Projecting IPv6 Packet Forwarding Characteristics Under Internet-wide Deployment

Craig A. Shue and Minaxi Gupta
Computer Science Department, Indiana University
{cshue, minaxi}@cs.indiana.edu

ABSTRACT

While routing table growth, its impact, and causes have been examined extensively for IPv4, little work in this direction exists for IPv6. This paper is the first step at examining performance aspects of IPv6 packet forwarding. We do so by using a software implementation of various packet forwarding algorithms used by routers and running them against IPv6 tables. In the lack of a wide deployment of IPv6, we generate IPv6 routing entries based on IAB allocation recommendations. We simulate growth of routing tables due to new prefix allocations and under partial deployment scenarios. Additionally, we consider factors that inflate routing table sizes artificially. These include load balancing, multi-homing, and failure to aggregate aggregatable prefixes. We conclude that if modern routers were to simply replace their IPv4 prefixes with an equivalent number of IPv6 prefixes, without changing anything else, an average lookup in the routing table will be 67% more expensive. Further, the IPv6 routing table will require at least 4.5 times more memory to store the same number of prefixes.

Categories and Subject Descriptors

C.2.2 [Network Protocols]: Protocol verification

General Terms

Measurement, Experimentation

Keywords

Internet, IPv6, packet forwarding, routing table growth

1. INTRODUCTION

The 128-bit IPv6 [1] address space provides approximately 5×10^{28} addresses for each of the roughly 6.5 billion people on planet earth. It is thus no surprise that it is widely believed to be an answer to IPv4's address exhaustion concerns.

IPv6'07, August 31, 2007, Kyoto, Japan.

These virtues aside, we would be remiss if we tried to deploy IPv6 throughout the Internet without carefully considering aspects of routing scalability. Two reasons necessitate this. First, the routing table sizes would be bigger for IPv6, simply because of the larger address space. Second, the acceleration in factors such as load balancing, multi-homing, failure to aggregate aggregatable prefixes, and sub-optimal prefix allocations is already increasing the IPv4 routing table sizes to the point where modern router hardware may soon not be able to cope with it [2]. All of these factors, except for sub-optimal prefix allocations, are likely to exist for IPv6 and could exacerbate the issue of routing scalability.

This paper takes the first step at examining the scalability of IPv6 packet forwarding. We do so by implementing the various *longest prefix matching* algorithms used by routers in software and running them against IPv6 routing tables.

One way to generate prefixes for IPv6 routing tables would be to extrapolate current deployment. However, very few IPv6 prefixes are currently being announced in the Internet. According to the Route Views Project [3], which provides BGP routing tables from many vantage points in the Internet, the highest number of IPv6 prefix entries at any vantage point was only 807 in January 2007. Given that the number of entries in July 2003 was 468, the increase in IPv6 deployment thus far has been far from stellar. In the lack of a wide-spread deployment, we turn to the recommendations of the Internet Architecture Board (IAB) to generate IPv6 prefixes. The latest IAB recommendation is that the registries allocate IPv6 unicast addresses in /48 prefixes in the general case[1], with /64 prefixes being issued when it is known that only one subnet is required [5]. This allocation scheme allows 2^{16} subnets per prefix, if the final 64 bits are used for host identification. Under this scheme, it is unlikely that organizations will resort to address fragmentation in order to be able to expand their networks. Guided by the IAB's recommendation, we generate IPv6 prefixes by randomly picking the prefix bits. The prefix lengths are varied between 48 and 64 bits according to the Pareto distribution. This distribution captures the expected behavior that majority of the organizations will use the shortest allocated prefix possible.

To investigate scalability aspects of IPv6 packet forward-

[1]The regional registries initially made allocations in 35 bit prefixes (which were later expanded to 32 bit allocations). However, subsequent allocations to the local registries require that end users be granted 48 bit prefixes in accordance with the IAB recommendations [4]. Since these end users are the likely BGP participants, we model growth assuming prefixes of 48 bits or longer.

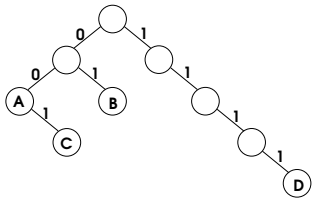

Figure 1: A traditional trie.

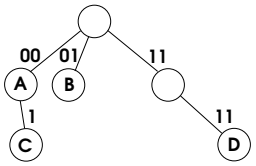

Figure 2: Multibit trie with stride length of 2.

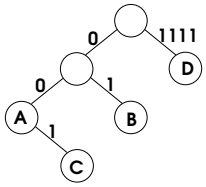

Figure 3: Path compressed trie.

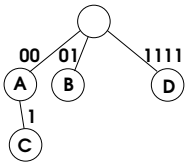

Figure 4: Path compressed and multibit hybrid trie.

ing, we consider 1) the time required to create routing tables, 2) the time required to lookup prefixes during packet forwarding, 3) the time required to update tables when entries get added or deleted, and 4) the memory requirements for holding the routing tables. We conduct our analysis on a Pentium IV 3.2GHz processor machine with 2GBytes of RAM and for three different cases. The first case projects the growth of prefixes entirely due to new prefix allocations. In the second case, we investigate the co-existence of IPv4 and IPv6 prefixes. Finally, we study the impact of factors that are causing growth in the size of IPv4 routing tables. These include load balancing, multi-homing, and failures to aggregate aggregatable prefixes.

We conclude that if modern routers were to simply replace the IPv4 prefixes in their routing tables with an equivalent number of IPv6 prefixes today, without changing the algorithms and data structures involved, an average lookup in the routing table will be 67% more expensive. Further, the corresponding IPv6 routing table will require at least 4.5 times more memory to store the same number of prefixes. We also note that the increased prefix lookup and memory costs from longer IPv6 prefixes can be minimized by using techniques that reduce memory accesses and memory required to hold the routing table by exploiting sparseness in prefix allocation. This is because the longer IPv6 prefixes are likely to be sparsely allocated in the foreseeable future.

The rest of this paper is organized as follows. In Section 2, we outline currently used approaches for IP routing. In Section 3, we discuss our performance metrics and evaluate IPv4. Section 4, presents an evaluation of IPv6. Finally, Sections 5 and 6 outline related work and conclusions respectively.

2. INTERNET PACKET FORWARDING

Routing in the Internet is made possible by the border gateway protocol (BGP). BGP allows routers in each domain to exchange reachability information about IPv4 prefixes owned by various organizations. The end result of this exchange is a forwarding table at each BGP router which contains outgoing interfaces corresponding to the prefixes. This table is referred to as the forwarding information base (FIB) for BGP routers. To forward packets toward their destination addresses, routers employ a *longest prefix match* on the prefixes contained in the FIB. This operation must be performed quickly to accommodate gigabit routing speeds. Accordingly, a variety of algorithms exist for storing and consulting the FIB [6]. Below, we outline the prominent ones.

The classical longest prefix match approach uses a trie data structure for storing the FIB. In a *traditional trie*, each node can contain next-hop and output interface information. An address lookup starts from the root node and, based on the input address, a link to a child representing a "1" or a "0" bit is traversed. During each traversal, the algorithm stores the values of the next hop and output interface information of the node, if it exists. Upon reaching a node without a required child link, the search aborts and the last recorded hop and output interface information are used. In Figure 1, we provide an example trie with four prefixes: prefix A (00*), prefix B (01*), prefix C (001*), and prefix D (1111*).

While straight-forward, the above lookup approach requires a memory lookup for each bit in the IPv4 address, yielding sub-optimal performance. To overcome this, work has explored the use of *multibit tries*. In multibit tries, each traversal can consume multiple bits of input. The number of bits consumed in each traversal is called the *stride*. Thus, instead of just having two children nodes, a trie using a stride of 2 causes each node to contain links for $2^2 = 4$ children. The choice of stride length is important; a good stride choice can increase performance but a poor stride choice may substantially increase the memory required to store the trie. Figure 2 shows the impact of using a stride of 2 on the trie from Figure 1. From this figure, we can see that the number of memory references to reach the leaves decreases. A clever implementation of multibit tries, *tree bitmap* [7], reduces the number of memory references required during packet forwarding, as well as the memory required to hold the FIB. Many router vendors today use this implementation.

An approach to optimize the traditional trie is to perform *path compression*. Such tries simply collapse one-way branches. This reduces the number of memory accesses required and limits the memory required to store the trie. PATRICIA [8] first introduced path compression. Modification were later made to the PATRICIA approach, allowing it to be used in longest prefix matching [9]. In Figure 3, we show the impact of path compression on the trie from Figure 1. The branch for prefix D is compressed to a single node, yielding faster lookups for that branch and lower memory consumption.

Path compression can be performed on multibit tries as well, including the tries that use the tree bitmap approach. In Figure 4, we show the impact of using both approaches on the trie from Figure 1.

3. BASELINE: IPv4

3.1 Methodology

We begin by implementing the trie algorithms described in Section 2 in software. We implement three different types

of tries: 1) a traditional trie, 2) a multibit trie with stride of 2, and 3) a trie using the tree bitmap approach. We examine each trie type both with and without path compression, making a total of six different types of tries. Each trie builds the forwarding table using the BGP FIB we obtained from one router in the Route Views Project [3] on April 22, 2007. The FIB contained 233,500 unique prefixes.

For each trie, we examine 1) the time required to create routing tables, 2) the time required to lookup prefix entries during packet forwarding, 3) the time required to update tables when entries get added or deleted, and 4) the memory requirements for holding the routing tables. All the performance trials were conducted on a machine with a Pentium IV 3.2 GHz processor with 2GBytes RAM. To measure the timings, we use the `RDTSC` instruction, which can be used to measure the elapsed cycle count, yielding nanosecond timing resolution.

To measure the routing table creation times, we timed how long it took to load the prefixes from a text file into the trie data structure in memory. To measure the lookup times, we randomly selected 1% of the input prefixes and recorded the amount of time required to perform each lookup. For updates, we selected 1% of the input records to be later removed and stored 1% of the input records in a list, without adding them to the trie. We then timed how long it took to delete an entry and to insert a new entry. We calculated the memory requirements for each implementation by multiplying the number of nodes required to encode the prefix entries by the size of each node.

3.2 Implementing Longest Prefix Match

Each trie must support three basic functionalities: insertion, search, and update of a prefix. Below, we describe the routines for insertion, search, and update when a single bit from the prefix is consumed at a time.

Traditional Trie: The insertion routine recursively consumes a bit of input at each node, traversing and creating children nodes as needed. Once the input has been consumed, a terminal node is created to store the outgoing interface information required to forward the packet. The lookup routine proceeds identically, except that it checks for, and records, any outgoing interface information at each node. Once the search routine runs out of matching nodes in the trie, it aborts and returns the last encountered outgoing interface information. An update is simply a deletion and insertion paired together. A deletion proceeds identically to a search, except that it removes the outgoing interface information if and only if it has an exact match after traversing the trie.

Multibit Trie: We implement a multibit trie with a stride of two. When performing a lookup, an insertion, or a deletion, the routine will use the greedy approach of using the longest stride length possible with the given input prefix. Our implementation does not use prefix expansion, but instead maintains pointers to shorter stride lengths. This allows for arbitrary prefix lengths and does not require the additional memory needed for expansion. We use a static array of pointers at each node, which results in faster lookups, but yields suboptimal memory consumption.

Tree Bitmap: We implement the tree bitmap approach described in [7]. The approach utilizes a bit vector in each node to indicate the presence of children in the tree. Each child node is then allocated contiguously in memory. The approach can access each child using a single pointer by consulting the bit vector and utilizing pointer arithmetic to reach the destination child. By reusing the same pointer, the tree bitmap approach can use longer multibit strides without increasing the amount of memory required.

Path Compression: In a trie using path compression, each node can contain multiple bits that it represents, in addition to the bits represented from its placement in the trie. Accordingly, the search and deletion routines compare these additional bits with their input. If they all match, they are removed from the input and the process continues as before. If they do not match, processing aborts as if an exact match could not be found, since the input cannot exist in the trie. The insertion routine is most affected by path compression. The insertion process stores the remainder of the input prefix each time it must create a node. The insertion routine may also need to split a node if part of the bit-string encoded within does not match the input prefix.

3.3 Results

Table 1, shows the lookup, creation, and update times, as well as the memory requirements for each of the six tries containing IPv4 prefixes. (The path compressed versions for each trie are denoted by PC.) We note that the tree bitmap approach, which is used by many modern routers [10], has the best lookup performance and the second lowest memory requirements. The path compressed version of this trie is the second best in lookup performance and has the lowest memory requirements.

	Value	Lookup Time (in ns)	Creation Time (in s)	Update Time (in ns)	Memory Required (in MBs)
Traditional	mean	2,710	0.754	6,091	19.364
	median	2,643		5,971	
	std. dev.	643		929	
Traditional, PC	mean	2,610	0.570	5,596	13.537
	median	2,631		5,650	
	std. dev.	324		697	
Multibit	mean	1,798	0.530	4,519	27.826
	median	1,779		4,416	
	std. dev.	339		912	
Multibit, PC	mean	1,714	0.390	3,797	20.615
	median	1,731		3,775	
	std. dev.	336		783	
Tree Bitmap	mean	1,125	0.442	3,632	8.031
	median	1,121		3,633	
	std. dev.	196		665	
Tree Bitmap, PC	mean	1,160	0.577	4,184	5.080
	median	1,153		3,923	
	std. dev.	214		3,361	

Table 1: IPv4 routing table results (233,500 prefixes).

4. IPv6 PACKET FORWARDING

To determine lookup, creation, update times, and memory requirements of IPv6, we repeat our analysis from IPv4.

4.1 Methodology and Implementation

We model IPv6 prefixes using the IAB recommendations. While most organizations are likely to use just one 48-bit prefix, others will want to subdivide their allocated range. Accordingly, we model prefixes from 48 bit to 64 bit in length using a Pareto distribution. We randomly generate the bits for each prefix. (The first three bits of all prefixes, "001,"

are simply to indicate that the address is a global unicast address.)

We conduct our analysis for three different cases. The first case projects the growth of prefixes entirely due to new prefix allocations. In the second case, we investigate the co-existence of IPv4 and IPv6 prefixes. Finally, we study the impact of factors that are causing growth in the size of IPv4 routing tables. These include load balancing, multi-homing, and failures to aggregate aggregatable prefixes. For each of these cases, we vary the number of prefixes to store in the IPv6 table from 50,000 entries up to 2 million entries. We select the lower-bound for the number of entries based on the observation that as much as 75% of the IPv4 entries could be a result of address fragmentation [11]. Since IPv6 is unlikely to have such a degree of fragmentation, we use a lower-bound where such entries are not present. We select an upper-bound that allows for significant growth in the number of entries, giving us an idea of IPv6 performance in the near future.

In our implementation for IPv4, we used 32-bit integers, since they were sufficient to store the prefixes. However, for IPv6, we switched to 64 bit integers, since they were needed to accommodate the longer prefix lengths.

4.2 Results

Table 2 shows a comparison of IPv4 and IPv6 results. For an even comparison with the 233,500 IPv4 entries, we pick a routing table with 250,000 entries for IPv6. Further, we present only the results for lookup times and memory requirements since creation and update times, though higher for IPv6, still fall within acceptable limits for modern routers.

We notice from Table 2 that the path compressed version of the tree bitmap approach offers the fastest lookups and lowest memory requirements. The tree bitmap approach, which is used by many modern routers [10] and had the best lookup performance for IPv4, is the second fastest in lookup time. It consumes 67% more lookup time on an average than its IPv4 counterpart. The path compressed versions of the other two tries, traditional and multibit, perform much better than the vanilla tree bitmap approach in terms of memory requirements. Specifically, the tree bitmap approach for IPv6 consumes 447.5% more memory than its IPv4 counterpart. These results indicate that path compression can effectively leverage the sparse nature of the IPv6 tries to both reduce memory requirements and the required time for lookups.

Case 1: Projecting IPv6 Prefix Growth

We now project the impact of growth in IPv6 forwarding table sizes due to new prefix allocations. As before, we focus on lookup times and memory requirements. For simplicity, we omit the traditional and multibit tries without path compression, since neither of these approaches are competitive on any count.

Figures 5 and 6 depict the lookup performance and memory requirements of our trials respectively. We note that the path compressed tree bitmap approach has the best lookup performance, followed by the tree bitmap approach and then the multibit trie with path compression. For memory requirements, the path compressed tree bitmap approach fares the best, followed by path compressed traditional trie and multibit trie with path compression respectively. Of the various tries depicted, the vanilla tree bitmap is the worst in its memory requirements. These results highlight the benefits of path compression in both memory and lookup speeds as the number of IPv6 grow.

		Lookup Time (in ns)		Memory Required (in MBytes)	
	Value	IPv4	IPv6	IPv4	IPv6
Traditional	mean	2,710	5,221	19.364	88.238
	median	2,643	5,296		
	std. dev	643	337		
Traditional, PC	mean	2,610	2,966	13.537	19.073
	median	2,631	2,951		
	std. dev	324	1,641		
Multibit	mean	1,798	3,343	27.826	109.857
	median	1,779	3,411		
	std. dev	339	285		
Multibit, PC	mean	1,714	2,038	20.843	26.746
	median	1,731	2,063		
	std. dev	336	305		
Tree Bitmap	mean	1,125	1,878	8.031	43.974
	median	1,121	1,905		
	std. dev	196	205		
Tree Bitmap, PC	mean	1,160	1,258	5.080	7.368
	median	1,153	1,278		
	std. dev	214	228		

Table 2: A comparison of IPv4 and IPv6 results (250,000 prefixes).

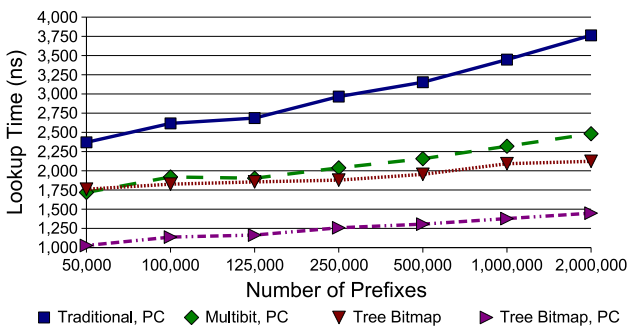

Figure 5: IPv6 lookup times under varying FIB sizes.

Case 2: Partial Deployment

Since a co-existence of IPv4 and IPv6 is likely to be the case in times to come, we now examine the lookup times and memory requirements for the case when IPv6 is deployed only in part of the Internet.

Figures 7 and 8 show the lookup times and memory requirements, respectively, of a router with FIBs for both IPv4 and IPv6. The results are shown for the case when the total number of combined prefixes are 200,000 in number. Once again, the path compressed tree bitmap approach performs the best in terms of lookup times and memory usage. The vanilla tree bitmap trie fares the second best in terms of performance, but the path compressed traditional trie is second best in memory usage. Also, as expected, the lookup times and memory requirements are greater when IPv6 accounts for 75% of the entries than when it accounts for only 25% of the entries. However, the behavior in the middle of the graphs is interesting for each of the tries: the lookup times and memory requirements level off as the proportions of the two protocols become equal and have a local minimum at

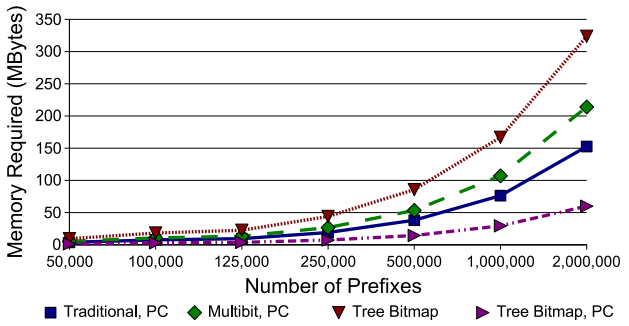

Figure 6: IPv6 memory requirements under varying FIB sizes.

60% IPv6 deployment. This is likely the result of IPv6 having better properties at lower levels of deployment combined with the decreased role of IPv4.

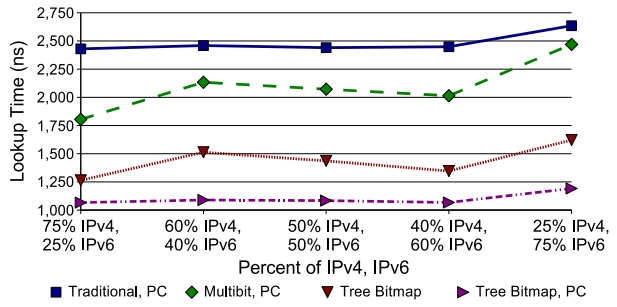

Figure 7: Lookup times when IPv4 and IPv6 contribute various percentages of the FIB (200,000 prefixes).

Case 3: Impact of Deaggregation on IPv6

We now consider the impact of other factors that cause the number of prefix entries in the routing tables to increase. In particular, we consider the three prominent factors, namely, load balancing, multi-homing[2], and failure to aggregate aggregatable prefixes. For exposition purposes, we label this collection as *deaggregation contributors*.

We first develop a set of simple algorithms to simulate these deaggregation contributors. To model load balancing, we split an existing prefix in half and announce a new, more specific route for both halves. For multi-homing, we take an existing prefix, randomly select a sub-prefix that fits inside the original prefix, and add both prefix entries. This models the case where a subset of an address range must be stored separately, since it can arrive through multiple routes. For failure to aggregate, we take a given prefix and create an identical prefix with just the last bit toggled, which models a case where two prefixes could easily be aggregated, but are not.

We take two randomly generated routing tables from the previous section, one with 100,000 prefixes and one with

[2]While Shim6 [12] can be used to avoid routing table growth due to multihoming in IPv6, it is difficult to predict Shim6's adoption. Accordingly, we choose to model multihoming growth.

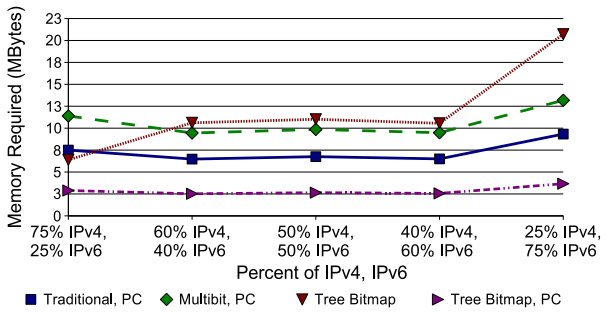

Figure 8: Memory required when IPv4 and IPv6 contribute various percentages of the FIB (200,000 prefixes).

200,000 prefixes, and apply the set of algorithms to model the deaggregation contributors. We model the cases where each algorithm is applied to a random 10%, 20%, and 30% of the entries in the table. We use a random sampling because it models the fact that prefix assignment (and thus the prefixes included in route announcements) is determined independently from the decision to employ traffic engineering (or failures to properly aggregate prefixes).

In Figure 9, we show the impact of adding entries due to the deaggregation contributors to the lookup times on our 200,000 entry table. (The results from the 100,000 entry table were similar and have been omitted for conciseness.) For each trie type, we observe increases in the lookup times as the percentage of deaggregation increases. However, the lookup times for the vanilla tree bitmap approach appears to be less affected by the increased deaggregation. Also, in each case, the tries perform as well as before relative to each other. The memory requirements follow the same trend as lookup times. We omit those results due to space constraints.

Due to space constraints, we do not show how these results compare with the case when randomly picked prefixes were added to the table, rather than picking entries specific to deaggregation contributors. For most tries, tries with randomly picked entries fared better.

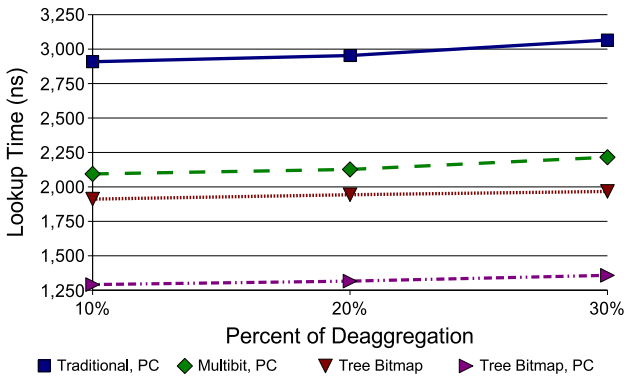

Figure 9: Impact of deaggregation on lookup times of IPv6 tables (200,000 prefixes).

5. RELATED WORK

IPv6 specification has been the subject of a number of IETF RFCs which specify the details of the protocol. The latest is [13], which specifies the addresses, and address allocation. Additionally, in [5], the IAB gave advice to the registries on allocating globally aggregatable prefixes.

In [11], the authors explore the impact of multi-homing, load balancing, address fragmentation, and simple aggregation failures on the size of BGP routing tables in IPv4. In [14], the authors characterize IPv4 allocations and compare them with BGP routing table changes. This work observes that 45% of the originally allocated blocks were split into smaller fragments. We have leveraged both of these works to make projections into IPv6 behavior. In [15], the authors develop a detailed model of routing table growth and validate it using backbone IPv4 routing tables. Unfortunately, we cannot perform a similar validation due to a lack of widespread IPv6 deployment.

IPv6 is one next-generation Internet approach to expand the address space available for hosts. Other approaches include IPNL [16], in which the authors propose to solve address exhaustion by making network address translation (NAT) a first class citizen, and ROFL [17], in which the authors demonstrate that routing on flat labels cannot be casually dismissed.

Performing fast longest prefix match lookups has been an active topic in research for decades [8, 9, 7, 18]. Some work has specifically focused on hardware implementations [19], with some particularly addressing IPv6 [20]. While relevant, these works focus on increasing lookup performance while our goal is to compare the performance of IPv4 and IPv6 on some representative algorithms and determine whether IPv6 can scale under larger routing tables.

6. CONCLUSION

It is generally accepted that routers will take longer to forward IPv6 packets, and that the routing tables under IPv6 will get bigger. However, the extent of this degradation had not been explored. In this work, we attempted to quantify the performance hit the routers may experience under IPv6. Our results also show that using path compression techniques can reduce this performance overhead by making the lookups less dependant on the prefix length. We note that the tree bitmap approach used by many modern routers [10], which yields the best performance and memory usage in IPv4 does not fair so well with memory usage in IPv6. However, when we modified the algorithm to use path compression, both its memory usage and lookup performance improved. This combined approach has the potential to make the performance of IPv6 competitive with IPv4.

Finally, while we tried our best to project IPv6 deployment using the latest recommendations, actual prefix allocations may be different. There is also a possibility that the number of entries in IPv6 routing table may be far fewer than what today's IPv4 routing tables contain. This could happen if some or all of the factors that inflate routing table entries today cease to exist. Short of knowing what might happen, we used similar number of IPv4 and IPv6 entries for comparison purposes. However, one cannot rule out the possibility that the performance overheads of longer IPv6 prefixes may be offset by fewer entries.

7. REFERENCES

[1] "IPv6 information page," http://www.ipv6.org.

[2] D. Meyer, L. Zhang, and K. Fall, "Report from the IAB Workshop on Routing and Addressing," IETF Internet Draft, December 2006.

[3] U. of Oregon Advanced Network Technology Center, "Route views project," http://www.routeviews.org/.

[4] A. APNIC and R. Registries, "Ipv6 address allocation and assignment policy," June 2002, http://www.arin.net/policy/archive/ipv6_policy.html.

[5] IAB and IESG, "IAB/IESG Recommendations on IPv6 Address Allocations to Sites," RFC 3177, Sept. 2001.

[6] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, 2001.

[7] W. Etherton, Z. Dittia, and G. Varghese, "Tree Bitmap: Hardware/Software IP Lookups with Incremental Updates," *ACM SIGCOMM Computer Communication Review*, 2004.

[8] D. Morrison, "Patricia - practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM*, 1968.

[9] K. Sklower, "A tree-based packet routing table for Berkley Unix," USENIX, 1991.

[10] G. Varghese, "Recent research directions," 2004, http://www-cse.ucsd.edu/users/varghese/research.html.

[11] T. Bu, L. Gao, and D. Towsley, "On characterizing BGP routing table growth," *Computer Networks*, vol. 45, no. 1, pp. 45 – 54, May 2004.

[12] E. Nordmark and M. Bagnulo, "Shim6: Level 3 Multihoming Shim Protocol for IPv6," May 2007.

[13] R. Hinden and S. Deering, "IP Version 6 Addressing Architecture," RFC 4291, Feb. 2006.

[14] X. Meng, Z. Xu, B. Zhang, G. Huston, S. Lu, and L. Zhang, "IPv4 Address Allocation and the BGP Routing Table Evolution," *ACM SIGCOMM Computer Communications Review*, 2005.

[15] H. Narayan, R. Govindan, and G. Varghese, "The Impact of Address Allocation and Routing on the Structure and Implementation of Routing Tables," in *ACM SIGCOMM*, 2003.

[16] P. Francis and R. Gummadi, "IPNL: A NAT-extended Internet architecture." ACM SIGCOMM, 2002.

[17] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker, "ROFL: Routing on flat labels." ACM SIGCOMM, 2006.

[18] H. Song, S. Dharmapurikar, J. Turner, and J. Lockwood, "Fast hash table lookup using extended bloom filter: An aid to network processing," in *ACM SIGCOMM*, 2005.

[19] D. Taylor, J. Lockwood, T. Sproull, J. Turner, and D. Parlour, "Scalable IP lookup for programmable routers," INFOCOM, 2002.

[20] T. Hayashi and T. Miyazaki, "High-speed table lookup engine for IPv6 longest prefix match," GLOBECOM, 1999.