CHAPTER 6

Performance and Analyses of Route Lookup Schemes

The route lookup schemes described in Chapter 5 were tested against practical routing tables from [98]. This chapter describes the performance of the schemes in comparison with other comparable schemes and also discusses the various performance related issues.

6.1 Performance of the Trie-Based Scheme

The address lookup part of the scheme was implemented in Verilog, since in an actual router this would be implemented in hardware. The generation of the SRAM and the DRAM data, which in a practical router would be performed in software, was written in C. A lookup can be performed every 64ns, using this scheme. The amount of memory consumed for different routing tables is shown in Table 6.1. For instance, the MaeEast routing table with over 23,000 entries takes around 25KB of SRAM to store the bit pattern and around 12MB of DRAM to store the next hop addresses. In a conventional trie implementation, around 25MB of DRAM memory (the second last entry in the table) would be required. The last entry in the table shows the amount of compaction that can be achieved in the on-chip SRAM. For all the routing tables around 1 byte of SRAM memory
Table 6.1: Memory Requirements for various Routing Tables

<table>
<thead>
<tr>
<th>Site</th>
<th>No of Entries</th>
<th>SRAM (KB)</th>
<th>DRAM (MB)</th>
<th>Trie Memory (MB)</th>
<th>Bytes/entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaeEast</td>
<td>23,113</td>
<td>24.4</td>
<td>11.43</td>
<td>24.28</td>
<td>1.08</td>
</tr>
<tr>
<td>MaeWest</td>
<td>35,752</td>
<td>34.75</td>
<td>16.32</td>
<td>34.683</td>
<td>1.99</td>
</tr>
<tr>
<td>PacBell</td>
<td>27,491</td>
<td>29.08</td>
<td>13.66</td>
<td>29.03</td>
<td>1.08</td>
</tr>
<tr>
<td>Paix</td>
<td>17,641</td>
<td>20.5</td>
<td>9.63</td>
<td>20.46</td>
<td>1.19</td>
</tr>
<tr>
<td>AADS</td>
<td>31,958</td>
<td>32.25</td>
<td>15.15</td>
<td>32.18</td>
<td>1.03</td>
</tr>
</tbody>
</table>

per entry in the routing table is required. This gives very good scalability which would be very important when routing tables become even larger in the future.

The overall compaction achieved in this scheme is much higher than other existing comparable schemes. The required SRAM is small enough (about 35KB for a routing database >30,000 entries) to easily fit on a chip. This is useful especially when moving to IPv6 where larger routing tables or multiple tables for different hierarchies would be used. The data in our case is compacted to around 1 byte for every entry in the routing table (for a 16-way trie). In comparison, the forwarding table by Degermark et al [82] uses 5-6 bytes per entry. The implementation by Huang et al [84] has an even larger forwarding table.

Also, the overall memory consumption (SRAM and DRAM) using this scheme is almost half that required in conventional implementations. The static instruction count for building the SRAM and the DRAM data from the trie is 170 and the total CPU time taken to build this is in the order of 100ms on a Sun Ultra 5 with a 333 MHz processor. Since most forwarding tables need to be updated only about once every second, building the entire database from scratch is not an issue.

The number of memory accesses in the implementation, are 8 SRAM accesses and 1 DRAM. The number of SRAM accesses can be reduced further by splitting the SRAM and performing a direct lookup on the first 16 bits. The number of accesses then would be 5 SRAM accesses and 1 DRAM access. This is easily pipelined so that the DRAM cycle time is the limiting factor. By implementing queues and multiple DRAMs in parallel, an even higher throughput can be obtained. In the current implementation with a single
DRAM, a lookup can be done every 64ns which gives over 15 million lookups per second. In a conventional implementation, the number of memory accesses that would be required are 8 DRAM accesses. DRAM accesses being quite expensive (60-65ns per random read/write as opposed to <10ns for SRAM) [55] the conventional implementation would be much slower than this scheme.

The amount of memory used in our scheme is more than the 3-4 MB typical of Patricia and basic binary schemes as in [1]. This is only because our scheme uses a 16-way trie in order to reduce the depth of the trie, and trie completion takes up extra memory. The advantages of using a 16-way trie is the reduction in depth, which leads to a smaller latency. There is more redundancy in the DRAM data as seen in Figure 5.14 but using extra off-chip DRAM memory in order to reduce DRAM accesses is a better alternative.

### 6.1.1 Analysis of the Scheme

The amount of memory consumption depends on various factors. This section discusses the sensitivity of various parameters to the total memory consumed and also gives a theoretical analysis of the memory consumed.

#### 6.1.1.1 Sensitivity of Performance to Degree of Trie

The overall performance of the forwarding engine, in terms of throughput can be kept constant by altering the hardware FSM. The amount of SRAM memory consumed decreases, mainly due to less wastage in the trie completion step. The amount of SRAM required for various degrees of the trie is shown in Table 6.2. Efficiency of memory consumption increases with decreasing degree of the trie, i.e. fewer bytes per entry are required to store the SRAM data. This is due to less memory wastage in trie completion. The total latency of the address lookup changes with the degree of the trie. This is due to the fact that for smaller degrees, more SRAM accesses have to be made to traverse the trie since the depth of the trie increases. For instance, a degree 2 trie would require 16 stages
Table 6.2: SRAM requirements for different degrees of the trie structure

<table>
<thead>
<tr>
<th>Site</th>
<th>No of Entries</th>
<th>Degree=16 KB(B/entry)</th>
<th>Degree=8 KB(B/entry)</th>
<th>Degree=4 KB(B/entry)</th>
<th>Degree=2 KB(B/entry)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MaeEast</td>
<td>23,113</td>
<td>24.4(1.08)</td>
<td>16(0.71)</td>
<td>8.09(0.36)</td>
<td>6.57(0.29)</td>
</tr>
<tr>
<td>MaeWest</td>
<td>35,752</td>
<td>34.75(1.99)</td>
<td>23.03(0.66)</td>
<td>11.41(0.33)</td>
<td>9.23(0.26)</td>
</tr>
<tr>
<td>PacBell</td>
<td>27,491</td>
<td>29.08(1.08)</td>
<td>19.24(0.72)</td>
<td>9.76(0.36)</td>
<td>7.99(0.3)</td>
</tr>
<tr>
<td>Paix</td>
<td>17,641</td>
<td>20.5(1.19)</td>
<td>13.09(0.76)</td>
<td>6.86(0.4)</td>
<td>5.6(0.33)</td>
</tr>
<tr>
<td>AADS</td>
<td>31,958</td>
<td>32.25(1.03)</td>
<td>21.33(0.68)</td>
<td>10.67(0.34)</td>
<td>8.67(0.28)</td>
</tr>
</tbody>
</table>

Table 6.3: SRAM and DRAM memory consumption for different degrees of the trie structure

<table>
<thead>
<tr>
<th>Site</th>
<th>Degree=16</th>
<th>Degree=8</th>
<th>Degree=4</th>
<th>Degree=2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>SRAM (KB)</td>
<td>DRAM (MB)</td>
<td>SRAM (KB)</td>
<td>DRAM (MB)</td>
</tr>
<tr>
<td>MaeEast</td>
<td>24.4</td>
<td>11.43</td>
<td>16</td>
<td>3.87</td>
</tr>
<tr>
<td>MaeWest</td>
<td>34.75</td>
<td>16.32</td>
<td>23.03</td>
<td>5.58</td>
</tr>
<tr>
<td>PacBell</td>
<td>29.08</td>
<td>13.66</td>
<td>19.24</td>
<td>4.66</td>
</tr>
<tr>
<td>Paix</td>
<td>20.5</td>
<td>9.63</td>
<td>13.09</td>
<td>3.17</td>
</tr>
<tr>
<td>AADS</td>
<td>32.25</td>
<td>15.15</td>
<td>21.33</td>
<td>5.17</td>
</tr>
</tbody>
</table>

to traverse down to the bottom of the trie structure. A more deeply pipelined FSM would be required to maintain the same throughput. Our implementation chose a 16-way trie in order to reduce the latency and to keep the hardware simple. This comes at a cost of higher memory consumption.

The amount of compaction from DRAM to SRAM decreases with decreasing degree of the tree as shown in Table 6.3. This is due to the fact that for a 16-way trie, information corresponding to a node with all 16 children is represented by a single bit in the SRAM. For a 4-way trie, information corresponding to only 4 children is represented by 1 bit. The SRAM data is about 500 times smaller than the DRAM data for a 16-way trie, 250 times for a 8-way trie, 125 times for a 4-way trie and about 64 times smaller for a 2-way trie.
### Table 6.4: Routing Table for computing lower bound

<table>
<thead>
<tr>
<th>Prefix</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>00*</td>
<td>1</td>
</tr>
<tr>
<td>01*</td>
<td>2</td>
</tr>
<tr>
<td>10*</td>
<td>3</td>
</tr>
<tr>
<td>11*</td>
<td>4</td>
</tr>
</tbody>
</table>

![Trie structure](image)

**Figure 6.1: Trie structure for the entries in Table 6.4**

### 6.1.1.2 Upper and Lower Bounds on the SRAM memory Required

The amount of compaction achieved in the SRAM in our implementation is around 1Byte/entry as shown in Table 6.1. The amount of compaction depends on the nature of the routing table and on the degree of the trie used. In this section, we determine the bounds on the compaction that can be achieved in order to give an idea of the possible range of compaction. Computing the bounds on the SRAM memory is easy. Every time a node is added to the trie structure, it results in the addition of an M-bit pattern (where M is the degree of the trie) in the SRAM. This fact can be used to compute the upper and lower bounds on the memory. To compute the lower bound, consider the entries shown in Table 6.4. For a 4-way implementation, the trie structure and the bit pattern would be as shown in Figure 6.1. The first entry results in the addition of a node, whereas subsequent entries fit into the created node till the node is complete. Therefore, M entries could be represented by an M bit pattern or in other words, 1 bit per entry is the minimum amount of memory required (the 1 corresponding to the root node can be ignored, as before).

To compute the upper bound, we consider the case when an entry results in the addition of as many nodes as possible. The maximum number of nodes that can be added
for any entry is equal to the depth of the trie. And since each node, results in an M-bit pattern the upper bound on the memory required is $D \cdot M$, where $D$ is the depth of the trie and $M$ is the degree of the trie. For example, consider the entry 0010 0010 ... 0010. This 32-bit entry results in the trie structure shown in Figure 6.2 and leads to 8 nodes being added to the 16-way trie. The amount of memory used in this example is 16 bytes/entry.

The SRAM memory therefore, lies between

1 bit/entry $\leq$ SRAM Memory $\leq$ $D \cdot M$ bits/entry

For the 16-way implementation of the paper, the bounds on the memory is given by:

1 bit/entry $\leq$ SRAM Memory $\leq$ 16 Bytes/entry

The upper and lower bounds correspond to extreme cases and are not representative of practical routing tables. The lower bound assumes that the trie is complete whereas the upper bound case assumes that all entries are 32 bits wide (for IPv4) and don’t share a common node along the path. This is hardly the case for practical routing tables which are usually sparse and share common nodes.

6.1.1.3 Expected SRAM Memory Required

The upper and lower limits on the SRAM memory correspond to pathological cases which don’t occur in real routing tables. This section determines the expected SRAM memory required, assuming completely random distribution of entries in the routing table.

**Theorem 1** The expected SRAM memory (bits/entry), for $n$ random uniformly distributed
entries, is given by

\[ E(\text{Mem(Bits/entry)}) = \frac{M}{\ln(M)} \]  

(6.1)

where \( M \) is the degree of the trie.

**Proof:** To prove the above equation, we start with the recurrence relation given in [70]. Let \( A_n \) be the average number of nodes in a random M-ary search trie that contains \( n \) keys. Then \( A_0 = A_1 = 0 \) and for \( n \geq 2 \),

\[
A_n = 1 + \sum_{k_1+\ldots+k_M=n} \frac{n!}{k_1!\ldots k_M!} (M^{-n})(A_{k_1} + \ldots + A_{k_M})
\]

(6.2)

\[
\frac{n!M^{-n}}{k_1!\ldots k_M!}
\]

is the probability that \( k_1 \) of the keys are in the first subtrie, \( \ldots k_M \) in the \( M^{th} \). Using symmetry and summing over \( k_2 \ldots k_M \), the above equation can be reduced to the recurrence relation given below:

\[
A_n = 1 + M^{1-n} \sum_{k=2}^{n} \binom{n}{k} (M-1)^{n-k} A_k
\]

(6.3)

The remainder of the proof does not appear as is in [70]. However, it closely follows the techniques and methods developed in [70] to solve recurrence relations and determine their asymptotic value. To determine the expected memory, we first solve for the general recurrence relation given by:

\[
A_n = C_n + M^{1-n} \sum_{k=2}^{n} \binom{n}{k} (M-1)^{n-k} A_k
\]

(6.4)

To solve the above relation, we take the binomial transform of \( A_n \), given by:

\[
\hat{A}_n = \sum_{k=2}^{n} \binom{n}{k} (-1)^k A_k
\]

(6.5)

Substituting for \( A_k \) from (6.4), we get
\[ \hat{A}_n = \sum_{k=2}^{n} \left( \begin{array}{c} n \\ k \end{array} \right) (-1)^k \left\{ C_n + M^{1-k} \sum_{l=2}^{k} \left( \begin{array}{c} k \\ l \end{array} \right) (M-1)^{k-l} A_l \right\} \]  

(6.6)

or,

\[ \hat{A}_n = \hat{C}_n + \sum_{k=2}^{n} \sum_{l=2}^{k} \left( \begin{array}{c} n \\ k \end{array} \right) (-1)^{k} M^{1-k} \left( \begin{array}{c} k \\ l \end{array} \right) (M-1)^{k-l} A_l \]  

(6.7)

However,

\[ \left( \begin{array}{c} n \\ k \end{array} \right) \left( \begin{array}{c} k \\ l \end{array} \right) = \frac{n!}{k! (n-k)! l! (k-l)!} \]

\[ = \frac{n!}{(n-l)! (n-k)! l! (k-l)!} \]

\[ = \left( \begin{array}{c} n \\ l \end{array} \right) \left( \begin{array}{c} n-l \\ k-l \end{array} \right) \]

(6.8)

Therefore, (6.7) can now be written as

\[ \hat{A}_n = \hat{C}_n + \sum_{k=2}^{n} \sum_{l=2}^{k} \left( \begin{array}{c} n \\ l \end{array} \right) \left( \begin{array}{c} n-l \\ k-l \end{array} \right) (-1)^{k} M^{1-k} (M-1)^{k-l} A_l \]  

(6.9)

Swapping the order of summation, (6.9) reduces to

\[ \hat{A}_n = \hat{C}_n + \sum_{l=2}^{n} \left( \begin{array}{c} n \\ l \end{array} \right) (-1)^{l} A_l \sum_{k=l}^{n} \left( \begin{array}{c} n-l \\ k-l \end{array} \right) (-1)^{k-l} M^{1-k} (M-1)^{k-l} \]  

(6.10)

Introducing a new variable \( j = k - l \), the above equation now reduces to

\[ \hat{A}_n = \hat{C}_n + \sum_{l=2}^{n} \left( \begin{array}{c} n \\ l \end{array} \right) (-1)^{l} A_l \sum_{j=0}^{n-l} \left( \begin{array}{c} n-l \\ j \end{array} \right) (-1)^{j} M^{1-j} (M-1)^{j} \]  

(6.11)
The second summation constitutes the binomial expansion of \((1 - \frac{M-1}{M})^{n-l}\) which is equal to \((1/m)^{n-l} = m^{l-n}\). Therefore, (6.11) reduces to the form

\[
\hat{A}_n = \hat{C}_n + \sum_{l=2}^{n} \binom{n}{l} (-1)^l M^{l-n} A_l
\]

\[
= \hat{C}_n + M^{l-n} \sum_{l=2}^{n} \binom{n}{l} (-1)^l A_l
\]

(6.12)

Since \(A_0 = A_1 = 0\), the summation in (6.12) represents \(\hat{A}_n\). The above equation then simplifies to

\[
\hat{A}_n = \left( \frac{M^{n-1}}{M^{n-1} - 1} \right) \hat{C}_n
\]

(6.13)

Taking the binomial transform again,

\[
A_n = \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{M^{k-1}}{M^{k-1} - 1} \hat{C}_k
\]

\[
= \sum_{k=2}^{n} \binom{n}{k} (-1)^k M^{k-1} + 1 - 1 \hat{C}_k
\]

\[
= C_n + \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{1}{M^{k-1} - 1}
\]

(6.14)

The equation is now in a solvable form, once \(\hat{C}_k\) is determined. \(C_k\) corresponds to the sequence \(0, 0, 1, \ldots, 1\) and the binomial transform of the sequence is easily shown to be \(k - 1\). (6.14) then finally reduces to

\[
A_n = C_n + \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{1}{M^{k-1} - 1} (k - 1)
\]

(6.15)

(6.15) gives the expected number of nodes in an M-ary trie storing \(n\) prefixes. Finally, what remains to be done is to compute the asymptotic value of (6.15). To do that, we split (6.15) and compute the asymptotic value of each term as shown below.
\[ A_n = C_n + \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{k}{M^k - 1} - \sum_{k=2}^{n} \binom{n}{k} (-1)^k \frac{1}{M^k - 1} \]

\[ = C_n + U_n + V_n \] (6.16)

\( V_n \) can be simplified as

\[ V_n = k = 2^n \binom{n}{k} (-1)^k \sum_{j \geq 1} \left( \frac{1}{M^{k-1}} \right)^j = \sum_{j \geq 1} (M^j(1 - M^{-j})^n - M^j + n) \] (6.17)

By further setting \( x = n/M^j \), \( V_n \) can be further reduced as shown in [99] to,

\[ V_n = \sum_{j \geq 1} n \left( e^{-x} - 1 + x \right) = n \sum_{j \geq 1} \frac{1}{x} \left( e^{-x} - 1 + x \right) \] (6.18)

\( U_n \) can be simplified similarly. The extra factor \( k \), changes the factorial term to give,

\[ U_{n+1} = \sum_{j \geq 1} (-1)(n + 1)(e^{-x} - 1) = (n + 1) \sum_{j \geq 1} (1 - e^{-x}) \] (6.19)

Next we use a couple of standard results from complex variable theory as described in [99].

\[ e^{-x} = \frac{1}{2\pi i} \int_{\gamma} \Gamma(z)x^{-z}dz = \frac{1}{2\pi} \int_{-\infty}^{\infty} \Gamma(1/2 + it)x^{(1/2+it)}dt \] (6.20)

This can be shown to be equal to the sum of residues, which is

\[ \sum_{0 \leq k < M} x^{-k} \frac{(-1)^k}{k!} \] (6.21)

Changing the limit of the integral in (6.20) is equivalent to the removing corresponding poles. The critical quantities in (6.18) and (6.19) can then be expressed as
\[
\frac{1}{2\pi i} \int_{-\frac{3}{2}-i\infty}^{-\frac{3}{2}+i\infty} \Gamma(z) x^{-z} dz = e^{-x} - 1 + x
\]
\[
\frac{-1}{2\pi i} \int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty} \Gamma(z) x^{-z} dz = 1 - e^{-x}
\]

(6.22)

(6.16) can then be written as (after placing the sum inside and simplifying)

\[
A_n = C_n + \frac{n}{2\pi i} \int_{-\frac{3}{2}-i\infty}^{-\frac{3}{2}+i\infty} \frac{\Gamma(z) n^{-1-z}}{M^{-1-z}} dz - \frac{n+1}{2\pi i} \int_{-\frac{1}{2}-i\infty}^{-\frac{1}{2}+i\infty} \frac{\Gamma(z) n^{-z}}{M^{-z}} dz
\]

(6.23)

Calculating the residues at the poles as shown in [99], the asymptotic value can be shown to be

\[
A_n = O(1) + \left( n\log n + \frac{n\gamma}{\ln(m)} - n/2 - nf_0(n-1) + O(1) \right)
- \left( n\log n + \frac{n\gamma}{\ln(m)} - n/2 + nf(n) + O(1) \right)
= \frac{n}{\ln(m)} + (\text{negligible terms}) + O(1)
\]

(6.24)

Since each node in the trie gives rise to M bits in the SRAM, the expected SRAM memory (ignoring the smaller terms) is

\[
E(\text{Mem(\text{Bits/entry})}) = \frac{M}{n} \frac{n}{\ln(M)} = \frac{M}{\ln(M)}
\]

(6.25)

From the above analysis, the expected SRAM memory can be calculated for the different routing tables. This is shown in Table 6.5. As before, the terms inside the bracket represents SRAM memory expressed in Bytes/entry. The expected value gives a reasonably good prediction of the SRAM memory requirement especially for lower


