The Impact of Address Allocation and Routing on the Structure and Implementation of Routing Tables

Harsha Narayan
University of California, San Diego
hnarayan@cs.ucsd.edu

Ramesh Govindan
University of Southern California
ramesh@usc.edu

George Varghese
University of California, San Diego
varghese@cs.ucsd.edu

ABSTRACT

The recent growth in the size of the routing table has led to an interest in quantitatively understanding both the causes (*e.g.,* multihoming) as well as the effects (*e.g.,* impact on router lookup implementations) of such routing table growth. In this paper, we describe a new model called ARAM that defines the structure of routing tables of any given size. Unlike simpler empirical models that work backwards from effects (*e.g.,* current prefix length distributions), ARAM approximately models the causes of table growth (allocation by registries, assignment by ISPs, multihoming and load balancing). We show that ARAM models with high fidelity three abstract measures (prefix distribution, prefix depth, and number of nodes in the tree) of the shape of the prefix tree — as validated against 20 snapshots of backbone routing tables from 1997 to the present. We then use ARAM for evaluating the scalability of IP lookup schemes, and studying the effects of multihoming and load balancing on their scaling behavior. Our results indicate that algorithmic solutions based on multibit tries will provide more prefixes per chip than TCAMs (as table sizes scale toward a million) unless TCAMs can be engineered to use 8 transistors per cell. By contrast, many of today's SRAM-based TCAMs use 14-16 transistors per cell.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network Operations—*routing*

General Terms

Measurement, Performance

Keywords

Routing tables, modeling, IP lookups

1. INTRODUCTION

In recent years, Internet measurement and modeling studies have focused on Internet topologies [5], paths [6], and routing behavior [7, 8]. Only recently has there been an exploration of the structure and growth of the global routing table. This exploration has been sparked by dramatic growth in the table size (from 30,000 to 120,000) in the last few years. Growth in table size has become a popular topic for mailing lists, and in the trade press [9] because of its alarming implications for router vendors. However, the growth in the table also leads to two natural research questions. *Q1.* How is this growth caused and how will changes in the relative prevalence of various causes affect table size? *Q2.* How does growth impact router implementations?

It is fairly well known that some of the causes of growth are the failure to aggregate properly, load balancing and multihoming. In a recent measurement study, Bu *et al.* [10] provide valuable initial insight into some of these causes by showing, for example, that multihoming contributes 20-30% of the prefixes in current routing tables and that this factor is on the rise. However, they stop short of exploring the sensitivity of table growth to changes in these causes. Furthermore, while they provide some link between causes and the total *number* of prefixes, they do not explore a link between causes (*e.g.,* multihoming) and the *structure* of the routing table. The structure of the routing table can be thought of as the shape of the tree (*e.g.,* trie) induced by the set of prefixes in the table.

Why might the structure of the table[1] be of interest — as opposed to merely the size of the table? Table structure is crucial because it helps to answer question *Q2* about the impact on router implementations. In particular, some router vendors do IP lookups based on compressed multibit tries [11, 12]. The amount of high speed memory required by such schemes (which is a first order measure of the implementation cost) is directly dependent on the structure of the routing table. Intuitively, if prefixes tend to bunch together (as opposed to being randomly scattered), multibit tries will result in very compact and affordable implementations even for large routing tables.

The issue is becoming particularly pressing because Content Addressable Memory (CAM) manufacturers are targeting offerings for large core router forwarding tables. Traditional problems with ternary CAMs (TCAMs) such as the scaling of the match logic and power consumption are being surmounted using innovative techniques. However, TCAMs still are much less dense than memories (14-16 transistor cells for SRAM-based TCAMs as opposed to 6 transistors for an SRAM cell). Thus it appears that for the same number of transistors, an algorithmic implementation (based, say, on compressed multibit tries) can handle a larger number of prefixes than a TCAM.

While this is the hypothesis, and some are working to provide

[1]In this paper, we use the term routing table to mean the collection of prefixes associated with routing entries, and ignore other route attributes (AS paths, next hops, BGP communities *etc.*).

algorithmic solutions for IP lookups as an alternative to TCAM solutions, there is little hard evidence to make a case one way or the other. The problem is that if the present exponential growth rate were to continue, one would expect core router tables to reach a million prefixes in another 6 years or so. Routers shipping today thus may have to support up to 1 million prefixes to be usable for 5 years, often the minimum period considered for such capital equipment.

Thus to reasonably compare TCAMs versus algorithmic lookup solutions one needs some way of generating realistic table sizes that are several times the size of today's routing tables. More importantly, to accurately model the storage of algorithmic (*e.g.,* trie-based) solutions, one has to ensure that the generated tables accurately reflect the structure of current (and hopefully future) routing tables. The upshot of this argument is that router vendors today *need a good model of the structure of current tables* that can be projected to the future.

Note that an accurate model can also help a chip vendor selling an algorithmic solution predict the amount of memory required to support a given number of prefixes. A model that reflects underlying causes is also useful in its own right to understand table growth, which we claim should be as much a phenomenon of independent interest as is Internet topology or BGP convergence.

1.1 Routing Table Models

In this paper, we introduce a model of routing table structure called ARAM. ARAM contains recognizable elements of the processes which govern routing table structure: the allocation of address space from registries to ISPs, and the advertisement of address prefixes into the routing system determined by routing practices such as multi-homing and load balancing. We use ARAM to study the storage requirement of IP lookup schemes as a function of table size. Because ARAM has parameters that can control the relative weight of allocation and routing processes, we can explore the impact of variations in the degree of multihoming or load balancing, for example, on the storage requirements of lookup schemes.

We argue that an ideal routing table model should have the following properties:

- *Causal:* Given there are well known causes of routing table growth such as multihoming and load balancing [10], a model should ideally reflect these intuitive driving forces in order to provide increased understanding. An alternative is to employ an empirical model which ignores causes, and directly encodes chosen measures of current tables such as prefix length distribution.
- *Parameterizable:* Merely reflecting the current structure of the routing table is dangerous because vast increases in some causes (such as multihoming) can lead to very different routing tables. Thus, an ideal model should have parameters (tuning knobs) that can control the effects of various parameters to enable a systematic sensitivity analysis.
- *Parsimonious:* To effectively use and reason with a model, the model should have as few parameters as possible. In the limit one could characterize the shape of the current routing table by providing the prefix length distribution for all possible initial 8-bit values of prefix bits. While this might characterize the shape better, it uses 256 * 32 parameters.
- *Accurate:* The model should match the "shape" of existing routing tables. Ideally, shape comparisons should use abstract measures that can be used to compare any two prefix tries, regardless of the particular lookup schemes.
- *Predictive:* The model should be able to accurately predict the memory used by various lookup implementations as measured on existing tables. Unlike the previous goal where validation is done using abstract shape measures, here the goal is to validate a model using more concrete measures such as the memory used by a representative IP lookup scheme.

The design of ARAM attempts to satisfy these sometimes conflicting goals. It derives representative routing tables in three steps that model the processes by which prefixes enter the table: first registries allocate address blocks to ISPs, ISPs advertise their allocations into the routing table and assign address space to customers, and customer in turn can advertise more-specific prefixes (*i.e.,* "punch holes") to effect specialized routing (*e.g.,* backup or load balancing). The model uses five parameters: one for the number of allocations in the first step, and two each for the two remaining steps that govern the frequency and extent of ISP and customer routing practice. While the model can be made more accurate by using more parameters, we decided to err in the direction of parsimony.

By three abstract measures of tree shape (prefix length distribution, prefix depth, and number of tree nodes), ARAM's routing tables compare well (Section 2.3.2) with twenty routing tables spanning the last five years. For example, for the prefix length distribution, ARAM exhibits less than 6% error for each of those twenty tables. To match a specific routing table, we used the same number of allocations as an input to ARAM as had been handed out by the registries at the time the routing table was taken. For the other four parameters, a single set of values was sufficient to produce a match.

Finally, for each of the twenty routing tables we chose, ARAM's matching routing tables also closely matched the number of transistors required for a compressed multibit trie implementation (Section 3). Our scheme of choice is the Tree Bitmap algorithm, due to Eatherton and Dittia [12]. It is less dated than the seminal Lulea scheme [11] and has fast updates (unlike the Lulea scheme). However, we believe the results would not change significantly for other implementations because the underlying abstract property that determines trie storage is the shape (*i.e.,* the relationships between prefixes) in the routing table.

We make no claim that ARAM captures all the aspects of routing and allocation practice, that it cannot be further tuned, or that there are no other effective causal models. However, our validation of ARAM (Section 2.3.2) gives us some confidence in our use of ARAM to generate larger table sizes and explore variations in routing and allocation practice. In particular, we use ARAM to shed some light on the somewhat acrimonious debate between TCAMs and algorithmic IP lookup schemes. Leaving aside power and match scaling, our results indicate that algorithmic schemes can provide more density (prefixes per unit area) than TCAMs. Of course, our results are subject to assumptions (*e.g.,* that the current growth trends continue), and thus should only be considered an initial (albeit quantitative) contribution to an ongoing debate.

Section 2 is devoted to the ARAM model and its validation, and Section 3 is devoted to using ARAM to predict IP lookup performance. Finally, Section 4 compares ARAM to previous work, and Section 5 states our conclusions.

2. ARAM: THE MODEL AND ITS VALIDATION

In this section, we discuss current allocation and routing practice, and then describe how ARAM generates routing table prefixes. We then validate ARAM's routing tables. As well, we validate each individual aspect of the model.

2.1 Introduction

In order to allow meaningful extrapolation of routing tables, ARAM attempts to explicitly model the factors that shape routing tables and the relationship between prefixes. Broadly speaking, there are two mechanisms that shape current routing tables: the allocation of prefixes by registries, and the advertisements of those prefixes (or more specifics thereof) in BGP tables by ISPs and their customers. Both these mechanisms are only informally codified, if at all. For this reason, we use the terms *allocation practice* and *routing practice* respectively to refer to them.

2.1.1 Address Allocation Practice

In this section, we describe the essential details of the hierarchical allocation of IPv4 addresses and address prefixes. ARAM attempts to capture some, but not all of these details. There are certain local variations in allocation practice that ARAM ignores [4].

The IPv4 address allocation hierarchy has four levels. The Internet Assigned Numbers Authority (IANA, currently administered by the Internet Corporation for Assigned Names and Numbers (ICANN)) delegates blocks of addresses to Regional Internet Registries (RIRs). These, in turn, *allocate* portions of their address blocks to Local Internet Registries (LIRs). With few exceptions, LIRs correspond to ISPs. In turn LIRs *assign* parts of their address space to "end-users" (organizations or smaller ISPs).

This hierarchy for address allocation has two logical functions. Decentralization of allocation from IANA to the RIRs and then to the LIRs promotes manageability of address allocation. Moreover, the fact that LIRs mostly correspond to ISPs promotes (at least in theory) the scaling of the routing system by enabling aggregatable address assignment.

We now describe the various levels of this hierarchy in some detail.

IANA is the guardian of the entire IPv4 address space. It delegates parts of the space to the RIRs in units of /8 on-demand. IANA holds 113 /8s [13], of which 35 have currently been delegated to various RIRs (the rest of the address space is made up of historical allocations and the class B space). In order to qualify for a new /8, an RIR has to have used up 80% of its existing /8 or demonstrate that it cannot meet an allocation request with its current /8[2].

There are four RIRs (ARIN in North America, RIPE in Europe, APNIC in Asia-Pacific and LACNIC, the newest RIR responsible for South America). Each RIR is responsible for address management in its designated geographic region. RIRs codify their address management practices in policy documents [14, 15, 16]. Although there are minor regional variations in policy [4] the practices of the various RIRs are qualitatively similar.

An RIR usually allocates address *prefixes* (address blocks aligned on a bit boundary) to an LIR. Address prefixes handed out to the LIR are called *allocations*. Though LIRs are usually ISPs, sometimes an RIR will allocate addresses directly to large end-users. Such allocations from an RIR to an end-user go by various names: "Direct Assignment", "Direct Allocation", "Provider Independent Allocation" or "Provider Independent Assignment".

The smallest size of an initial allocation made by an RIR is /20. Further allocations are made when the LIR has assigned 80% of its previous allocations to customers. The size of subsequent allocations is determined by the usage rate of past allocations and the expected growth rate (ISPs have to make a business case to the RIRs to demonstrate their expected growth rate). RIRs try to ensure, but do not guarantee, that an allocation to an LIR is aggregatable with prior allocations.

[2]To reduce administrative load, this policy is currently being changed so that an RIR may get enough /8s to last for 18 months at a burn rate that is determined by its recent rate of allocations.

The lowest rung of the address allocation hierarchy is that between ISPs and their customers. Address blocks handed out by ISPs to their customers are called *assignments*. The size of these address blocks depends upon customer demand. It is difficult to gauge how ISPs manage their allocated space, but we believe that most ISPs use a first-fit or best-fit algorithm to make assignments [17, 18], regardless of whether the space assigned to an individual customer is aggregatable. Our belief is based on the fact that two freely available software packages for making assignments (FreeIPdb [19] and NorthStar [20]) both use a best-fit algorithm to choose address blocks to make assignments.

2.1.2 Elements of Routing Practice

Allocation practice determines how, and to whom, address blocks are assigned. Routing practice determines which of these address blocks appears in the routing table, and in what form (either in its entirety, as sub-blocks, or as more specifics of a block). ARAM attempts to capture the dominant routing practices that we have been able to infer from the data. It does not incorporate several arcane routing practices [4].

The ideal, espoused by CIDR, is that each allocation to an ISP has exactly one entry in the routing table, and no assignments appear in the routing table. That is, each customer advertises its assignment as a route in BGP (or perhaps using static routing or sharing an IGP with the ISP, although these practices are probably less prevalent now), but the ISP aggregates these routes, and advertises to its peers or its upstream provider the address prefix representing its allocated space. Reality is far from this.

Deviations from this ideal are caused by a variety of routing practices. Most of these routing practices result from *multihoming*. We use this term in a very general sense to include customers connected to multiple upstream providers and ISPs buying transit from multiple providers. In the following paragraphs, we describe these routing practices and their effect on the routing table.

Some ISPs *split* their allocation, and advertise the split prefixes separately in BGP. For example, the allocation made to UUNET, `63.64.0.0/10` is sometimes advertised as four /12s rather than directly as a /10. There are two generic examples of ISPs that may adopt this practice, both of which are attempts to engineer traffic flows. A small ISP may split its allocation and advertise different split prefixes to different upstream providers, thereby effecting "load sharing": *i.e.,* spreading in-bound traffic across different upstream providers. A large national ISP may have an internal level of hierarchy in address allocation. That is, it may split its allocation into address prefixes, one each for the different geographic regions where it has infrastructure—customers from a particular region get address space from the address prefix assigned to that region. The ISP then advertises these split prefixes without aggregating them; this allows it to have more fine-grain control of routing. (There are some other examples where an allocation appears to have been split in the routing table [4]).

A similar practice is followed by some large end-users like web-hosting services, which have obtained a provider independent allocation. Such an end-user might split its allocation and announce the different sub-blocks from different upstream providers, for load-sharing reasons.

The previous routing practices described scenarios in which the original allocation does not appear in the routing table, but sub-prefixes of that allocation do (this is not always true, since sometimes an ISP will advertise its allocation intact and also advertise sub-prefixes; we return to this in a later section). These sub-prefixes may not completely cover the original allocation's address space.

We label this practice *splitting*, and refer to the allocation that splits as a split allocation, and to the products of splitting as split prefixes. As we discuss later, not all allocations split. Some allocations appear in their entirety as one prefix in the routing table. We call these *intact* allocations, and the corresponding prefixes *intact* prefixes.

We now describe routing practices that describe how *assignments* to customers (or sub-prefixes thereof) appear in the routing table. There are (at least) three qualitatively different routing practices in this category.

A customer of an ISP might provide for backup connectivity by advertising its assigned space through an upstream provider (say ISP B) different from the one it obtained the assignment from (say ISP A). In this case, A's allocation appears (either intact or split) in the routing table and is a less-specific prefix of the customer's assignment. A slight variant of this practice is that the customer, instead of advertising its entire assigned space advertises a *part* of it through the upstream, preferentially drawing traffic from its upstreams. Finally, a customer which gets its addresses from one ISP might split its assignment into multiple prefixes to perform load sharing among different providers.

From the perspective of the routing table, these routing practices result in more-specific prefixes being advertised in BGP. We say that a prefix *spawns* more-specifics, and that these routing practices govern *spawning*.

2.2 ARAM

A detailed causal model of a routing table might have attempted to explicitly include *all* the processes described above: the allocation of address space to LIRs, the assignment of address prefixes to customers, and the various routing practices described above. Such a model, while closer to physical reality, is very difficult to define, and even harder to validate. As we shall see, the upper-levels of the address assignment hierarchy (IANA to RIRs and RIRs to LIRs) are relatively easy to model. Data about assignments is available, and it may be possible to construct models for this. However, succinctly capturing the physical processes underlying routing practice seems to be beyond our reach today. While the kinds of routing practices that occur today are probably small in number, economic and other considerations determine which ISPs and which customers follow what kinds of routing practices. Capturing these considerations is left for a future generation of models.

ARAM is a less ambitious, but nonetheless very useful, hybrid model, with three parts. First, it models the allocation of address prefixes from RIRs to LIRs. Then, it incorporates knobs that determine how these allocations appear in the routing table—*i.e.,* whether the allocation is split, or appears intact. Finally, it models the appearance of customer assignments in the routing table by providing knobs that determine prefix spawning (Section 2.1.2).

In the rest of this section, we describe these three components of ARAM, and provide intuitive justification for our design of the model. In Section 2.3, we validate ARAM's design.

Before describing the internals of ARAM, we describe its inputs and outputs. The output of ARAM is a routing table of a desired size. More precisely, the output is the collection of address prefixes that might appear in a routing table of this size, and *does not* include other routing attributes that might be associated with these prefixes, such as AS paths, next hops, or BGP communities. The desired size of the routing table is not an explicit input to ARAM; rather it is implicitly defined by five input parameters:

- The number of allocations made by RIRs to LIRs is determined by a parameter N.
- F_{split} is the probability of an allocation splitting and C_{split} is the percentage of address space of a split allocation that is advertised in the routing table
- Finally, F_{spawn} is the probability that a split or intact prefix spawns one or more prefixes, and C_{spawn} is the fraction of address space of a split or intact prefix that is advertised in the routing table as spawned prefixes (*i.e.,* as more specific prefixes).

Apart from N, the other parameters encode the *aggregate* impact of various routing practices described in Section 2.1.2. Exporting these knobs allows users of the model to extrapolate how changes in routing practice might affect the "shape" of routing tables (*i.e.,* the relationship between prefixes).

Modeling Allocations

The first step in generating a routing table in ARAM is to generate N allocations. Following current registry practice, ARAM generates allocations whose prefix length is between /10 and /20. The number of prefixes $N(x)$ of length x (more precisely, x is the prefix length minus 9, so that the shortest prefix generated is a /10) is determined by a function of the form $N(x) \propto x^k$ for some constant k.

The intuition for this simple form of allocation distribution arises from the observation that RIR allocation policy is based on documented need. To get address space of a certain size, an LIR must justify the need for that size either by presenting a business case or by demonstrating an appropriate rate of assignments to customers [15]. Thus, a large ISP is (eventually) allocated a large block of addresses (shorter prefix length), and small ISPs are allocated smaller blocks (longer prefix length). Since the distribution of ISPs can plausibly be said to be a power law (following the more general, and well-established empirical observation that companies are power-law distributed by size [21]), we can expect that $N(x)$ follows the form described above.

This model makes one important simplification. In practice, an LIR is progressively allocated space based on need and, because these allocations are not guaranteed to be contiguous, an LIR may have several distinct prefixes allocated to it. ARAM, however, implicitly allocates one prefix to an LIR, and assumes that this prefix is the aggregated result of several allocation requests. Furthermore, this version of ARAM hard-codes the value of $k = 3.4$ into the model. In principle, of course, we could have exported k to enable the user of a model to explore how quantitative changes in the allocation distribution affect routing tables. We have left this for future work.

The mechanics of allocation in ARAM work as follows. ARAM maintains a pool of /8s from which it makes these allocations. As a matter of detail, the /8s that ARAM uses are the same ones that are in use by RIRs, or that have been reserved by IANA for future use. There are about 113 such /8s. ARAM repeatedly performs the following steps N times (assuming that the /8s are numerically ordered):

- It draws a random sample from the distribution $N(x) \propto x^k$. This determines the prefix length of an allocation.
- It allocates this prefix from the first /8 whose current utilization is less than 80%.

In this way ARAM sequentially fills up /8s until N allocations have been made.

Modeling Advertisement of Allocations

The next step in the model determines how allocations appear in the routing table. As we discussed in Section 2.1.1, although one might expect an allocation to appear intact in the routing table, a variety of prevalent routing practices cause allocations to "split" and appear as a collection of sub-prefixes in the routing table.

Rather than attempt to model these routing practices, ARAM defines the frequency and extent of this practice using two parameters. F_{split} defines the percentage of allocations that split. For each allocation, ARAM tosses a coin with probability F_{split} to determine whether that allocation is split or intact. When an allocation splits, in practice, the collection of split prefixes corresponding to that allocation does not usually cover the allocation's address space. Intuitively, these split prefixes cover only that part of the address space actually utilized (assigned to customers). To model these, ARAM uses the C_{split} parameter and, for each split allocation, generates a number of prefixes such that the address space covered by those prefixes is as close as possible to C_{split} times the address space of the split allocation.

Into what prefixes does an allocation split? From our analysis of the data, there does not appear to be a dominant pattern of allocation splitting, nor does there appear to be a rationale (*e.g.,* larger allocations splitting in a certain way) for the way split allocations appear in the routing table.

There is one important exception to this. Some web-hosting and content providers (these are not the only examples of such practice; the other examples defy classification but are numerous) that obtain their own allocations from RIRs sometimes split into a large number of small prefixes (*e.g.,*/24s). Each such prefix intuitively represents one data-center or part thereof. ARAM captures this practice by assuming that a fixed fraction (20%) of split allocations split in this way.

For the rest of the split allocations, we use a rather ad-hoc *splitting rule*. Two observations guided the design of this rule. First, in practice, allocations do not split into equal-sized address blocks. In particular, when splitting is for the purposes of load sharing, one might expect that skewed traffic distributions to different parts of the address space will result in size diversity among the split prefixes. Second, for reasons of routing manageability, we suppose that allocations will split into a relatively small number of prefixes. The *splitting rule* we use is:

> Split the allocation into prefixes of length /$(i+2)$ and /$(i+3)$, where /i is the length of the original allocation. The number of /$(i+2)$s and /$(i+3)$s are determined as follows: for every /$(i+2)$, two /$(i+3)$s are also produced (*i.e.,* equal amounts of address space appear as /$(i+2)$s and as /$(i+3)$s). Finally, as many /$(i+3)$s are added as are necessary to cover up to C_{split} times the address space of the allocation.

Modeling More Specifics

In this final step of the model, ARAM determines the prefixes which are spawned from intact or split prefixes. Essentially, spawned prefixes in a routing table appear as more-specifics of other prefixes, and represent various routing practices: backup routing and customer load sharing among upstreams (Section 2.1.2), or an ISP announcing its allocation as well as prefixes split from that allocation (Section 2.1.1).

As with splitting, rather than modeling these routing practices, ARAM encodes their effect using two parameters: F_{spawn}, and C_{spawn}. F_{spawn} is the probability that an intact or split prefix actually spawns at least one more-specific prefix. Intuitively, not all such prefixes spawn more-specifics. Consider a cable ISP that splits up its allocation; because such an ISP serves residential customers or small businesses, the ISP does not have customers who engage in multi-homing. Similarly, an end-user who receives a direct allocation from the RIR would not need to spawn prefixes if it does not perform load sharing.

Furthermore, not all the space covered by an intact or split prefix appears as spawned prefixes. For example, only some of an ISP's customers may actually multi-home, resulting in more specifics in the routing table. Once ARAM has decided, based on a coin toss with probability F_{spawn} whether a given intact or split prefix spawns some prefixes, it then generates spawned prefixes such that C_{spawn} of the address space of the original prefix is covered by spawned prefixes.

Finally, as with splitting, our rule for generating spawned prefixes represents a delicate compromise between trying to capture diversity in routing practice, and keeping the model simple and understandable. We have two *spawning rules*. First, all spawned prefixes are in the range /19-/24. This rule follows from ISP filtering practice which limits the longest prefix that may appear in backbone routing tables to /24. In addition, very few customers get blocks larger than /19 from ISPs. Second, to generate spawned prefixes for a given "parent" prefix, ARAM repeats the following two steps until the fraction C_{spawn} of the parent prefix is covered.

- Pick the largest i between 19 and 24 such that one /i, two /$(i+1)$s, four /$(i+2)$s and so on up to 2^{24-i} /24s can be generated within the spawnable address space. (The basic idea is that equal address space is devoted to each prefix length). The motivation for this rule is its simplicity. It closely parallels our splitting rule; in that case, however, routing manageability was used as a motivation to keep the number of split prefixes relatively small. Such a consideration isn't necessary for spawning since an ISP cannot, in general (of course, there may be exceptions to this) control what its customers do with their assignments.

- Assign each prefix, without overlap, to a random location within the parent prefix's address space. The intuition behind this is that from the perspective of an ISP, which customer decides to advertise its assignment for backup or load balancing is generally uncorrelated with the address space, so that the more-specifics can be expected to be quite random.

In summary, notice that ARAM models a two-depth routing table. By *depth* of a prefix, we mean the number of its less specific prefixes or ancestors. All intact and split prefixes appear at depth zero of the routing table (*i.e.,* they have no less specific routing table entries). All spawned prefixes appear at depth one and have one parent (either an intact or a split prefix) from which they are spawned. Actual routing tables have a small percentage of prefixes at other depths (not more than 10% during the last 5 years).

2.3 Validation of ARAM

In this section, we validate ARAM by comparing actual routing table snapshots against comparably sized tables generated by ARAM. We then discuss each aspect of ARAM's design, and provide quantitative justification wherever possible.

2.3.1 Data Sources, Assumptions and Methodology

To validate our modeling of allocation practice, we use data from the three main RIRs (ARIN, RIPE and APNIC)[3]. Databases of allocations made by the RIRs are publicly available [24, 25, 26]. LIRs register their assignments in a Whois database; a bulk data dump of these assignments is also available from the RIRs upon written request.

We processed the allocation databases, to sanitize them, in two ways. First, while most allocations are powers of two, some (about 0.5%) cannot be expressed as a single prefix. This may happen

[3]A fourth, LACNIC, was established only in Nov 2002 and is not included in our validations.

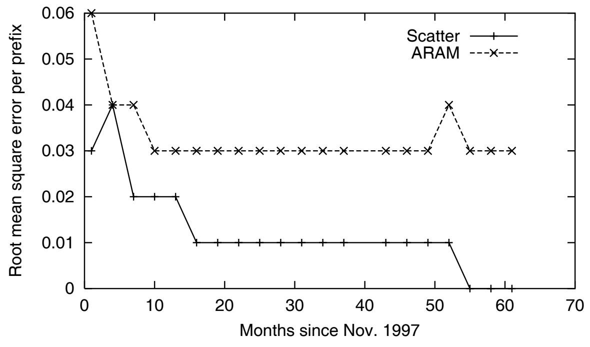

Figure 1: Normalized RMS error per prefix

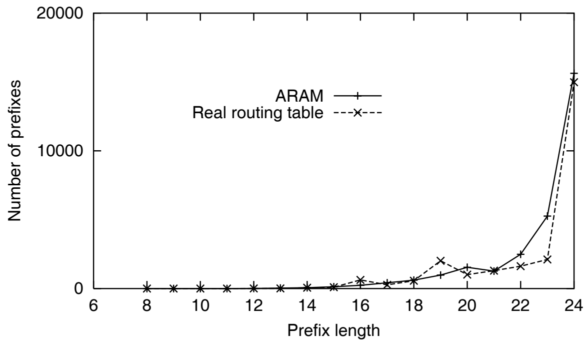

Figure 2: August 1998: Prefix length distributions

when an RIR makes allocations which are not a power of two [22] or if two contiguous prefixes of different sizes were allocated to the same organization [23]. We break up such allocations into the smallest possible number of prefixes. Second, occasionally some ISPs will announce aggregated entries of their allocations. In our study, we treat the aggregate as the allocation. This processing results in two invariants; all allocations can be expressed as address prefixes, and an allocation cannot be covered by a less specific prefix from the routing table.

For validating and understanding routing practice, as well as to calibrate ARAM's performance, we used routing tables from Route Views [27]. We processed the routing tables in two important ways. First, we removed *historic* routing table entries (those prefixes that belonged to address space that had been allocated prior to the establishment of the registries). ARAM does not attempt to model these historic prefixes. Since our goal is to extrapolate current allocation and routing practice, we argue that the effect of these historical prefixes will soon become negligible. Indeed, in 1997[4], these historical prefixes formed almost 45% of the routing table. As of this writing they account for almost 25% of the number of prefixes. Second, we removed prefixes longer than /24. Most ISPs filter prefixes longer than /24, although some such prefixes do appear in the Route Views database (possibly because the Route Views' peers do not apply filtering rules to their feeds).

2.3.2 *Validating* ARAM

In this section, we describe the results of several kinds of validation tests we performed on ARAM. Recall that the goal of ARAM is to plausibly capture the relationships between prefixes. This notion is somewhat difficult to precisely define, and we define instead three indirect *metrics* that capture various aspects of the relationships.

Our first metric is the normalized RMS error in the *prefix length distribution*. This metric computes the root mean square error between the prefix length distribution generated by a model and the prefix length distribution in real routing tables. We normalize the RMS error by the number of prefixes in each table, resulting in a "normalized RMS error per prefix" metric.

Our second metric is the *depth ratio*, the ratio of the number of prefixes at depth zero to that of the number of prefixes at depth one in a routing table. It attempts to capture a different facet of prefix relationships than the length distribution—how the less specifics and more specifics relate to each other.

Our final metric is a slightly more fine-grained measure of prefix relationships. The *unibit trie density* captures the number of nodes in the unibit trie [36] *per prefix* in the routing table. Unibit trie density is inversely related to prefix density; when prefixes are more closely clustered together in a routing table, fewer unibit trie nodes are required per prefix. This metric captures the relationship between parent prefixes and spawned prefixes. For example, when the difference in the lengths of a parent prefix and a spawned prefix is large, the number of unibit trie nodes is more than when the difference is small. Similarly, if of three prefixes, one is the parent of the other two, the unibit trie is likely to be smaller than a table where these prefixes were randomly scattered across the address space.

Using these metrics, we not only compare ARAM to actual routing tables, but we also calibrate ARAM against[5] a random *scatter* model. This scatter model simply scales the prefix length distribution of the current routing table by a constant factor to get the prefix length distribution of a larger (or smaller) table. It then "scatters" these prefixes randomly among the 113 /8s that are currently reserved for use by IANA.

Our performance comparisons use 20 routing tables dating back to November 1997, approximately[6] one every three months.

2.3.2.1 *Prefix Length Distribution.*

Figure 1 plots the normalized RMS error per prefix for ARAM and scatter. Notice that ARAM's RMS error is uniformly low (less than 6% over the 5-year measurement period). This is remarkable because ARAM is not explicitly engineered to produce a specific prefix length distribution (it only has coarse-grained prefix length ranges encoded in it). What is more remarkable, however, is that in producing an ARAM to match a routing table at time *T*, we used the number of allocations that had been made by the registries at *T*. Furthermore, in generating the matching tables, we used *one set of values for the split and spawn parameters*. (This is testament to the fact that, statistically speaking, the incidence of splitting and

[4]While we would have liked to study the evolution of the routing table soon after the deployment of CIDR, we have been unable to do so due to the lack of periodic routing table snapshots from that era.

[5]Another natural model to compare ARAM to would have been RTG [28]. For various logistical reasons this hasn't been possible. Despite our efforts, the source code for RTG did not run to completion when asked to generate large tables. Furthermore, the RTG paper does not include enough details to reproduce the prefix generation part of the generator. We are working with the authors of [28] to resolve some of these issues. In the meantime, we resort to a qualitative comparison between the models (Section 4).

[6]In one case, incomplete data in the Route Views database prevented us from taking samples *exactly* three months apart.

spawning hasn't changed much over the 5 years.) To give the reader a visual sense of the closeness of the match, Figure 2 compares the prefix length distributions in actual routing tables to those generated by ARAM.

Scatter has lower RMS error than ARAM, in general. This is because scatter, by design, attempts to match the prefix length distribution. The reason the RMS error is non-zero in some cases is that we generated all the scatter tables using the prefix length distribution from one instance of the actual routing table.

2.3.2.2 *Depth Ratio.*

Figure 3 plots the depth ratio for ARAM, scatter and the routing table. ARAM clearly matches the depth ratio of each instance of the routing table, but this is not surprising since ARAM is implicitly engineered (given its split and spawn parameters) to generate a number of prefixes at depth zero and depth one that is comparable to those in a real routing table.

However, it is important to note that scatter gets the depth distribution completely wrong, and it is not hard to see why. Because prefixes are scattered randomly among the /8s, it is less likely that a prefix will be at depth one (*i.e.,* a more specific) especially for smaller routing tables. This metric starkly illustrates (as does the next) how ARAM clearly outperforms scatter.

2.3.2.3 *Unibit Trie Density.*

Finally, Figure 4 plots the unibit trie density for ARAM, scatter, and the real routing tables for our twenty measurement points. Scatter grossly overestimates the density, because its prefixes are essentially uncorrelated with respect to where they are placed in the address space.

However, a real routing table exhibits significant relationships between prefixes, resulting in smaller unibit tries for the same prefix length. ARAM performs well; for current routing tables, ARAM is off by less than 7% in this metric. This is encouraging, especially considering the simplifications we make in the model (Section 2.2).

2.3.3 *Validating ARAM's Design Assumptions*

Having validated the performance of ARAM, we now quantitatively justify many of our design decisions.

2.3.3.1 *Allocation Size Distribution.*

An important component of ARAM is that the prefix length distributions of allocations is well modeled by a function of the form $y = x^k$ (Section 2.2). We used $k = 3.4$. The R-squared value for this value of k is 0.92. The fit deviates from the actual data at two prefix lengths: /16 and /19. Pre-CIDR allocations of class B address space account for the deviation at /16. The deviation at /19 is essentially a boundary effect; until recently, the RIRs used /19 as the minimum allocation size

An exponential also fits the data well, although, as we have discussed before, there is an intuitive justification for the algebraic form (that larger ISPs get smaller prefixes, and ISP sizes can be expected to follow a power-law).

There is one caveat to this distribution. ARAM assumes that the maximum size of an allocation is /10 since currently there are no allocations whose prefix length is smaller than 10. ARIN allocates (at a given time) blocks no larger than a /13, and the existing larger allocations that one sees in the databases are the result of contiguous allocations made over time. We expect this policy to continue, since a larger maximum allocation size reduces an RIR's /8 significantly (at a given time, each RIR only has a small number of /8s that it allocates from). The other RIRs may change their policies to have a maximum allocation size too.

ARAM, for simplicity, starts allocating prefixes from a fresh /8 when the current /8's utilization reaches 80%. In practice, /8s can have a utilization ranging from 80% to 99%. We learnt from APNIC [30] and RIPE [31] that they use their /8s up to 80% before asking IANA for more. We also learnt from ARIN [29] that they do not make fresh allocations from a /8 whose usage has crossed 80%; such a used /8 is only used for growing allocations which have already been made from it (recall that RIRs try to make contiguous allocations). This is a practice followed in general, by all RIRs. However the usage or non-usage of allocations would not have a major effect on trie properties because only a small percentage of prefixes are intact prefixes. ARAM also leaves one partially used (*i.e.,* with less than 80% utilization) /8 per RIR. In practice, there may be more than one such /8 per RIR; IANA may sometimes choose to give two new /8s to an RIR at once [29].

2.3.3.2 *Modeling Intact and Split Prefixes.*

ARAM represents the fraction of prefixes that split using a single parameter F_{split}. This is a simplification. In practice, we see that the fraction of split allocations decreases with decreasing allocation size. We could have used a distribution as an input, but in the absence of a clearer understanding of *why* splitting varies with prefix length, doing this would not have helped us understand how routing practice contributes to extrapolations. Obtaining this understanding is left for future work, but a possible explanation for the downward trend for F_{split} is that larger allocations split more frequently because they need finer grain control over traffic.

We earlier gave some reasons for allocation splitting: geographic distribution of prefixes, and load sharing. One relatively minor, but interesting regional variation is caused by a small number of LIRs in the RIPE region (*e.g.,* the National Institute for R&D in Informatics of Romania) which do not provide Internet connectivity to their assignees. Their allocations appear as split allocations in the routing table.

Recall that our splitting rule (Section 2.2) sometimes splits an allocation of length /i into prefixes of length /$(i + 2)$ and /$(i + 3)$. Actual routing practice is much less cleaner than our rule would suggest (details omitted for brevity), and the incidence and extent of splitting varies with allocation size.

2.3.3.3 *Modeling Spawned Prefixes.*

In Section 2.2, we assumed that spawned prefixes are in the range /19 to /24, largely because ISP assignments fall into that range. The assignment records bear this out. Less than 0.4% of all assignments registered in the whois databases (Section 2.3.1) are larger than /19. Even the largest ISPs rarely make assignments larger than /19. ARIN and APNIC require that they be consulted before any assignment larger than a /19 is made. ARIN allows extra-large ISPs to consult only when an assignment larger than /18 is to be made. RIPE has no such maximum assignment size. We learnt that there are few LIRs who would make assignments larger than /19 in the RIPE region [31].

ARAM also assumes that the spawned prefixes are randomly scattered within a parent's address space. We empirically observed that assignments made by ISPs which appear in the routing table follow no particular pattern across the ISPs' address space. This is consistent with the intuition that customers may generally multihome at will, and ISPs generally do not attempt to make multihoming assignments from a separate block (there are some exceptions).

Our choice of single parameters F_{spawn} and C_{spawn} for the frequency and extent of spawning is clearly a first order approximation. F_{spawn} actually decreases with increasing prefix length of the spawning prefix, but without an understanding of the reasons be-

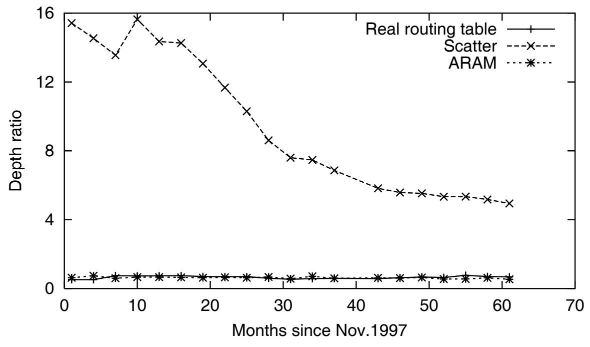

Figure 3: Depth Ratio

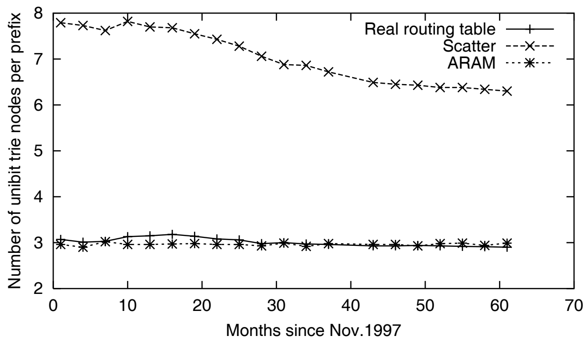

Figure 4: Unibit Trie Density

hind this variation, we were loathe to include a richer parametrization in the model.

Finally, our spawning rule is biased towards producing a larger number of longer prefixes. Thus, ARAM produces a large number of /23s and /24s (see Figure 2 for example). The larger number of /24s produced by ARAM is consistent with the data and is caused by a boundary effect; ISPs filter their routing tables at /24, and many small organizations that want to multi-home advertise /24 [7]. However, the bias shown by the spawning algorithm towards /23s is an artifact of the simplistic choice of spawning rule, and deviates from the data a little. Overall, however, as we have shown above, the routing tables produced by ARAM qualitatively match several years' worth of routing tables.

Finally, ARAM generates prefixes only at depth zero and depth one (recall that *depth* of a prefix is the number of less specific prefixes in the routing table). Over the last 5 years, consistently, fewer that 10% of the routing table prefixes have a depth greater than one (detailed data omitted for brevity).

3. EVALUATING THE SCALABILITY OF IPV4 LOOKUP TECHNIQUES

In the introduction we said that router vendors need a model that is able to generate realistic tables of a million or more entries in order to evaluate the performance of algorithmic solutions for IP lookup. The validation in the last section provides some confidence that ARAM generates realistic tables. In this section we apply ARAM to produce larger tables in order to test the scalability of a specific algorithmic solution when compared to Ternary CAMs.

3.1 TCAMs and Multibit Tries

TCAMs (Ternary Content Addressable Memories) can store the values 0, 1 and X (a "don't care" value). The ability to store don't care values makes TCAMs apt for IP prefix lookups. Essentially, TCAMs can compare a given destination address to *all* stored prefixes in parallel and return the longest match in one memory access. To store a w bit prefix, a TCAM would use w *cells* (*i.e.,* each cell stores one bit of information). For example, IP addresses would require 32 cells per prefix. Variations in routing table structure have no effect on the memory requirements of TCAMs.

By contrast, algorithmic solutions are based on storing prefixes in *tries*. High-speed algorithmic solutions store tries in SRAM. *Multibit* trie algorithms process multiple bits in the trie in a single memory access and are therefore faster. However a multibit trie increases the memory required for the lookup structure. To offset this storage increase, modern implementations that store the forwarding table in SRAM generally compress multibit trie nodes.

[7]This fact is also recognized in RIR policy. ARIN recently ratified a policy, which allows multihoming as a justification for a customer to get a /24 from an LIR (*i.e.,* the requirement that 25% of an assignment be used immediately is waived) [32]

In this paper, we study the scalability of the Tree Bitmap [12] compression algorithm. While the Tree Bitmap algorithm is basically a refinement of the seminal Lulea [11] algorithm, it appears to be more popular because it allows fast updates. CAMs have fast updates [45], and, to keep up with the competition, vendors prefer implementing algorithmic solutions with fast updates. In any case, we believe that similar scaling results will hold for any compressed multibit trie implementation, of which the Tree Bitmap algorithm can be considered a representative.

An example of a multibit trie is shown in Figure 5, where the root node is a multibit node of *stride* (the number of bits processed in a single memory access) two. The Tree Bitmap algorithm uses bitmaps to encode the multibit trie compactly. The bitmaps are of two kinds *internal* and *external*. These two bitmaps are sufficient to encode a multibit node. The internal bitmap ($2^{stride} - 1$ bits long) records the prefixes present in a multibit node. The external bitmap (2^{stride} bits long) records which child nodes of the multibit node exist. Thus, each multibit node contributes at most one internal bitmap and one external bitmap.

Along with the bitmaps, each multibit node maintains up to three pointers. The *child pointer* points to the child nodes; since the child nodes are stored contiguously, one pointer suffices to access them. The *results pointer* points to an array of next-hops. The third pointer is to the internal bitmap - it exists only when the multibit node contains prefixes. Each node also maintains flags that are associated with details of the algorithm and may vary with implementations.

For illustrative purposes, consider 8-bit IP addresses and a three level multibit trie with strides of 2, 3 and 3 (as in Figure 5). The root node of the multibit trie can have four child nodes since its stride is two. The external bitmap of the root node in Figure 5 is 1000 because only the first child node (of the four possible children) exists. Multibit trie nodes that neither contain a prefix nor have pointers to lower levels of the trie are assumed to have been removed.

The child multibit trie node in Figure 5 also has an internal bitmap of 0001100 which encodes the prefixes it contains. The bitmap is of length 7 because there are 7 nodes in the subtrie; the first bit corresponds to the root, the next two bits to the two children of the root, and the last four bits to the four leaves. The last four bits are 1100 because only the first two leaves have stored prefixes.

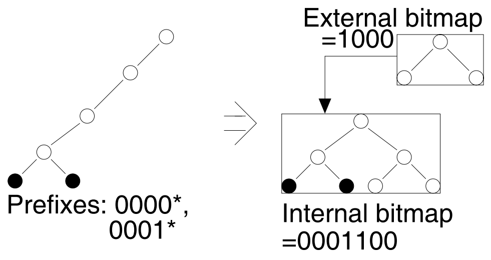

Figure 5: This figure shows an example of a unibit trie on the left and its corresponding multibit version on the right; for the prefixes 0000* and 0001*. The bitmaps of the Tree Bitmap Algorithm for the multibit version are also shown. The memory required is 45 bits.

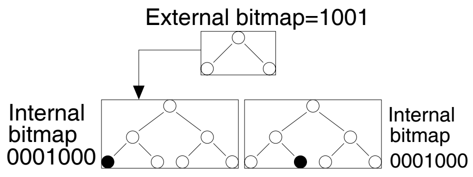

Figure 6: This figure shows what happens if the prefixes change to 0000* and 1101*. Now, two child nodes are required. The external bitmap is 1001 because the the first and the last child nodes of the root exist. The memory required is 73 bits.

To calculate the memory used in Figure 5, suppose that the memory is bit addressable. Also assume that 5 bits per node are required for flags. The memory used by the root node is 17 bits (8 bits for child pointer, 4 bits for the external bitmap, and 5 bits for flags). The child node uses 28 bits (8 bits results pointer, 8 bit pointer to the internal bitmap, 7 bits for the internal bitmap, and 5 bits for the flags)[8].

Unlike TCAMs, the memory requirements of multibit tries depends upon the relationship between the various prefixes in the routing table. To illustrate the importance of trie structure, consider the slightly different routing table containing 0000*, 1101* in Figure 6. This small change leads to a fairly big change in the memory requirement because of the need to create a second child node. The final result, using similar calculations, is $17 + 28 + 28 = 73$ bits. Note that a TCAM would have needed just $2 \times 8 = 16$ bits to store these two prefixes.

The moral of the examples in Figure 5 and Figure 6 is that the structure of the trie has a significant effect on the memory required to store the lookup data structure.

3.2 The Transistor Model

Our metric for comparing lookup techniques is the *number of transistors* required to store a given number of prefixes. While there are other important metrics like power (a typical algorithmic solution can consume six times less power than a TCAM of the same size), TCAM vendors have been working to reduce power using clever banking techniques. Transistor count appears to be a more fundamental differentiator. Lesser transistors mean more chips per wafer or higher yield. Transistor count can also act as a product differentiator in the following sense. For a given chip area and technology, there is a maximum number of transistors that can fit on a chip. If designers can provision transistor counts for storing prefixes, they can allocate the remaining transistors on the chip for additional fast-path packet processing functionality.

TCAM product offerings use between 14 and 16 transistors to store one bit of data (called a cell). To store an IP prefix, TCAMs will need 32 bits, or between 448 and 512 transistors. By contrast, six transistors are used to store one bit in SRAM. Therefore, we calculate the number of transistors taken by a multibit trie solution by multiplying the number of bits by six. For the example we discussed in Figures 5 and 6, a 14-transistor per cell TCAM would have used at least 224 transistors. The Tree Bitmap scheme would have required 270 (6×45) or 438 (6×73) transistors respectively.

Multibit trie algorithms also require a fixed overhead of logic for implementing the algorithm, and for the overhead of memory allocation and deallocation. From discussions with a vendor who has implemented a fully pipelined algorithmic solution, 20 million transistors appears to be a fairly conservative estimate for the lookup logic [9]. This overhead gets amortized over the number of prefixes. Clearly, TCAMs *do not require* this extra overhead as their lookup logic is (effectively) distributed in each bit.

3.3 Applying ARAM to Evaluate Scalability

In Section 2.3 we found that by simply varying the number of allocations while keeping the other parameters fixed, we obtained a good match with the routing table's prefix length distribution, depth distribution and trie density. As another measure of how well ARAM matches current routing tables, Figure 7 compares the number of transistors that would be required, using the Tree Bitmap algorithm, for ARAM's routing tables, the actual routing table, and those produced by the scatter model. Note the close match between ARAM's tables with the actual routing tables over the entire duration of our measurement period. By contrast, the scatter model is significantly inaccurate.

We now embark upon our scaling comparison of TCAMs and multibit tries. For our baseline comparisons, we fix all ARAM parameters to be those we used to match existing routing tables, and only vary the number of allocations N. The transistor counts used for these larger tables are shown in Figure 8. The number of transistors taken by TCAMs are shown as straight lines: CAMs-i represents an i transistor per cell TCAM. We see that unless TCAM

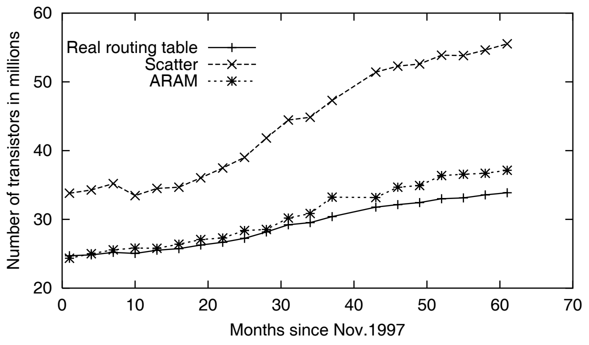

Figure 7: The memory used by the multibit trie by the real routing table, scatter and ARAM.

[8]Some of these overheads may appear large but are actually quite small for 32 bit addresses; our example used 8 bit addresses for illustration.

[9]This overhead is for a programmable processor and 32 pipeline stages. A much smaller overhead is reported in [3]. Our overhead of 20 million transistors is thus very conservative. Smaller overheads only slant the comparison further in favor of SRAM-based algorithmic solutions.

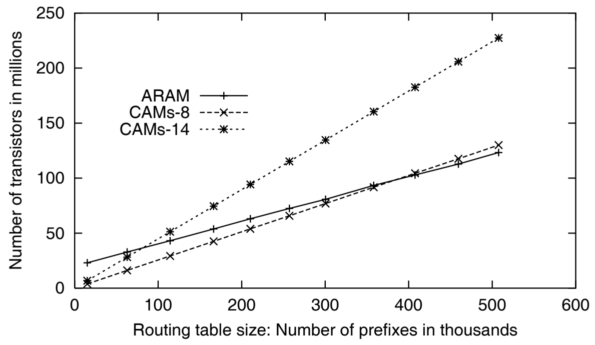

Figure 8: The size of the routing table is increased by increasing the number of allocations (other parameters are kept constant). We see that unless TCAMs use 8 transistors per cell or less, multibit tries would use lesser transistors than TCAMs.

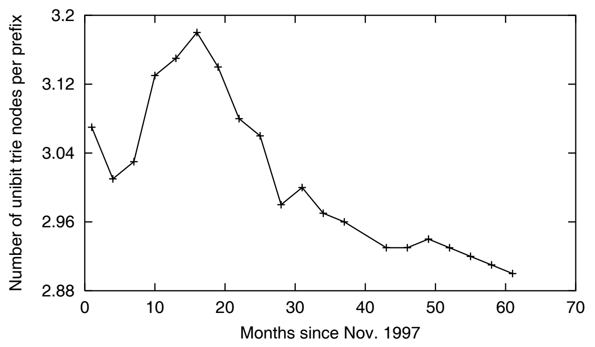

Figure 9: The density of the trie has been increasing with time.

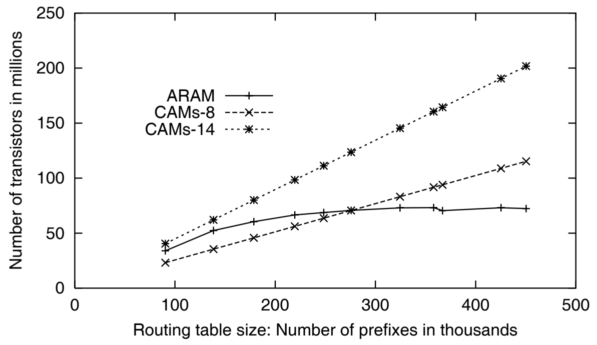

Figure 10: The size of the routing table is increased by increasing C_{spawn} (other parameters are kept constant). We see that multibit tries handle this case very well.

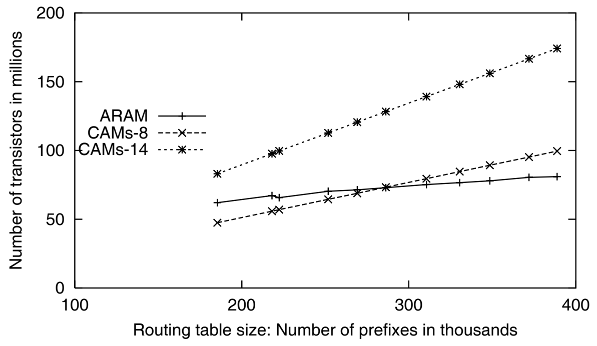

Figure 11: The size of the routing table is increased by increasing F_{split} (other parameters are kept constant). We see that multibit tries scale better in this case. This is because load shared prefixes are clustered together - more than multihoming assignments.

technology advances to the point where it becomes feasible to use 8 or fewer transistors per cell, multibit tries will scale better.

So far we have evaluated scaling in transistor counts based on extrapolating current routing practices as embodied by the ARAM parameters required to match current routing tables. It is worth asking what happens to these projections if routing practices change and the parameters deviate from the values we chose.

If routing practices change in the direction of more prefix spawning (caused, for example, by significant increase of load-sharing), we claim that the multibit trie algorithm will do even better. More prefixes will fit in the same multibit trie node, thereby amortizing the overhead of flags and bitmaps even better. Figure 10 shows that as C_{spawn} increases, the memory taken by the multibit trie algorithm increases slower than the size of the routing table. Others [37] in the community have been observing, for some time now, the increase in multihoming practice. Although we used just one set of values for the parameters—except for the number of allocations N—to approximately match all past routing tables, we have observed that the C_{spawn} in real routing tables has been steadily increasing, although almost imperceptibly. This is evidenced by the decrease in unibit trie density over time from 3.2 to 2.9 or so, Figure 9; that is, prefixes have become more "clustered" with time. A similar behavior is also seen in Figure 11; when F_{split} increases, there are more load shared prefixes at depth zero of the routing table. Varying C_{split} shows a similar scaling [4]. In general, load shared prefixes are clustered together.

On the other hand, with changes in some routing practices, compressibility of multibit tries is not improved with increasing table size. For example, increasing F_{spawn} shows a linear scaling of the multibit trie algorithm [4].

An interesting aspect of varying C_{spawn} is not apparent in Figure 10, but does become clearer in Figure 12. Notice that as C_{spawn} increases, the shape of the tree, as captured by the unibit trie density, changes non-monotonically. In particular, we see a peak in the number of unibit nodes per prefix around C_{spawn}=0.1. This non-monotonicity allows us to compute the worst-case number of transistors for a given number of allocations. To do this, we use a value of C_{spawn} of nearly 0.1, and set F_{spawn} to 1 and F_{split} to 0.

For the same number of allocations which contribute to the present day routing table, we found that the worst case parameters above generated 85,530 prefixes which required 541 transistors per prefix in the multibit trie algorithm. However the nominal parameters produced a routing table with 77,817 prefixes which required 206 transistors per prefix. The intuition is that the value of C_{spawn} is low enough not to allow the cost of a multibit node to be amortized over many prefixes, but is high enough to ensure that all depth zero prefixes spawn. Since TCAMs require 448-512 transistors per prefix, we can say that TCAMs would have scaled better (at least in the near term) if there had been no load sharing (and other forms of splitting) and all allocations spawned multihoming assignments.

We say near term because as the number of allocations increases, the density of the trie increases enough to allow the multibit trie to better TCAMs in the number of transistors. For this specific set of parameters, multibit tries would overtake TCAMs when the number of allocations grows to thrice their present number (graph not shown).

4. RELATED WORK

In this section, we briefly survey the literature on IPv4 address lookup algorithms and on address allocation and routing growth.

IPv4 address lookup techniques have received a fair amount of attention in recent years [36, 1, 2, 11, 12], and we do not attempt to be exhaustive in our survey of this sub-field. However, broadly speaking, there are two classes of lookup techniques: *algorithmic* approaches, exemplified by [12], attempt to cleverly *compress* forwarding tables compactly into memory, while hardware approaches rely on (ternary) content-addressable memories (CAMs) to perform fast lookup. We have described these schemes in Section 3.1.

Most work to date has evaluated their schemes on current routing tables. However, as we have just shown, the memory requirements of some of these lookup schemes can be sensitive to the relationship between prefixes. To our knowledge, no work has examined how these algorithms *scale* with routing table size.

However, to capture prefix relationships, we need to be able to generate realistic routing tables. One closely related piece of work, RTG [28], has attempted to generate routing tables for the purpose of generating realistic BGP updates. RTG does not attempt to explicitly model prefix relationships in the BGP routing tables that it generates. Rather, it takes an *empirical* approach: given a current routing table, it extracts some statistics from the table, and attempts to generate tables with comparable statistics. Furthermore, to *extrapolate* routing table sizes to an order of magnitude larger than today, it is necessary to *explicitly* (if only approximately) capture how allocation and routing practice affects routing table entries. This enables an examination of how quantitative deviations from current routing practice affect IPv4 lookup algorithms. ARAM, unlike RTG, explicitly models address allocation at the RIR level, and contains parameters that model multihoming and traffic engineering.

There exists a body of work that has measured or is examining routing practice. For example, Huston [37] and Gao *et al.* [10], have measured the prevalence and time evolution of routing practices but have not attempted to model these in order to generate routing tables. More recently, Xu *et al.* [46] have analyzed address allocation data and have correlated address allocations with routing table growth. However, they do not attempt to *model*, as we do, the structure of routing tables.

The IETF's Ptomaine working group [38] seeks to reduce the number of prefixes in the routing table thereby reducing load on routing. Various ideas being examined include techniques to control route propagation by using BGP community attributes [39, 40], filtering routes on RIR allocation boundaries [41], or multihoming techniques that can reduce routing load [42]. ARAM is complementary to these efforts, in that it does not prescribe measures that would reduce routing table sizes, but could be used to describe the shape of the resulting routing tables after changes in routing practice.

Basu and Narlikar [44] seek to reduce the effect of updates on throughput of lookups in a pipeline. They do this by balancing memory among the stages of a pipeline. To make a tradeoff between the speed of a fixed stride multibit trie dynamic programming algorithm and the efficiency of a variable stride multibit trie, they selectively increase strides to cover contiguous /24s. As we have seen this contiguity arises mainly due to load sharing.

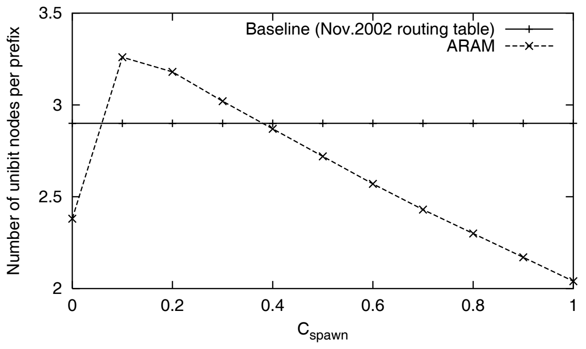

Figure 12: This graph shows the number of unibit trie nodes per prefix as C_{spawn} is increased from 0 to 1 (other parameters are kept constant). We see that the unibit trie density is lowest when C_{spawn} is around 0.1. This helps to calculate the worst case storage. The baseline is the unibit trie density for the Nov. 2002 routing table.

Finally, Kohler *et al.* [43] examine the structure and distribution of IP addresses found in traffic traces, and point out that the occurrence of IP addresses in Internet traffic is multifractal. This is somewhat orthogonal to our work, which looks at IP address allocation and prefix routing, rather than the traffic levels originated by, or destined to IP addresses.

5. CONCLUSIONS

This paper describes a model of the structure of Internet routing tables. ARAM is a parameterizable, parsimonious, accurate and predictive model of routing table growth. It qualitatively captures the processes by which address blocks are allocated in the Internet, and those by which they appear in routing tables. ARAM holds intrinsic interest in enabling the study of routing table growth in the abstract (*i.e.,* independent of any particular lookup implementation). It also applies to the evaluation of lookup schemes, right down to the transistor level.

Using such an evaluation, we find that, to a first order of approximation, multibit tries scale better than TCAMs with increasing routing table sizes. Furthermore, we find that the disparity between multibit tries and TCAMs increases with increased multihoming and load-balancing. These results have interesting implications for the choice of lookup technologies in routers. Of course, our conclusions can be negated by CAM designs which have fewer transistors per cell. There are rumors of such designs, but it unclear as to whether there is substance behind these rumors.

As an aside, in deriving these results our paper spans several levels of abstraction. Starting with the way addresses are allocated, incorporating some of the mechanisms by which ISPs advertise routing information, our paper ends with transistor level models that quantify the cost of hardware implementation.

As routing practices evolve, it will be interesting to see how accurately ARAM continues to model the routing table. For example, an increasingly popular routing practice is a VPN (Virtual Private Network) service provided by a backbone ISP. To provide this service, ISPs need to keep and advertise separate routes to each of the endpoints of the VPN. If this practice becomes significantly more prevalent than it is today, it may be necessary to tweak ARAM's splitting and spawning rules to better match routing practice. VPN prefixes could perhaps be modeled by more clustered splitting/spawning rules. VPNs can be a potential reason for prefix tables to grow more than a million. Furthermore, it will be interesting to track the evolution of allocation and routing practices for IPv6, and to study whether ARAM can be extended to model IPv6 routing tables.

Finally, we do not expect ARAM to be the last word in routing table modeling; as more accurate data becomes available, it may be possible to better infer routing practice and therefore design more accurate models than ARAM.

6. ACKNOWLEDGEMENTS

We thank the many people who answered our questions on address allocation and routing practices. In alphabetical order, they are, Cengiz Alaettinoglu, Buddy Bagga, Adam Bechtel, Randy Bush, Peter Clark, John Crain, Daniel Golding, Ejay Hire, Lee Howard, Richard Jimmerson, Glen Larwill, Herb Leong, Mike Loevner, Bill Manning, Barry Margolin, Christopher Morrow, Alec Peterson, Mark Prior, Gerard Ross, David Schwartz, James Sybert, Son Tran, Leo Vegoda and Will Yardley. We also thank Will Eatherton for feedback on an earlier version of the paper. Harsha Narayan and George Varghese were funded by NSF grant ANI-0074004. Ramesh Govindan was partially funded by the NSF under grant number ANI-0112649.

7. REFERENCES

[1] V. Srinivasan, G. Varghese. *Fast Address Lookups Using Controlled Prefix Expansion.* ACM Transactions on Computer Systems, Volume 19, Number 4, November 2001.

[2] M. Waldvogel, G. Varghese, J. Turner, B. Plattner. *Scalable High-Speed Prefix Matching.* ACM Transactions on Computer Systems, Volume 17, Issue 1, February 1999.

[3] David E. Taylor, Jonathan S. Turner, John W. Lockwood, Todd S. Sproull, David B. Parlour. *Scalable IP Lookup for Internet Routers.* IEEE Journal on Selected Areas in Communications, Volume 21, Number 4, May 2003.

[4] H. Narayan, R. Govindan, G. Varghese. *The Impact of Address Allocation and Routing on the Structure and Implementation of Routing Tables.* University of California,San Diego Tech Report CS 2003-0749.

[5] E. Zegura, K. Calvert, S. Bhattacharjee. *How to Model an Internetwork.* Proceedings of INFOCOM 1996.

[6] V. Paxson. *End-to-End Internet Packet Dynamics.* Proceedings of SIGCOMM 1997.

[7] C. Labovitz, A. Ahuja, A. Bose, F. Jahanian. *Delayed Internet Routing Convergence.* Proceedings of SIGCOMM 2000.

[8] C. Labovitz, R. Malan, F. Jahanian. *Origins of Internet Routing Instability.* Proceedings of INFOCOM 1999.

[9] Lightreading. www.lightreading.com.

[10] T. Bu, L. Gao and D. Towsley. *On Characterizing Routing table growth.* Proceedings of GlobalInternet 2002.

[11] M. Degermark, A. Brodnik, S. Pink. *Small Forwarding Table for Fast Routing Lookups.* Proceedings of SIGCOMM 1997.

[12] W. Eatherton *et.al. Tree Bitmap : Hardware/Software IP Lookups with Incremental Updates.* Available at www.eathertons.com/sigcomm-withnames.PDF

[13] IANA /8 delegations. www.iana.org/assignments/ipv4-address-space

[14] Policies for IPv4 address space management in the Asia Pacific region. www.apnic.net/docs/policy/add-manage-policy.html

[15] ARIN Policies. www.arin.net/policy/index.html

[16] M. Khne, N. Nimpuno, P. Rendek, S. Wilmot. IPv4 Address Allocation and Assignment Policies in the RIPE NCC Service Region. www.ripe.net/docs/ipv4-policies.html

[17] Daniel Golding. Private Communication.

[18] Mark Prior. Private Communication.

[19] FreeIPDB - The Next Generation IP Database. www.freeipdb.org

[20] Northstar IP Management Tool. www.brownkid.net/NorthStar

[21] J. J. Ramsden and Gy. Kiss-Haypal, *Company Size Distribution in Different Countries*, Physica A, vol 277, pp. 220-227, 2000.

[22] James Sybert. Private Communication.

[23] Mike Loevner. Private Communication.

[24] ARIN Allocations. ftp://ftp.arin.net/pub/stats/arin/

[25] APNIC allocations. ftp://ftp.apnic.net/pub/stats/apnic/

[26] RIPE Allocations. ftp://ftp.ripe.net/ripe/stats/

[27] University of Oregon Route Views Project www.routeviews.org

[28] O. Maennel and A. Feldmann. *Realistic BGP Traffic for Test Labs.* Proceedings of SIGCOMM 2002.

[29] Richard Jimmerson. Private Communication.

[30] Son Tran. Private Communication.

[31] Leo Vegoda. Private Communication.

[32] Ratified Policy 2001-2: Reassignments to multihomed downstream customers Policy. www.arin.net/policy/2001_2.html

[33] Gerard Ross. Private Communication.

[34] A. Lord. Proposal for IPv4 allocations by LIRs to ISPs. APNIC Open Policy Meeting, September 2002. www.apnic.net/meetings/14/sigs/policy/docs/addrpol-out-apnic-sub-alloc.pdf

[35] A. Lord. Downstream Allocations by LIRs: A proposal. RIPE 43 Meeting, September 2002. www.ripe.net/ripe/meetings/archive/ripe-43/index.html

[36] M. Ruiz-Sanchez, E. Biersack, W. Dabbous. *Survey and Taxonomy of IP Lookup Algorithms.* IEEE Network. Vol. 15. Issue 2. 2001.

[37] G. Huston. *Analyzing the Internet's BGP Routing Table.* Internet Protocol Journal. Volume 4, Number1, March 2001.

[38] Prefix Taxonomy Ongoing Measurement & Inter Network Experiment (ptomaine). http://www.ietf.org/html.charters/ptomaine-charter.html

[39] G. Huston. *NOPEER community for BGP scope control.* draft-ietf-ptomaine-nopeer-00.txt April 2002.

[40] O. Bonaventure, S. DeCnodder, J. Haas, B. Quoitin, R. White. *Controlling the redistribution of BGP routes.* draft-ietf-ptomaine-bgp-redistribution-01.txt. August 2002

[41] S. Bellovin, R. Bush, T. Griffin, J. Rexford. *Slowing Routing Table Growth by Filtering Based on Address Allocation Policies.* www.research.att.com/ jrex/papers/filter.pdf

[42] T. Bates and Y. Rekhter. *Scalable Support for Multi-homed Multi-provider Connectivity.* RFC 2260.

[43] E. Kohler, J. Li, V. Paxson, and S. Shenker. *Observed structure of addresses in IP traffic.* Proceedings of 2nd Internet Measurement Workshop, November 2002.

[44] Anindya Basu and Girija Narlikar. *Fast Incremental Updates for Pipelined Forwarding Engines.* Proceedings of Infocom 2003.

[45] D. Shah and P. Gupta. *Fast Updates on Ternary CAMs for Packet Lookups and Classification.* Hot Interconnects 8.

[46] Zhiguo Xu, Xiaoqiao Meng, Cathy Wittbrodt, Songwu Lu, Lixia Zhang. *Address Allocation and the Evolution of the BGP routing table.* Technical Report CSD-TR03009 , UCLA Computer Science Depratment, 2003. http://www.cs.ucla.edu/wing/pdfdocs/address_tr.ps