IP Route Lookups as String Matching

Austin Donnelly and Tim Deegan
University of Cambridge
Computer Laboratory
Cambridge, CB2 3QG, U.K.
E-mail: {*Austin.Donnelly,Tim.Deegan*}*@cl.cam.ac.uk*
Tel: +44 1223 334600
Keywords: longest prefix matches, finite state automata

Abstract

An IP route lookup can be considered as a string matching problem on the destination address. Finite State Automata (FSA) are a flexible and efficient way to match strings. This paper describes how a routing table can be encoded as an FSA and how, through a process of state reduction, we can obtain an optimal representation. This gives insights into the basic properties of the longest-prefix match problem.

1. Introduction

Ever since the introduction of classless inter-domain routing (CIDR), researchers have worked to optimise the process of doing a longest prefix match on IP addresses to discover the subnet to which they belong.

Many finely-honed schemes exist, the trend being to design data structures which fit entirely in a modern workstation's caches [3], or algorithms which are easy to implement in hardware [5].

Many of these schemes make use of tries, a construction similar to a tree but where the search key is encoded in the path taken to reach a particular node, rather than in the nodes themselves. The trie data structure seems intuitively the correct representation to use, but no authors have actually proved that it is optimal. In this paper, we show how the trie data structure arises naturally from the minimisation of an FSA corresponding to a routing table. We do not offer a faster route lookup scheme — instead we show that current practice is the best possible one in a novel way.

This paper does not consider multi-dimensional range matching or layer-4 routing. Some of these schemes use tries as their basic building blocks [9], while others use completely different techniques [10].

In Section 2, we outline the basic properties of the longest prefix match problem; in Section 3.2 we show how it can be re-stated in terms of an FSA string matching problem. Section 4 covers how the states in an FSA can be minimised, leading to the much-used basic trie structure with route aggregation. Finally, Section 5 makes observations about the problem in general, and suggests further optimisations that may be possible.

2. Longest prefix matches

The basic IP route lookup problem can be stated as follows. We have a routing table R which maps each subnet to a forwarding equivalence class (FEC). An FEC defines how the router will handle a particular class of packets. It may specify the outgoing queue / interface, link-layer next hop details, etc. Given a packet with destination IP address a we need to find the subnet in R the address is within, and thus the FEC which dictates how the packet should be forwarded (if at all). By saying an IP address is *within* a subnet, we mean that the subnet is a prefix of the IP address. If R includes multiple subnets which are all prefixes of a, we select the longest subnet in order to disambiguate the match. This is why IP lookups are called *longest prefix matches*.

We can consider the subnets stored in a routing table to be strings of binary digits. The network number is the value of the string as a binary number, and the length of the string gives the netmask length. For example, the 16-bit subnet 128.232.0.0/16[1] would become the string 1000000011101000.

A trie can be used to store these subnet strings, allowing common prefixes to be factored out as much as possible. The edges of the trie are labelled with a member of the al-

[1] The number following the slash in the notation `a.b.c.d/n` gives the number of contiguous 1's in the netmask.

phabet the keys are over. As we use subnet strings as keys this means each edge is labelled either with a 1 or a 0.

Nodes in a trie are *marked* if the labels on the path from the root to that node form a valid key. Since we are storing a routing table in the trie, we note the FEC the subnet is mapped to in the node itself.

Figure 1 shows an example. Black nodes mark the end of a valid path from the root. The trie in question stores the prefixes 00, 010, 100, 1, and 111. Note that 1 is a prefix of 100, so the address 101 would be matched by the shorter prefix 1, while the address 100 would be matched by the longer 100 prefix.

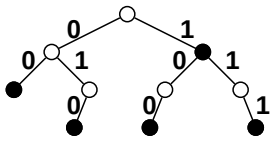

Figure 1. A trie storing 5 prefixes

If we are to distinguish between N subnets then there must be N marked nodes in the trie, and the minimum depth of the trie will be $O(\log(N))$ nodes. We expect that in actual fact the trie will be deeper since few internal nodes will be marked, and the trie will not be balanced.

To verify this assumption, a trie was built from data taken from the route-views project[2] on August 22 2000. It shows that the trie is in fact quite unbalanced, mainly due to the multicast address space (binary prefix 1110) being sparsely populated. The minimum and maximum path lengths in the trie were 8 and 32, corresponding to the shortest and longest netmasks encountered in the routing table. There were 66 routes to individual machines (i.e. with a netmask length of 32).

The full path length distribution is given in Figure 2. The path length is on the x-axis, and the number of occurrences in the route database is plotted on a log scale on the y-axis. From this, it can be seen that just over 65% of paths in the trie are either 24 or 16 edges long. Clearly, this is not a flat distribution of strings in the trie, so it should be possible to take advantage of this skew in the distribution in order to minimise the number of pairwise comparisons needed to be made. The authors of [10] used this distribution to their advantage by checking the hash tables for 16 and 24 first, before progressing onwards.

As a somewhat unrelated aside, looking back at route-views snapshots from a little over a year ago shows a general increase in the number of small networks (/18 to /24 in particular), and a slight decrease in the number of large networks (e.g. /8), presumably due to large networks being split to re-use the addresses in smaller netblocks. 24-bit networks are the fastest growing netmask length, with over 17000 new entries of that length appearing during the period from May 1999 to August 2000 – almost a factor of seven more than the next largest-growing size (/23).

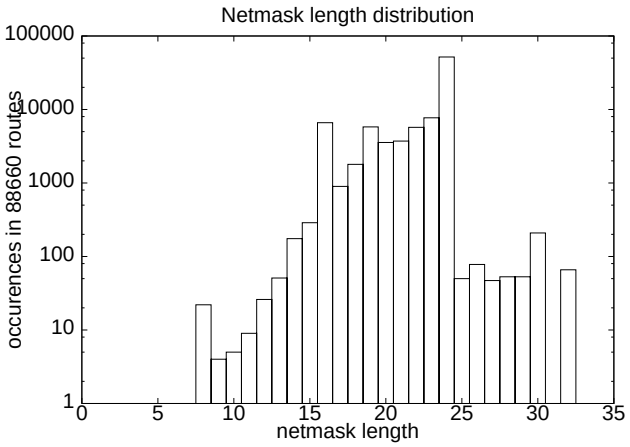

Figure 2. Netmask length distribution

3. Mapping to an FSA

The IP route lookup problem is similar to a few other problems. So far, it has been stated in terms of finding the longest possible match. It can also be re-stated as follows: subnets are intervals on the integers. IP addresses are integers. Doing a route lookup is finding the smallest interval a particular integer is within. This is the approach taken in [6], in which a lower bound of $\Omega(\log\log(N))$ pointer dereferences is proved. It should be noted that [6] deals with a simplified problem, since they do not allow intervals to enclose other intervals. The authors of [5] generalise this technique to multiple dimensions with arbitrary overlaps.

Mapping a faulting virtual address to a page table entry is also a similar problem, where the most specific matching entry should be used.

Another way of stating the problem is to map it onto the string matching problem. By creating an FSA from the routing table then minimising it, a comparison trie is built which includes optimisations specific to the particular routing table in question. The remainder of this paper shows how such automata may be constructed, and how their minimisations degenerate into binary tries.

3.1. Routing table representation

We introduce $\Sigma = \{0, 1\}$, the alphabet over which addresses and subnets are strings. F is the set of all FECs,

[2] `http://www.antc.uoregon.edu/route-views/`

We filtered out all routes from one ISP who was inadvertently advertising spurious prefixes. In particular, they claimed connectivity to 60 contiguous class A (/8) networks. The addresses in question are reserved by IANA, not owned by the ISP. Furthermore, they are unroutable at other major exchanges. We have contacted the ISP to determine the cause.

plus a special FEC ω used to denote the lack of routing information – packets "forwarded" according to policy ω are simply dropped, since they are unroutable. Our routing table R is defined as a set of 2-tuples of the form (ν, f), where $\nu \in \Sigma^\star$ is a subnet string, and $f \in F$ is the FEC associated with the subnet ν. R may contain a default route (ε, f_d), where ε is the empty string and f_d is the default FEC.

There are $n = |R|$ routing table entries, and for convenience we refer to the ith routing table entry as (ν_i, f_i), where $0 \leq i < n$. We define $\nu_i[j] \in \Sigma$ to be the jth character in the ith routing table entry's subnet string if $0 \leq i < n \wedge 0 \leq j <$ length(ν_i), and $\perp$ otherwise. Note that i and j count from 0, not 1.

3.2. FSA construction

A non-deterministic FSA can be represented as a 5-tuple (S, Σ, I, A, T), where S is the finite set of states, Σ the finite alphabet, $I \subseteq S$ the set of initial (or start) states, $A \subseteq S$ the set of accepting states, and $T \subseteq S \times \Sigma \times S$, the transition relation. Our FSAs include an extra property, $W \in A \times F$, a total function which maps every accepting state to a FEC in R.

The FSA is said to accept a string x if, starting in any state $i \in I$, it consumes successive symbols of x moving from state to state according to the transition relation and is left in a state $a \in A$ (i.e. an accepting state) with no further symbols of x remaining. If at any stage there is no possible transition for the input symbol, then the string x is normally considered rejected by the FSA.

For easy of reference, we name states $\text{st}_{i,j}$ where both i and j are cardinals. Clearly it is possible to encode names of this form as a single cardinal, for example $\text{st}_{i,j} = 2^i \times 3^j$. The non-deterministic FSA M corresponding to the routing table R is then composed of:

- the alphabet $\Sigma = \{0, 1\}$,
- the set of accepting states $A = \{\text{st}_{i,j} : (\nu_i, f_i) \in R \wedge j = \text{length}(\nu_i)\}$,
- the set of states $S = \{\text{st}_{i,j} : \nu_i[j] \neq \perp\} \cup A$,
- the set of initial states $I = \{\text{st}_{i,0} : (\nu_i, f_i) \in R\}$,
- the forwarding mapping $W = \{(a, f) : a \in A \wedge (\exists i, j : a = \text{st}_{i,j}) \wedge (\nu_i, f_i) \in R \wedge f_i = f\}$.
- and the transition relation $T = \{(q_s, c, q_d) : \exists i, j : q_s = \text{st}_{i,j} \in S \wedge c = \nu_i[j] \neq \perp \wedge q_d = \text{st}_{i,j+1} \in S\}$.

As an example, consider the routing table in Figure 3. This table contains five subnet prefixes and their associated FECs. Each individual bit is addressable as $\nu_i[j]$, so for example $\nu_0[0]$ has the value 0, while $\nu_2[0]$ is 1. Notice subnets 2, 3 and 4 are all forwarded according to the same policy: FEC 1. Later, we show how these routes may be aggregated into a single entry.

subnet 0 (**0 0**	, fec 2)
subnet 1 (**0 1 0**	, fec 0)
subnet 2 (**1**	, fec 1)
subnet 3 (**1 0 0**	, fec 1)
subnet 4 (**1 1 1**	, fec 1)

Figure 3. An example routing table

Every bit in the routing table becomes a transition into a fresh state in the FSA being constructed, as shown in Figure 4. Accepting states are shown with an extra circle around them; initial states have a wedge to their left. Note that this is a single non-deterministic FSA with multiple start states – symbol matching occurs in parallel.

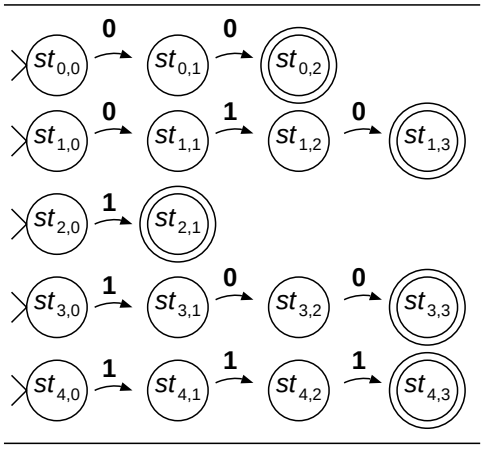

Figure 4. Initial FSA, M

This non-deterministic finite state machine can be used to do IP route lookups. The destination IP address of an incoming packet is matched from MSB to LSB and if an accepting state is reached, the packet has been successfully classified. Since there is one accepting state per subnet, the state can be mapped using W to the forwarding equivalence class the packet belongs to. Failed matches are rejected as unroutable.

There are 4 problems with the automaton as constructed:

P1 It is non-deterministic, and therefore must be determinised before it can be run.

P2 It matches the strings in the routing table exactly, however the strings are meant to be prefixes. We assume that every symbol in the input string must be consumed by the FSA in order to match.

P3 It does not "backtrack" on failure to find a potential previous match. In our example, this means it would reject the string 110, when it should accept it as being part of subnet 2.

P4 Because minimisation only preserves the language matched, accepting states may be merged together. This is a problem since we rely on knowing *which* accepting state caused an address to be matched in order to find the forwarding information from W.

The process of determinisation identifies states which are indistinguishable and merges them into a single new state. Two states q_0 and q_1 are indistinguishable if an input string leading from an initial state to q_0 could also lead from an initial state to q_1. In the example in Figure 4, states $st_{0,1}$ and $st_{1,1}$ could be merged, as could states $st_{2,1}$, $st_{3,1}$ and $st_{4,1}$. Since q_0 and q_1 may themselves be initial states, all initial states are equivalent, and may be merged. The result of determinising our example FSA is shown in Figure 5. A more formal definition of determinisation is given in Appendix A.

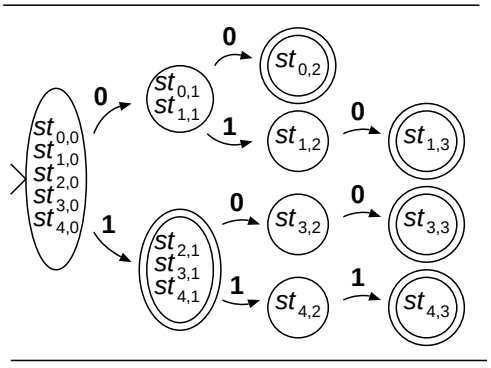

Figure 5. FSA after determinisation

P2 and P3 are really aspects of the same problem: the transition relation is not total. By ensuring that all states have a transition defined for each symbol in Σ we make the transition relation total, and there is no ambiguity concerning matches.

P4 may be solved by *protecting* the accepting states from being merged as described in Section 4.1. This can be done at the same time as the transition relation is made total.

4. Minimisation

In a determinised FSA each non-accepting state corresponds to a bit-compare operation, so in order to minimise the number of pairwise comparisons needed to classify a packet, the number of states in the FSA for a routing table should be minimised.

Minimisation of an FSA is a topic already covered amply in the literature [4], with [11] providing an especially clear categorisation of the different algorithms available.

There are two main ways of minimising an FSA: Brzozowski's algorithm [2] and those based on subset construction techniques, effectively calculating an equivalence relation between the original states and the minimised FSA's states such that states that are members of the same equivalence class are indistinguishable.

Any means of minimisation is acceptable, indeed, the entire reason for encoding IP route lookups as an FSA is specifically to reduce the IP route lookup problem to a previously solved one.

4.1. Protecting accepting states

Since minimisation preserves only the language accepted, we must encode the forwarding information present in W in the language itself. We do this by extending the alphabet to include extra symbols we call *labelling letters*. Each member f of F has an associated labelling letter l, which we write as $l = \mathcal{L}(f)$. We also add some extra states: α, which will eventually become the only accepting state in the automata; and a state for each $f \in F$ which "eats" extra 0 or 1 symbols, written $\mathcal{E}(f)$. These *eat-states* are needed to correctly match entries in the routing table as prefixes.

We will transform each accepting state into two states connected by the labelling letter appropriate to the old accepting state. The first state is the eat-state for the FEC in question, the second is always α. In effect, we are changing the language we want to accept from IP addresses to IP addresses each appended with the labelling letter appropriate to its FEC. By tagging the old accepting states with labelling letters, we only allow the merging of two paths through the FSA if they both lead to the same FEC (i.e. they are indistinguishable from a routing point of view).

More precisely, the eat-states are defined by extending the determinised M to produce M' as follows:

- $\Sigma' = \Sigma \cup \{\mathcal{L}(W(a)) : a \in A\}$,
- $S' = S \cup \{\alpha\} \cup \{\mathcal{E}(f) : f \in F\}$
- $I' = I$,
- $A' = A$,
- $W' = W$,
- $T' = T \cup T_{loop} \cup T_{label}$.

where $T_{loop} = \{(\mathcal{E}(W(a)), c, \mathcal{E}(W(a))) : a \in A \wedge c \in \{0,1\}\}$ and
$T_{label} = \{(\mathcal{E}(W(a)), \mathcal{L}(W(a)), \alpha) : a \in A\}$.

T_{loop} addresses problem P2 (match as prefixes) by adding transitions from all the eat-states looping back to

themselves on symbols 0 and 1. T_{label} adds a transition from each eat-state on its associated labelling letter to the α state, in order to ensure that the final symbol in every accepted string is a labelling letter.

To solve problem P3 (longest prefix match), we apply a depth-first walk of M' to generate a new FSA M'' with additional transitions which fully specify the behaviour of the automaton on failure to match. This depth-first walk is defined by the recursive algorithm given in Figure 6.

```
1:  totalise(f, q):
2:     if q ∈ A then
3:        f = W(q)
4:     endif
5:     add (q, L(f), α) to T
6:     for c in {0,1} do
7:        if ∃(q, c, q_d) ∈ T then
8:           totalise(f, q_d)
9:        else
10:          add (q, c, E(f)) to T
11:    done
```

Figure 6. Algorithm to extend transition relation

The `totalise` algorithm keeps the current best-matching FEC f which is used to fill in missing transitions. It also keeps track of q, the current state it is processing. The algorithm is started with q set to the start state of the FSA and f set to the special FEC ω used to denote an unroutable packet.

As the `totalise` algorithm walks the FSA graph, f is updated each time an accepting state is traversed, and because the walk is depth-first f is always the longest matching FEC. It is used in line 10 to add missing transitions out of the current state into the appropriate eat-state for the longest matching FEC, where extra 0s and 1s may be safely consumed before finally matching on the labelling letter. Line 5 ensures the match may terminate early.

After being processed by `totalise`, the FSA is changed so the only accepting state is α.

To summarise: the routing table R is encoded as an FSA by building the naïve non-deterministic FSA M as described in Section 3.2. M is then determinised, and the new states α and the eat-states added as specified above, giving M'. Finally, the `totalise` algorithm is run on the automaton, and α made the sole accepting state to produce M''. Figure 7 shows how our example looks at this stage. This (ugly-looking) FSA is now safe to minimise.

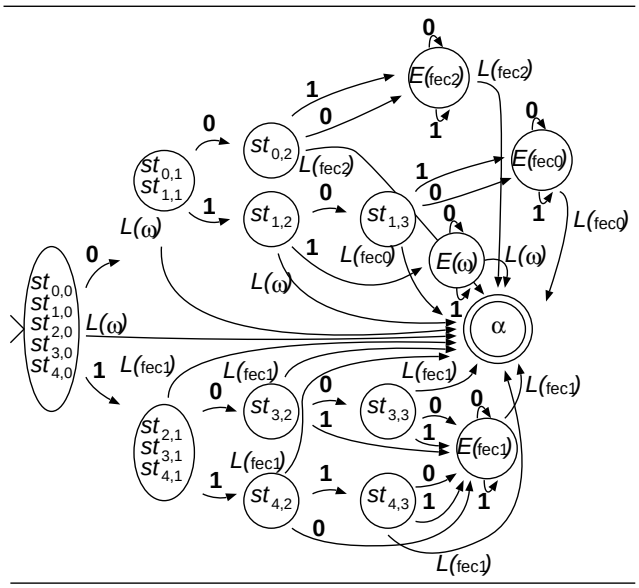

Figure 7. FSA after protection, M''

4.2. Minimisation details

The FSA M'' resulting from the procedures given in Section 4.1 is deterministic but not total (remember the alphabet has been grown). Any minimisation algorithm may be used, but for concreteness, we use Brzozowski's algorithm [2]: determinise, reverse, determinise again, and finally reverse again, arriving at M'''.

Figure 8 shows the result of minimising our example. The state names have been mostly removed since their relation to the original names present in Figure 4 is tenuous.

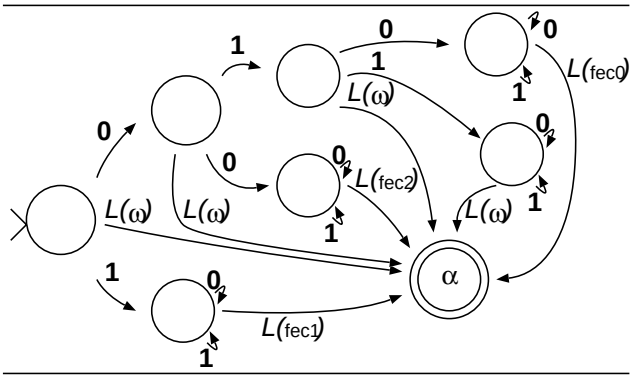

Figure 8. FSA after minimisation, M'''

4.3. Removal of protection

Finally, now that M'' has been minimised, the labelling letters can be removed to yield a deterministic automaton

M'''' with the original alphabet, and a correct W'''' set mapping accepting state to FEC.

This is done as follows:

- $\Sigma'''' = \{0, 1\}$,
- $S'''' = S''' - \{\alpha\}$
- $I'''' = I'''$,
- $A'''' = \{q : (q, \mathcal{L}(f), \alpha) \in T''' \wedge f \in F \neq \omega\}$,
- $W'''' = \{(q, f) : (q, \mathcal{L}(f), \alpha) \in T''' \wedge f \in F \neq \omega\}$,
- $T'''' = T''' - \{(q_s, c, q_d) : q_d = \alpha\}$.

That is, a state is accepting if there was a labelling letter transition out of it (and it wasn't $\mathcal{L}(\omega)$). For each accepting state, the W'''' relation gives the FEC associated with that state by using the labelling letter to find out which it was.

This produces the final, optimal, FSA M'''. For our example, the result is shown in Figure 9.

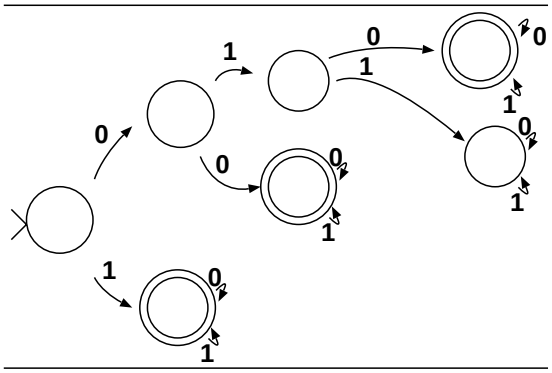

Figure 9. Final optimal FSA, M''''

M'''' is optimal because the minimisation algorithm produces an FSA with provably minimal number of states (see e.g. [11] for a proof). Since in our framework, each state corresponds to a bit-compare, we have shown the minimal sequence of bit compares needed to classify an IP address. Notice the strong similarity to a trie data structure. In fact, minimisation of an FSA as described here seems to be equivalent to a trie with route aggregation performed: the entries in Figure 3 for subnets 2, 3, and 4 have been merged since they share a common prefix and all route to the same FEC.

5. Observations

We have considered the case where only a single bit may be compared at a time. Major speedups are possible by comparing multiple bits in parallel, as done for example in the LC-tries proposed in [7] and in the controlled prefix expansion scheme [8]. This would correspond to expanding the alphabet to include symbols composed of multiple bits, e.g. $\Sigma = \{00, 01, 10, 11\}$. Being able to change the alphabet on a per-node basis would give the ability to capture the semantics of an LC-trie, where the branching factor is potentially different at each level in the trie.

If all transitions out of a state lead to the same destination state, this indicates that the bit being tested at this position in the IP address is not significant in determining the FEC. This information may be useful when generating hash functions from IP address to FEC, since including such bits in the hash does not help in discriminating between FECs. In Patricia-style trie representations [1], these bits are ignored by the "skip" property.

We note that the constructed matcher is tailored to a *particular* routing table. The concept of taking advantage of the structure inherent in a particular routing table is a powerful one.

Another technique inspired by the comparison tree resulting from the minimised FSA might be a hardware implementation. By building the comparison tree in a field programmable gate array (FPGA), data can be switched at high speed. Each node in the FSA is implemented as a one-bit decoder, and the routing table is expressed in the interconnections between these decoders. As each state in the FSA is a test of a particular bit in the destination IP address, the appropriate bit is applied to the decoders at each level in the tree, thus selecting a path through the tree to the output. There is a per-packet setup delay while the destination address is presented to the comparison tree and the gates settle to enable the appropriate path through the tree to the output, but after this the remainder of the packet data can be sent at a rate limited only by the propagation delay through the tree. When the routing table changes the FPGA needs to be re-programmed with the new tree. Sadly, current FPGA technology is unsuitable for this sort of design due to limitations on their internal interconnection networks; this does not preclude a custom FPGA designed with this use in mind.

Further work includes investigating other interesting transforms on the routing table while it is encoded as an FSA.

6. Conclusion

We have shown how the process of doing an IP route lookup may be re-phrased as a string-matching problem. We have applied well-known results from the field of finite state automata to show how the optimal string matcher specialised for a routing table may be constructed. We note the similarity to binary tries currently in use. This gives us confidence that binary tries are in fact the best way of doing IP route lookups if restricted to single bit compares.

We do not consider using FSAs themselves for route lookups to be a practical technique – the minimisation process tends to be exceedingly expensive, $O(2^{|S|})$. For a recent backbone routing table with around 68000 entries, $|S|$ is approximately 205000 after determinisation, leading to prohibitively expensive runtime costs. The value of the idea lies in the insights observable from this novel perspective on the inherent nature of the longest prefix match problem.

A. Appendix: Determinisation

Given a non-deterministic FSA $M = (S, \Sigma, I, A, T)$, the deterministic FSA $M' = (S', \Sigma', I', A', T')$ which accepts the same language as M is defined as follows:

- $S' = \{Q : Q \subseteq S\}$. The new set of states is the powerset of the original set of states.
- $\Sigma' = \Sigma$. The alphabet is unchanged.
- $I' = \{I\}$. The new start state is the set containing the set of start states.
- $A' = \{Q \in S' : \exists q \in Q \wedge q \in A\}$. A new state is accepting if any of its constituent states was accepting.
- $T' = \{(Q'_s, c, Q'_d) : Q'_d = \{q_d : \exists q_s \in Q'_s \wedge (q_s, c, q_d) \in T\}\}$. There is a transition from Q'_s to Q'_d in the new FSA if there is a transition in M from any constituent state of Q'_s on the same symbol, in which case Q'_d is composed of all the states reachable from Q'_s on consuming input symbol c.

This is called *subset construction*, since the states in the new FSA M' are subsets of the states in M. This means that determinisation usually results in M' having many more states than M.

References

[1] J. C. Bays. *The Complete PATRICIA*. PhD thesis, University of Oklahoma, 1974.

[2] J. A. Brzozowski. Canonical regular expressions and minimal state graphs for definite events. In *Mathematical theory of Automata*, volume 12, pages 529–561. Polytechnic Institute of Brooklyn, Polytechnic Press, N.Y., 1962.

[3] M. Degermark, A. Brodnik, S. Carlsson, and S. Pink. Small Forwarding Tables for Fast Routing Lookups. *Computer Communication Review (ACM SIGCOMM'97)*, 27(4):3–14, Oct. 1997.

[4] J. E. Hopcroft and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 1979.

[5] T. V. Lakhsman and D. Stiliadis. High-Speed Policy-based Packet Forwarding Using Efficient Multi-dimensional Range Matching. *Computer Communication Review (ACM SIGCOMM'98)*, 28(4):203–214, Oct. 1998.

[6] K. Mehlhorn, S. Näher, and H. Alt. A lower bound on the complexity of the union-split-find problem. *SIAM J. Comput.*, 17:1093–1102, 1988.

[7] S. Nilsson and G. Karlsson. Fast address lookup for internet routers. In *International Conference of Broadband Communications*, 1998.

[8] V. Srinivasan and G. Varghese. Fast address lookups using controlled prefix expansion. *ACM Transactions on Computer Systems (TOCS)*, 17(1):1–40, Feb. 1999.

[9] V. Srinivasan, G. Varghese, S. Suri, and M. Waldvogel. Fast and Scalable Layer Four Switching. *Computer Communication Review (ACM SIGCOMM'98)*, 28(4):191–202, Oct. 1998.

[10] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner. Scalable High Speed IP Routing Lookups. *Computer Communication Review (ACM SIGCOMM'97)*, 27(4):25–36, Oct. 1997.

[11] B. W. Watson. A taxonomy of finite automata minimization algorithms. Technical Report 93-44, Faculty of Mathematics and Computer Science, Eindhoven University of Technology, 1993. ISSN 0926-4515.