovals) visible to a switch (not shown) with three levels of hierarchy. The telephone numbers are the standard US 10-digit code: where the "X" digit is a wild card, meaning that any digit matches that position. Table 1 shows the corresponding routing table. This routing table has addresses with four different fields: one 10-digit, two 6-digit, three 3-digit, and one 0-digit. For example, the switch has direct connections to two 6-digit switches that handle the 201-829 (via Port B) and 201-876 (via Port C) exchanges. The final entry is a 0-digit default switch that routes to everything else (via Port G).

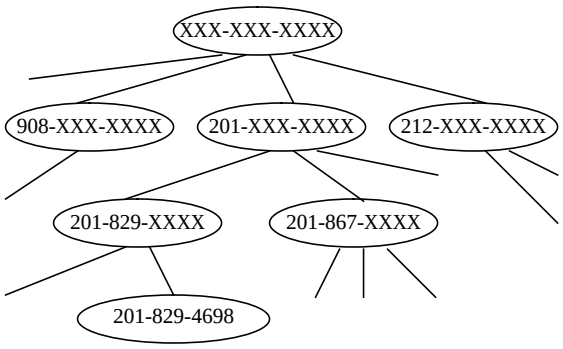

Figure 1 Fixed-position address hierarchy (with decimal addresses)

Table 1
Hierarchical address example

Address (decimal)	Next Hop
201-829-4698	Port A
201-829-XXXX	Port B
201-876-XXXX	Port C
201-XXX-XXXX	Port D
212-XXX-XXXX	Port E
908-XXX-XXXX	Port F

Consider a packet with address 201-829-4484. It matches three of the entries (201-829-XXXX, 201-XXX-XXXX, and XXX-XXX-XXXX). In this case, the best match is the one that matches on the most non-wildcard digits; thus the 201-829-XXXX represents the best route (port B), and perhaps the only correct route. A packet with address 908-829-4698 would be forwarded over Port F. Here, even though the 829-4698 part of the address matches 7 digits in the first entry, *any* mis-matched digits (908 instead of 201) results in a mis-match.

We can model the general lookup function as follows. The routing table consists of a list of *address/mask* pairs, and the associated information that gets returned as a result of the lookup. For example, Table 2 shows how the entries in Table 1 would be stored. The input to the lookup function is the *packetAddress*. If the *size* of a mask is the number of 1 bits in the mask, the result of the lookup function is the associated information of the entry with the largest mask such that: mask&packetAddress = address, where & is the

Table 2
Address and mask for Table 1

Address (decimal)	Mask (hex.)
201-829-4698	FFF-FFF-FFFF
201-829-0000	FFF-FFF-0000
201-876-0000	FFF-FFF-0000
201-000-0000	FFF-000-0000
212-000-0000	FFF-000-0000
908-000-0000	FFF-000-0000
000-000-0000	000-000-0000

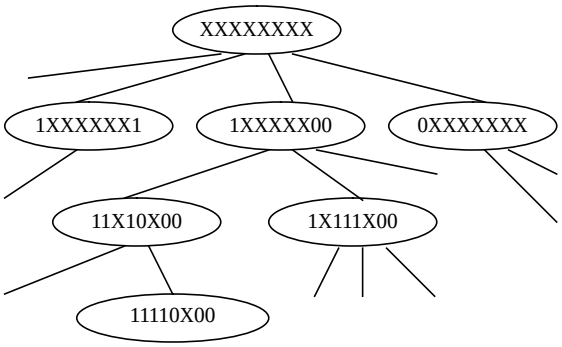

Figure 2 Variable-position address hierarchy (with binary addresses)

bitwise logical AND function. The flat routing table lookup, then, is a special case of the hierachical lookup function where the masks are all 1's.

Unlike the above example, just because two addresses are at some level *i* of the addressing hierarchy does not mean that they have the same mask (as it does with for instance telephone addresses in the USA). For instance, subnet numbers in IP addresses can fall on arbitrary bit boundaries [5]. Notice also that the 1's in the mask need not be contiguous. Indeed, there are known address assignment algorithms that can efficiently utilize the address space by taking advantage of variable-position (and in some cases non-contiguous) masks. Examples of this are kampai addressing [6], and variable-position subnet number assignment [7].

3.3: Variable-position Hierarchical Addresses

Figure 2 shows a view of a network similar to that used in the example of Figure 1, with seven areas (ovals) and three levels of hierarchy, but employing variable-position addresses (numbers inside the ovals). An area has the same locality for routing as the example of Figure 1.

Table 3
Variable-position addressing example

Address (binary)	Next Hop
11110X00	Port A
11X10X00	Port B
1X111X00	Port C
1XXXXX00	Port D
0XXXXXXX	Port E
1XXXXXX1	Port F
XXXXXXXX	Port G

Table 4
Address and mask for Table 3

Address (binary)	Mask (binary)
11110000	11111011
11010000	11011011
10111000	10111011
10000000	10000011
00000000	10000000
10000001	10000001
00000000	00000000

The numbers inside the ovals represent the addresses that switch has control over: that is, the numbers which are below it in the hierarchy. For example, the bottom oval in Figure 2 (11110X00) has two addresses associated with it: 11110000 and 11110100. The oval above this one (11X10X00) has four addresses associated with it: 11010000, 11010100, 11110000, and 1111100. From this small example, we can see that the addresses are much less rigidly structured than in the telephone example. Each area has a number of addresses equal to some power of 2.

If a switch has access to the seven areas shown in Figure 2, it will have a variable-position addressing table like that shown in Table 3. Table 4 shows the address and mask derived from Table 3. The lookup operation is the same as that already described for fixed-position hierarchical addresses (matching entry with largest mask).

For the routing table lookup function, there are important differences between fixed-position hierarchical addressing and variable-position hierachical addressing. The main difference with respect to this paper is that there are potentially many more mask combinations that must be considered in the lookup function for variable-position addressing. The consequences of this will be brought out

later in the paper. (The other major difference is that variable-position masks are not necessarily contiguous. However, this difference only affects software-driven routing table lookup, and so is not a factor in this paper.)

4 Routing Table Lookup

This section briefly reviews the general approaches to the lookup, using RAM, binary-CAM and ternary-CAM.

4.1: RAM-based Lookup

The RAM has two major operations:

- Write an entry into a specific address
- Read an entry by its address.

There are a number of (software) algorithms used to perform the lookup function using standard Random Access Memory (RAM).

A RAM can be perform the lookup in a single cycle if the data being searched (i.e. the packet address) is used as a direct index (RAM address) into memory. In this case, the size of a RAM is determined by the size of the search field. For example, with a 16-bit search field the RAM size is 64K (2^{16}) words. The number of words stored in a RAM has no effect on this size and cost. Thus, if there were only 256 words, each with a 16-bit search field, the RAM must still have 64K words. The size and cost of the RAM when used as a direct index grows exponentially with the search field. With current RAM technology trends, a 24 bit search pattern is the practical limit of an economic RAM-based search engine.

A linear search is the most efficient algorithm for table lookup, requiring only one entry per active address. If the entries in the routing table are searched in order of largest mask first, then the first match will be the best match. Of course, the linear search runs in time O(N), where N is the number of entries, and so can take considerable time.

A faster approach is to form a tree search: using a binary tree, a patricia tree, a trie tree, and so on [8]. In general, these trees can push the search time towards log$N$. In the case of the binary and patricia trees, the log base is 2. The log base can be increased, thus reducing search time, but for sparsely populated address spaces, this can result in unacceptable memory requirements [1]. Furthermore, even this search time may be excessive. For a high speed switch, the allotted search time can be measured in a few tens of instructions or less.

Under good conditions, a hash function can execute the lookup function in constant time [8], only slightly slower than direct access. The worst case search time, however, can be considerably worse than that of the tree searches. The performance is a function of the size of the hash memory and the number of addresses that must be searched in a given time window (after which a hashed entry will be timed out). While the routing table, because of hierarchical addresses, might be relatively small (10's or 100's of entries), the number of addresses that might potentially be searched can be large. This number depends on user traffic patterns, that can be hard to predict, especially for emerging data networks. Therefore, the amount of memory or the search time needed for hashing might be unacceptably large.

4.2: Binary-CAM-based Lookup

A CAM has three major operations:

- Write into the next free location
- Search for a word match.
- Read matching entries.

Data may be transferred to or from an CAM without knowing the memory address[2] of the word [4]. Binary data is automatically written to the next free word. To read a word the user must first do a search operation. Then, if there are multiple matches, the CAM decides (based on some internal state) which matched word to read next. Reading is useful because a CAM word has two parts. The most important part is the *search-field*, which is the part of the word that is matched with the search pattern. This typically contains the addresses of the known destinations. The CAM word also contains a *return-field*, which is the information returned during a read. This contains either the related information or an index.

All the CAM lookup algorithms are similar to the parallel search used in a RAM; however, the size of a CAM needed for direct access is determined primarily by the number of words that require storage. The size of the search field only affects the number of bits a word requires. For example, with a 10-bit search field, 10 bits per word are required. Thus, if there were only 256 possible inputs, each with a 16-bit search field, the CAM must have 256 words - with each word being at least 16 bits. The size and cost of a CAM grows linearly with the size of the search field and the number of entries.

The simplest CAM application is filtering, because it does not require any returned information other than the existence of the address. For example, to implement the network address screening function in SMDS, a CAM can be loaded with a list of valid/invalid network addresses. When a packet arrives, its address is used to search the CAM. Only if the CAM flag indicates a match/no-match is the packet processed.

If the amount of associated information is small (e.g. an output port index number), the CAM word is can store the

2. Some CAMs require addresses [3]; but addressless-CAMs are preferred for networking applications, because they directly store related information and reduce overall complexity [4].

related information in full. This direct access is fast (requires a single CAM read) and has low complexity. However, because the size of the CAM word is limited (by cost), the associated information must be relatively small, currently the maximum economic size is around 100 bits.

If the amount of associated information is large (e.g. because a large lower layer encapsulation address is required, or because multiple outputs are listed in the case of multicast forwarding), the CAM word is unable to store the related information. It therefore stores a unique index. The index is read, just as with direct access; but now the index is used as an address to read a RAM. This **indirect access** requires both a CAM read and a RAM read. Indirect access is therefore slightly slower and more complex than direct access; but allows more associated information.

4.3: Ternary-CAM-based Lookup

A ternary-CAM [9] has the same three major operations as a binary-CAM; but, while a binary-CAM stores one of two states (0 and 1) in each memory location (i.e. in each bit of a word), a ternary-CAM stores one of three states in each memory location (and also allows the search pattern to be one of the three states). We represent the three states by: 0, 1, and X. A ternary-CAM stores a don't care condition in the extra state (X), effectively allowing each word its own personal mask register. For hierarchical addresses, the ternary-CAM allows the address and mask information to be combined in a single ternary word.

We will introduce algorithms to perform lookup using a ternary-CAM in Section 6.

4.4: Cost Comparison

As a first order approximation, we assume the cost per bit of a:

- Binary-CAM is ten times the cost of a RAM.
- Ternary-CAM is twice the cost of a binary-CAM.

Since the entries in general routing tables tend to be sparsely populated over the (network) address space, direct indexing of the packetAddress into RAM is prohibitive. Therefore, it is necessary to either use a slower, software-driven search of RAM, or go to a hardware-based approach such as CAM (or a hybrid). Often, the time of the software-driven RAM search is unacceptable. With higher speeds, at some point it is necessary to go with the faster and well-bounded search time of a CAM.

Thus, although approximately an order of magnitude more expensive in terms of hardware, CAM-lookup solutions can offer superior performance compared to even the most sophisticated RAM-based search algorithms (careful management of the scarce CAM resources can help reduce costs - see Appendix A).

5 Binary-CAM Lookup Algorithms

This section describes three binary-CAM lookup algorithms/architectures: B1, B2, and B3 (B1 and B2 are well known and are included only for completeness).

5.1: B1 (Single-Cycle Single-Logical CAM)

A binary-CAM is directly useful for flat address lookup (or lookup for a switch at a fixed position in the hierarchy).

The system loads the CAM words with the routing table: i.e. the address and the associated information or index. The system also loads the mask register so that only the address is matched during a search (the associated information is masked). After loading the address, associated information, and mask, the CAM is ready to perform routing table lookup.

When a packet arrives at the switch, the system extracts its packetAddress. This packetAddress is used to *search* the CAM. If there is a match, the system *reads* the associated information in the subsequent cycle. After initialization, the CAM returns the associated information or index in two cycles: *search-read* (CAM search and CAM read).

If a new entry needs to be loaded into the routing table, the system adds the entry into the next free location. If an old entry needs to be removed from the table, the system selectively deletes that entry.

Method B1 is fast and has low complexity; but is only suited to flat addressing applications, because it only has a single fixed mask and assumes there is only one match.

5.2: B2 (Multiple-Cycle Single-Logical CAM)

If it is necessary to use more than one mask register, such as in hierarchical addressing, we cannot use method B1. We must be able to use different masks and have multiple search operations. A binary-CAM using multiple cycles works well for fixed-position hierarchical address lookup.

The system loads the CAM words with the address and the associated information (or index); but does not fix the mask register.

When a packet arrives, the system extracts its packetAddress and begins multiple mask loading and search operations. In the first cycle, the system loads the mask register of the address furthest down in the hierarchy (see Figure 1): that is, with the largest (most 1's) mask in the routing table (see Table 2). Then it searches (using the packetAddress) the CAM for a match. If a match occurs, it reads the associated information or index. If no match occurs, the second cycle begins and the system loads the mask register with the mask containing the second most 1's. The system again searches for a match. If a match occurs, it reads the associated information or index. If no match occurs, a third cycle begins. The system continues

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.