Routing Tables: Is Smaller Really Much Better?

Kevin Fall
Gianluca Iannaccone
Sylvia Ratnasamy
Intel Labs Berkeley

P. Brighten Godfrey
University of Illinois at Urbana-Champaign

ABSTRACT

We examine the case for small Internet routing and forwarding tables in the context of router design. If Moore's Law drives cost-effective scaling of hardware performance in excess of the Internet's growth, protocol modifications and "clean slate" efforts to achieve major reduction in routing table sizes may be unnecessary. We construct an abstract model for the computation and memory requirements of a router designed to support a growing Internet in light of advances being made in multi-core processor design and large, fast memories. We conclude that these advances are largely sufficient to keep the sky from falling.

1. INTRODUCTION

The size and growth of the Internet's routing tables has raised considerable attention in the last few years [1]. In research gatherings this purported problem isn't so much an area of active investigation, but rather a starting assumption that justifies various techniques such as protocol changes or alternative addressing and routing architectures aimed at ameliorating its effects.

Given the considerable interest and potentially dire implications of being caught unprepared, it is somewhat surprising that we find scant (public) discussion as to exactly why the present routing system is believed to have a scaling problem, and whether or not the size of the routing table is at fault. In this paper, we aim to take a closer look at this issue in hopes of shedding more light on whether the ability of future routers to do their job is really threatened by the routing table size growth. We take a router-centric approach, and investigate how the consequences of larger tables affect various parts of a high-end router that might be placed in the default-free zone of the Internet topology. Understanding the situation in some detail matters because there are lots of ways to design a router, and depending on the precise nature of the problem, we will either be able to ride Moore's law toward the future with a modest set of engineering changes, or fall off it and require a major overhaul, perhaps with a new addressing architecture or forcing disconnection for some segment of the population (bordering on a potential human rights issue) [2].

We begin by deconstructing a router to its abstract constituent parts, focusing on those pieces that might be impacted by scalability concerns. We then develop a framework for evaluating the scalability of each component, and use it to draw high-level conclusions about the ability of each component to scale in the future. Our evaluation uses rough calculations based on public information and some degree of discussion with industry experts. While we are not the first to discuss overall scaling trends (see [3] and [4], for example), our contribution is to examine these trends with specific regard to modern components used for implementing routers, and how network research may be guided given this understanding.

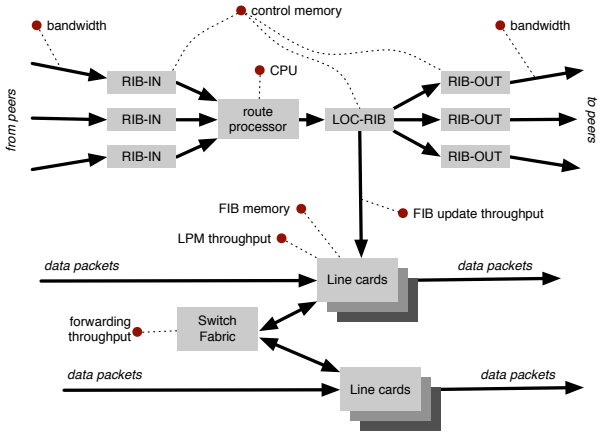

Figure 1: A model router with potential bottlenecks identified: computing, communication, and storage resources.

2. A MODEL ROUTER

There are several major components that comprise a router (or, more specifically the components of a router related to routing and forwarding of IP datagrams), as illustrated in Figure 1. There are I/O interfaces that receive and transmit packets, wiring within the router for internal communication, memory to hold tables and other state, and processors for selecting best routes and forwarding packets. It is conceivable the rate of growth in the routing table could exceed the growth capacity of the technology used to implement any of these key components. If so, we are headed for an artificial limitation on the Internet's performance, or greatly increased costs implied by the need to integrate ever more exotic technology to keep up.

Although a variety of approaches have been used, there is often a logical (or physical) separation between the *data plane* and *control plane* components of the router. In the data plane, a set of *line cards* contain the I/O interfaces and provide the interface between physical network media and the router's internal *switch fabric*. On each line card there exists a forwarding engine (or element) that executes the Internet's longest matching prefix (LPM) algorithm to direct datagrams to their selected next hops. The engines generally access fast *FIB memory* that contains the forwarding table (FIB). The engine inspects an arriving packet and determines, based on the FIB contents, which port to send the packet. Data plane components such as these must be implemented with attention to high-speed operation, as each component contributes to a packet's forwarding latency.

Logically (and often physically) separate from the data plane is the control plane, comprised of one or more *route processors* (RPs) that are responsible for determining the best next hop for each destination and programming each line card so traffic is directed to the appropriate place. RPs in core Internet routers execute a shortest path algorithm (e.g., Dijkstra), advertise and receive routing information with a set of *peers* using the *Border Gateway Protocol* (BGP), and update the FIB memory as required should a destination become reachable via a different port (or not reachable at all).

From a computer systems perspective, a router such as this is a small distributed system in the data plane and a centralized system in the control plane. Each element of the distributed system (line card) has computation (forwarding engine), memory (FIB), and I/O ports (to the switch fabric and physical network media). RPs also have computation, memory, and I/O ports. Both line cards and RPs are affected by the size and update rate of the routing table. The RPs are also directly affected by the number of peers the router communicates with.

3. CONTROL PLANE CONSIDERATIONS

In this section we focus on the control plane requirements of a BGP router. We first outline the control plane architecture, and then evaluate the implications on processing, internal communication, and storage resources implied by the execution of the BGP processing task.

A BGP router maintains a (TCP) *peering session* with each of its *neighbors* or *peers*. For each neighbor, it learns route prefixes coincident with the receipt of BGP messages. Prefix tables built using received BGP messages are stored in a RIB-IN data structure – one for each peer. When there are changes to some prefix, the available paths for that prefix are re-evaluated to select the best path. The locally-chosen best paths are then stored in the LOC-RIB. The computation produces two outputs: one containing (filtered) routing information to its downstream neighbors (stored in the RIB-OUT structures), and another containing a representation of the forwarding table (FIB), used to program the forwarding engine on a line card.

One key parameter in our evaluation is the number of neighbors that a router maintains, a number which can vary dramatically and can be adjusted by the network designer (e.g., with the use of route reflectors). In our evaluation we assume 100 neighbors. This is a relatively large value; for example, Juniper suggests that route reflectors or confederations may be used when there are more than 100 iBGP neighbors [5].

Given a fixed number of neighbors, network dynamics will drive four principal forms of overhead we are interested in: (1) the memory required to store all of the above data structures—RIB-IN, RIB-OUT and LOC-RIB; (2) the time to process an update; (3) the bandwidth to propagate updates to the FIB; and (4) the bandwidth to propagate updates to peers. We discuss each of these next.

Memory.

Assume a BGP router has p neighbors (peers). Assume each peer delivers advertisements for n prefixes. The memory required to store all the above data structures can be estimated as large as: $c_1 n(2p+1)$, because there is a RIB-IN and RIB-OUT structure for each peer and the LOC-RIB for programmling the local forwarding engines. c_1 is a constant that will depend on the data structure used to store prefix information. While we do not know exactly what data structures commercial routers use, we make what we believe are reasonable choices. At a high level, we want fast access to the information for a given prefix (so as to process updates efficiently) and low overall memory use. Very likely the former is more important than the latter given low DRAM cost. Hence, we assume prefix information is stored in an array and in addition we have a hash table indexed by prefix that points to the prefix's entry in the array. If we assume $n = 0.5M$ prefixes, $p = 100$ peers and $c_1 = 100$ bytes, then the total memory required is about 10GB. While not insignificant, this is not really a barrier for standard embedded or general purpose processor systems today. (Note this includes the worst case assumption that every peer provides a table representing the entire Internet).

Processing Time.

Processing time is tricky because we must take into account update burstiness which has long been identified as an issue (for several reasons, many tied to unusual events or bugs). Our approach is to first estimate how long processing a single update should take and then use simulation driven by a trace of actual BGP updates from Route Views [6] (RV) to estimate the impact of burstiness.

Single update processing time. To consider the processing time for a single update, we consider what fundamentally is required under – once again – reasonable assumptions about how this information is stored. What does it take to process an update for a prefix P? We must: (a) look up the path information for P from the RIB-IN for each of the p peers, (b) compare these paths, based on policy and path length considerations and (c) write the result to the RIB-OUT and LOC-RIB. (a) and (c) are dominated by memory accesses and (b) is dominated by computation.

Let c_2 denote the number of memory accesses required to access the information for a single prefix in the RIB-IN and RIB-OUT for a peer and c_3 be the number of instructions required in step (b). Then, we can estimate the processing time in terms of cycles as: $c_2 m_a p + c_3$, where m_a is the memory access time. We assume a clocks-per-instruction (CPI) ratio of 1.0 – conservative because we're assuming the data we need is never in cache and typical CPIs for compute instructions are closer to 0.5. If we once again assume $p = 100$, $c_2 = 10$, and typical $m_a = 100$ and $c_3 = 50000$ (all very conservative), then we get a processing requirement of roughly 100K cycles. Considering a single modern processing core has a 3GHz clock rate, that gives us about 33μs to process an update. This estimate is also conservative because we actually have 8 cores on a modern CPU. If we assume one can obtain a $3\times$ improvement from using 8 cores as is realistic on a modern CPU and a cache (about 12MBytes in a standard processor today), we can process an update in roughly 11μs. (Note that multi-threaded implementations of BGP have already been demonstrated in the open source community [7].)

We now compare this estimate to two benchmarking studies. Wu et al. [8] indicated an update throughput of up to 3332 prefixes/sec for a Cisco 3620 router, and up to 10000 prefixes/sec for a dual-core 3 GHz Intel Xeon running XORP, corresponding to per-prefix processing times of 300μs and 100μs per prefix, respectively. One major difference is clearly the modern hardware which we use for our estimate. In addition, might it be because the control plane measured was single-threaded? Or is it due to the age of the tested processors which lack large caches or memory bandwidth? For example, the Cisco CRS route processor card includes two 1.2GHz Freescale 7457 CPUs. These single-core processors are manufactured with a 130nm process that is four generations behind current 45nm state-of-the-art processes. They exhibit a rated maximum power consumption of around 24W. A core found in modern general purpose processors can run at a frequency of 3GHz in the same power envelope and area (and with a larger cache and bandwidth to memory).

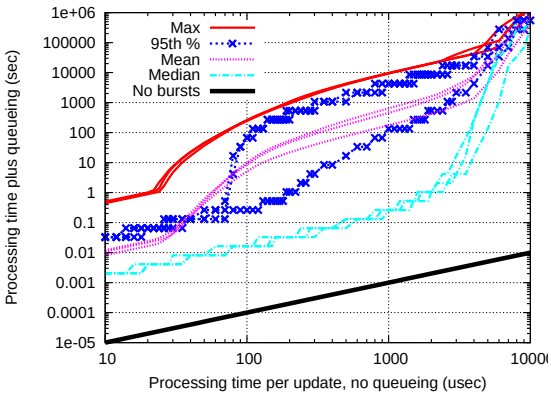

Figure 2: Latencies of BGP updates, taking into account observed burstiness, as a function of time to process one update. For each statistic, there are three lines corresponding to Dec 2008, Jan 2009, and Feb 2009. The bottom line corresponds to ideal latency with no burstiness.

More recently, a 2009 Cisco-sponsored benchmark by Isocore of a Cisco ASR1004 aggregation service router [9] showed BGP control plane convergence in 75sec given a task of processing 1 million IPv4 routes, i.e., 75μs per route. This control plane (IOS) is implemented on a dual core 2.66GHz processor, suggesting that our 11μs estimate for an 8-core, 3GHz processor may be reasonable. We will use that estimate for the next step of our evaluation.

Accounting for burstiness. To characterize bursts, we use a simple queuing model informed by RV data. The RV data set consists of a trace of BGP update messages received by a router with $p \approx 43$ peering sessions using full Internet (default-free) routing tables. We assume that all these updates are queued, and processing one update message takes x microseconds.

Figure 2 shows update processing time (including queuing) versus base processing time x (which assumes no burstiness thus no queuing). We show the median, mean, 95th percentile, and maximum time over all updates. For each of these statistics there are three lines, one for each of three months of data. Due to the volume of data, the median and 95th percentile lines are computed by putting times into factor-of-2 sized buckets; the lower end of each bucket is shown. For example, the figure shows that in the worst month, processing 95% of updates within 1-2 seconds of their arrival requires a CPU that can process an update in about 70 μsec, or a throughput of 14,286 updates per second.

We note several features of this graph. First, burstiness significantly increases the processing time of updates: even for the median update, it is more than 100$\times$ the ideal time (i.e., if the queue were always empty when an update arrived). Second, the results across the three months are quite consistent in all metrics except the 95th percentile. This disparity is due to the occurrence of significant events causing BGP session resets between the RV collector and one or more of its peers. In December 2008, such an event occurred but accounted for less than 5% of the update messages for that month, while in January and February 2009 these events accounted for more than 5% of updates.

What can we conclude from this graph? Using the processing time of $x \approx 11$ μsec estimated above, the maximum latency is under one second. But this assumes a router with ≈ 43 neighbors. We can scale the RV results up to our assumed 100 neighbors by multiplying the required base processing throughput by $\frac{100}{43}$; this corresponds to the somewhat pessimistic assumption that the burstiness of updates from the 57 additional neighbors exactly aligns with the burstiness in the dataset. In this case the maximum latency becomes ≈ 2 sec. Viewing the results another way, if we target a 30-second maximum per-router update processing time[1] then this target can be met with a present-day CPU even if update rates (or, roughly, FIB sizes) grow by 1.7$\times$. Similarly we could support a $\leq$ 30-second *mean* processing time even if update rates grow by 9.8$\times$.

In summary, there is no fundamental processing limitation imposed by CPU performance in supporting the control plane route processing. As the route processing algorithm is parallelizable, and as updates may simply be delayed if necessary, there is no hard real-time performance requirement. Instead we have a tradeoff between CPU cost (or power) and global routing convergence time. Our estimates indicate that we can achieve a reasonable point in this tradeoff space—using a single modern CPU to achieve 30-second per-router update processing time—with a 1.7$\times$ or 9.8$\times$ factor of leeway in the CPU speed or incoming update rate, depending on whether the maximum or mean convergence time is of interest. We conclude that processing time is not an immediate scaling *barrier*, but that it can be *desirable* to have much better convergence time since, for example, fast convergence improves overall network reliability.

Interface to the FIB.

We assume that the size of a message sent from the control plane to update the line cards (and FIB memory) is c_4 bytes per prefix. The information required includes the prefix (max 4 bytes for IPv4) and new next-hop information (2 bytes, as described in [10] and in the following section). Hence we can assume approx $c_4 = 10$ bytes per prefix. If we assume a per-update proccessing time t (which we estimated at 11μs above) and if we assume that updates come continually and each one requires an update to the FIB, then we require a FIB update throughput of $\frac{c_4}{t}$, which for our numbers comes to approx 7Mb/s. This rate is easily achieved with a separate control network or as a small fraction of the switch fabric bandwidth. Of course, there is the additional question of whether the lookup engine's FIB data structure can be updated at this rate, a question we examine in the following section.

Bandwidth to receive/transmit updates.

We consider a worst-case scenario to compute the bandwidth required to propagate updates from the router in question to its peers. Similar to the manner in which we computed the control-processor-to-FIB, let t be the time to process an update. Assume, pesimistically, that updates arrive continually, each requires propagation, and that each is sent as an individual message of $c_5 = 100$ bytes (i.e., there's no aggregation of multiple prefix updates into a single message). This scenario would require a worst-case update bandwidth of 24Mb/s to each peer (and this is really wost-case). For the default-free zone routers we are considering, this bandwidth requirement presents no fundamental limitation.

[1]Note that updates can already be delayed by a larger due to BGP's *Minimum Route Advertisement Interval* (MRAI) MRAI timer, which is usually set to 30 seconds.

4. DATA PLANE REQUIREMENTS

Many forwarding algorithms and data structures have been proposed for implementing the IP longest matching prefix algorithm [11]. We base our discussion on the *Tree-BitMap* algorithm (TBM) [10], for two main reasons. First, TBM does well on all the fronts that are typically of concern for IP lookups: storage requirements, update overhead and lookup times. Second, TBM is used in Cisco's flagship CRS-1 router [12] and can therefore be viewed as representative of state-of-the-art in current routers.

With TBM, the FIB data structure is in essence a very compact representation of a *multi-bit* trie; *i.e.*, a trie in which each node has an "expansion stride" of k bits. The compactness of the TBM trie is due to a novel encoding scheme that ensures all trie nodes have the same fixed size and that this size is small (we quantify node size shortly). Each TBM trie node contains two bitmaps plus two pointer values. For a TBM trie that expands k bits at each level, the first bitmap indicates which of the node's $2^k - 1$ prefixes of length less than k are present. This bitmap requires $2^k - 1$ bits. The second bitmap contains a bit for each of the 2^k possible child nodes, indicating the presence (or lack thereof) of each child node – this requires 2^k bits. The *child* pointer stores the memory location of the first child. By storing the children of a node in contiguous memory, this single pointer is sufficient to compute the pointer to any child of the node. The second *result* pointer stores the memory location of a separate array that stores the next-hop information for the prefixes present in the trie node.

The complete TBM solution introduces several optimizations to the basic structure outlined above. For the performance bounds in this paper, we assume two key optimizations from [10]. The first is an "initial array" which contains 2^i dedicated nodes corresponding to the first i address bits. This allows quick (single memory access) resolution of the first i bits for a storage cost of $s \times 2^i$ bytes, where s is the size of each entry in the initial array we define below. The second optimization is to include "skip" trie nodes used to avoid long non-branching prefix-less paths to a prefix in the basic TBM trie.

With the above assumptions, the bounds on resource requirements due to TBM are as follows.

FIB size.

A single trie node is of size $s = (\lceil \frac{2^{k+1}-1}{8} \rceil + 6)$ bytes, assuming 3 bytes per pointer. The upper bound on the total FIB memory requirements, including the initial array, the trie, and the space needed to hold next-hop information (we assume 2 bytes each) [2] can be shown to be $s \times 2^i + 2n(s+1)$ bytes, where n is the number of prefixes, i is the (optional) number of bits indexed by the initial array and k is the expansion stride for each trie node.

FIB lookup time.

The number of memory read accesses required to complete the address lookup in the FIB is $\frac{p-i}{k} + 1$, where p is the length of the prefix being looked up; i and k are as above.

FIB update time.

Updating an existing prefix – *i.e.*, either changing the next-hop information or temporarily disabling/enabling a prefix entry – involves a lookup operation for that prefix followed by a write to the corresponding trie or array entry. Hence the number of memory accesses required for each update is $\frac{p-i}{k}$ memory reads plus 1 memory write.

Adding an altogether new prefix to the FIB is more expensive, yet still bounded. TBM requires that a node's children be laid out contiguously in memory; thus in the worst case, adding a new prefix might involve rewriting the entire set of trie nodes that are peers of the new prefix. Hence in the worst case adding a new prefix can involve copying up to 2^k trie nodes (about 2^{k+1} memory accesses). Fortunately, the addition of new prefixes is a rare occurence.

[2]The authors of [10] suggest up to 16 bytes of data should be allocated for next hop information to handle features such as load balancing. We use a simpler 16 bit value to cover the needs of a basic router with up to 64K interfaces that we are considering.

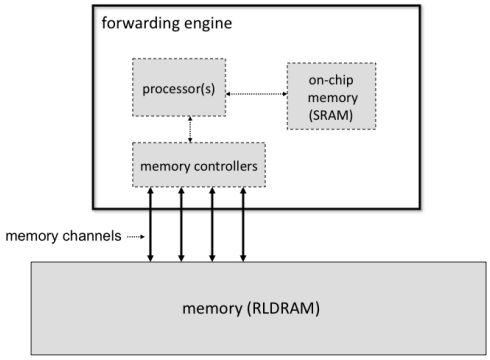

Figure 3: A forwarding engine performs packet scheduling and execution of the LPM algorithm at high speeds.

FIB update bandwidth.

As described in [10], both the FIB lookup and update operations are easily implemented in hardware; in the event of a prefix update, it is thus sufficient for the route processor(s) to just communicate the to-be-updated prefix and next-hop information to the lookup engine on the linecards (as opposed to having the route processor itself compute the updated TBM and push it out directly to the FIB memory on the linecard). Thus if we assume an average (worst-case) prefix update rate of m, then the corresponding update bandwidth from the control plane to each linecard is cm, where c is the constant number of bits required to encode the prefix and next-hop information.

In the following section, we examine the potential hardware options to support the above requirements and their resultant cost, power and performance tradeoffs.

5. IMPLEMENTATION

In the previous sections we have seen the amount of processing and storage required to support path selection in the control plane and packet forwarding and FIB updates in the data plane. To determine if Moore's Law (and the silicon industry that works hard to keep it a "law") will support the projected needs we have identified, we now explore our component options available for implementing future routers.

Figure 3 shows the layout of a conceptual line card reponsible for data plane processing. ASICS may be designed to trade off almost any power and performance rquirements at the cost of design, development and testing. ASIC manufacturing is closely tracking Moore's Law although the overall complexity and development cost may lead to a move to general purpose processors in the future. However, one aspect of the data plane that requires closer inspection is the cost of memory operations involved in moving packets in

and out of the line cards and performing the look up and forwarding operations.

We have already discussed the implementation requirements for the control plane in Section 3. In this section, we focus on the hardware requirements for implementing the data plane. We require low-latency memory for holding forwarding tables (FIBs). Random Access Memories are usually compared based upon three primary metrics: cost, density, and access time. Static RAMs (SRAMs) are the fastest and most costly memories. They are used commonly for on-chip registers and cache hierarchies and are available only in low density configurations due to their relative size. At the other end of the spectrum, Dynamic RAMs (DRAMs) offer the least cost and high densities (approx. $2/Gbit on today's spot market [13]) but have the largest access times. The gap between SRAMs and DRAMs is quite large. For example, SRAMs may have access times of 1 ns or less [14] while DRAMs are on the order of 50-60 ns (and have not decreased significantly in recent years). SRAM density is at least one order of magnitude less than DRAMs while cost can be two orders of magnitude greater.

Neither SRAMs nor DRAMs are suitable options for implementing the FIB. SRAMs are too small and expensive. When CPU cache was supported with external (SRAM) integrated circuits, the relative costs for SRAM were less. As of today, cache memory has moved almost entirely on-chip, so the market for separate high-speed SRAM components has largely dried up. They are now considered relatively exotic devices, with coincident high costs [1].

DRAMs are also not a viable option for supporting LPM lookups given the high random access latencies. On a 40 Gb/s link, (40B) packets may arrive every 8 ns and on 160Gbps, every 2 ns. Performing lookup operations at that rate with DRAM's access times of 50-60 ns would require pipelining the lookup across a large array of DRAM chips (up to 30 for 160 Gbps) accessed in parallel (and replicating the entire forwarding table). The complexity, cost and power consumption of such a design makes it unrealistic.

For these reasons, several manufacturers have produced other memory products to address the limitations of SRAMs and DRAMs for low latency applications such as packet processing workloads. *Reduced Latency DRAMs* (RLDRAMs) provide the high density of DRAMs together with faster random access times at a relatively low cost. Double Data Rate (DDR) RLDRAM (RLDRAM II) allows 16-byte reads with random access times of 15 ns [15]. Furthermore, the memory layout on chip provides eight banks to help reduce the likelihood of random access conflicts. If care is taken in organizing (and replicating) a data structure across the memory banks in such a memory, it is possible to issue up to 8 memory access in one 15 ns interval. On the cost side, RLDRAM is between 2x and 5x the cost of DRAM with memory denominations of 576 Mbit/chip.

As discussed in Section 4, the TBM implementation of the LPM technique requires traversing a multi-bit trie data structure. The number of memory accesses in the data structure per packet depends on the two parameters i and k. Using the value of $i = 8$ as suggested in [10], we can then pick a value of k based on how many memory chips and memory channels we can afford (in terms of both cost and power consumption). Increasing the size of the table requires more memory chips. Accessing multiple chips independently in parallel requires multiple memory channels (see Figure 3). For example, a value of $k = 24$ would result in one single read operation per lookup, reducing our pipeline length (i.e. a packet would only take about 15 ns to be processed). Unfortunately, doing so would also require 4 Mbytes per prefix! A more reasonable value of $k = 6$ would instead lead to 4 memory accesses per packet with a packet processing time of about 64 ns. The memory

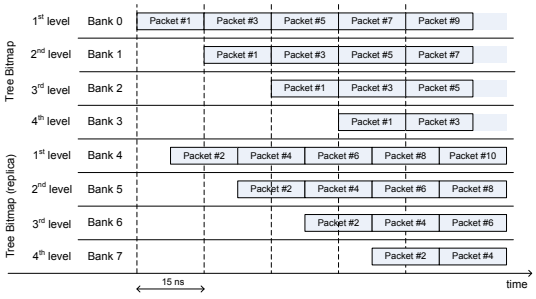

Figure 4: Example of pipelined memory accesses for lookup operations. The memory has 8 banks and an access time of 15ns. 40B packets arrive back to back on a 40 Gbps link (i.e., 8 ns interarrival).

requirement per prefix would then be a modest maximum of 46 bytes.

Given that a 40 Gbps link can receive up to 8 packets in a 64 ns interval, the memory subsystem would be required to handle 8 concurrent memory accesses. The eight-bank RLDRAM II architecture can handle such a requirement. However, the entire FIB would have to be replicated given that at any point in time two packets may be accessing the same level of the tree bitmap [10]. Figure 4 illustrates this scheme with an example. The data structure is spread across 4 banks and replicated in the other 4 banks. This way the memory chips can allow at every point in time to have two packets accessing the same area of the data structure.

The same argument can be used for a 160 Gbps link that would require 4 times more memory chips, 4 memory channels and to replicate the FIB 4 more times as well.

To put things in perspective, a FIB with 1M prefixes at 40 Gbps would require 80 Mbytes of RLDRAM II, i.e. two 576 Mbit chips. Each chip would consume less than 2W of power and cost less than $20 — negligible considering current power and cost rating of line cards. On a 160 Gb/s link, it would require 320 Mbytes, i.e., 5 memory chips.

The memory performance numbers we have seen support the conclusion that we can process packets at line rate provided we can handle multiple packets in parallel and hide memory latencies. This can be accomplished as long as the FIB is replicated in memory (2 times for 40 Gbps and 8 times for 160 Gbps). Naturally, replicating the data structure improves read performance at the cost of more expensive writes.

As described in Section 4, updating an existing prefix in the FIB results in one lookup operation plus one write on the next hop information (that resides in the on-chip memory [10]). Thus, with $i = 8$ and $k = 6$, an update operation (assuming all replicas are updated in parallel) would stop the forwarding engine and require buffering of one packet per replica while the update is in progress.

Adding or deleting a prefix is more expensive as it require to change the trie structure. With $k = 6$, each prefix would require 128 write operations per replica. Therefore, adding one prefix requires stopping the forwarding engine for the equivalent of about 128×16 ns $= 2\mu$s. Adding 1M prefixes would stop the forwarding engine for 4s. Table 1 summarizes our analysis for 40 Gbps and 160 Gbps links over a wide range of routing table sizes.

	Memory size (MB)		Cost ($)		Power (W)	
Prefixes	40 Gbps	160 Gbps	40 Gbps	160 Gbps	40 Gbps	160 Gbps
256K	20,48	81.92	3.2	12.8	0.64	2.56
512K	40.96	163.84	6.4	25.6	1.28	5.12
1M	81.92	327.68	12.8	51.2	2.56	10.24
5M	409.6	1,638.4	64	256	12.8	51.2

Table 1: Memory size, cost and power for different routing table sizes and link speeds.

In summary, FIB memory size, cost and power consumption do not seem to be valid reasons of concern for the overall scalability of Internet routers. Even the most pessimistic projections do not envision a routing table size of 5M entries within the next decade [4]. One area of concern, however, is the FIB update costs for adding or deleting prefixes (not just modifying them). Such events occurs at much slower timescales (network or customer additions happen on weekly schedules) although there may be pathological scenarios that cause a large number of prefixes to be temporarily withdrawn. In that case, one solution could be to update the trie data structure to indicate that some entries are no longer valid. More work is required to investigate efficient ways of implemeting this mechanism.

6. CONCLUSION

In this paper, we have explored whether the growth in the size or the rate of churn in the Internet's routing and forwarding tables should be a major concern to the networking community. Several research projects aim at ameliorating scaling concerns for tables and churn, yet we find no extreme urgency in this pursuit. In essence, we conclude that the current growth trajectory of the Internet's routing system will not pose an insurmountable or unaffordable challenge to implementation. That said, we do not suggest building routers is an activity devoid of scalability concerns requiring careful attention. We have focused on the scaling behavior of the classic routing and forwarding functions of a router, yet there are numerous other features a commercial router needs to support (e.g, access control lists, policies, load balancing) which we have not explored.

From a broader perspective, the viability of implementing a fast router is one necessary but not sufficient contributing factor affecting the Internet's growth. There may be other technical and non-technical factors that ultimately wield more influence. For example, concerns regarding equal access ("network neutrality"), address scarcity, and security may play more important and immediate influential roles. Fortunately, our analysis suggests, thanks largely to Moore's Law, that there will remain head-room in the component technologies used to implement network elements like routers so that if and when these other growth concerns require specific network functionality, they can be rendered at sufficient performance and levels of cost.

7. REFERENCES

[1] D. Meyer (Ed), L. Zhang (Ed), and K. Fall (Ed). Report from the IAB Workshop on Routing and Addressing, September 2007. RFC 4984.

[2] DiploFoundation. IPv6 and Its Allocation. www.diplomacy.edu/poolbin.asp?IDPool=130, Feb. 2006.

[3] Geoff Huston and Grenville Armitage. Projecting Future IPv4 Router Requirements from Trends in Dynamic BGP Behavior. In *Proceedings of ATNAC '06*. Australian Telecommunication Networks and Applications Conference, December 2006.

[4] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable Internet Protocol (AIP). In *Proc. ACM SIGCOMM*, Seattle, WA, August 2008.

[5] Juniper Networks. Managing a large-scale as. `http://www.juniper.net/techpubs/software/erx/erx41x/swconfig-routing-vol2/html/bgp-config13.html`.

[6] Route Views project. `http://routeviews.org`.

[7] Quagga Routing Suite. `http://quagga.net`.

[8] Q. Wu, Y. Liao, T. Wolf, and L. Gao. Benchmarking BGP routers. In *IEEE 10th International Symposium on Workload Characterization, 2007. IISWC 2007*, pages 79–88, 2007.

[9] Isocore. Validation of cisco asr 1000, March 2009. `http://www.cisco.com/en/US/prod/collateral/routers/ps9343/ITD13029-ASR1000-RP2Validationv1_1.pdf`.

[10] Will Eatherton, Zubin Dittia, Z. Dittia, and George Varghese. Tree Bitmap : Hardware/Software IP Lookups with Incremental Updates, 2002.

[11] H. Jonathan Chao and Bin Liu. *High Performance Switches and Routers*. Wiley-IEEE Press, 2007.

[12] http://cseweb.ucsd.edu/users/varghese/research.html.

[13] DRAM Exchange Spot Prices. `http://www.dramexchange.com`.

[14] QDR SRAM Consortium. `http://www.qdrconsortium.com`.

[15] Micron rldram memory. `http://www.micron.com/rldram`.