Packet Forwarding: Name-based Vs. Prefix-based

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

Abstract— **Using domain names for routing, instead of IP prefixes, has the potential to address many of the core outstanding issues in today's Internet. To initiate research in that direction, this paper compares the performance of name-based routing in the core of the Internet with that of IPv4 routing. Our analysis concludes that name-based routing is well within the scope of feasibility.**

I. Introduction

Users of popular Internet applications specify service end points using human-friendly domain names. The domain name service (DNS) resolves these domain names into IP addresses and the underlying communication subsystem uses only the IP addresses to deliver data. This setup has worked well so far. However, today, the unallocated IPv4 address space is scant and the DNS infrastructure is vulnerable to many types of financial and security attacks, including denial-of-service (DoS) and phishing.

Solutions to address both of these concerns have been proposed. IPv6 [1] proposes to solve the address exhaustion problem. However, it is not clear how scalable it would be under multi-homing, traffic engineering and sub-optimal prefix allocations [2]. DNSSEC [3] proposes to address security issues related to DNS. Its adoption has been hindered in part due to the lack of key distribution authority in the Internet.

This paper takes a fresh approach to solving address exhaustion and DNS security-related concerns. It envisions a future Internet that replaces the IP-based addressing and routing in the Internet with one where hosts are identified only by their domain names and the routing subsystem forwards packets based on domain names. (Subsequently, we refer to the latter scheme as *name-based routing*.) Using the widely-accepted domain names as host identifiers has the advantage that the end users do not have to be concerned with aspects of Internet evolution. This is important to make transition to the new scheme practical. If adopted, name-based routing would have the following impact:

Address space: The domain names are infinitely expandable in practice. Thus, address space exhaustion concerns will be alleviated.

DNS infrastructure: A translation from domain names to IP addresses would no longer be required, eliminating the need to have the DNS infrastructure[1]. Thus, all the DNS-related security attacks will be eliminated.

Provider switch: Currently, IP addresses serve both as identifiers as well as locators, making it hard for organizations who lease network prefixes from their providers to change providers. Since domain names are provider independent, this restriction will be eliminated under name-based routing.

Many challenges need to be addressed before name-based routing can become a reality. First, the IP header will have to be redesigned such that packets can contain domain names instead of IP addresses. Second, the routing protocols will also have to be redesigned to exchange domain names instead of IP prefixes. Third, scalability aspects of name-based routing tables and forwarding speeds will have to be considered. Fourth, support for multi-homing, mobility, and advanced services, such as multicasting and anycasting, will have to be provisioned. Finally, since a transition to the new scheme cannot occur overnight, issues in backward compatability would have to be carefully examined.

In this paper, we take a first step at investigating the feasibility of name-based routing. Our focus is primarily on comparing the performance of name-based packet forwarding with modern IPv4 packet forwarding. Specifically, we evaluate the feasibility of name-based routing in terms of the time required to create, look up, and update routing tables in the core of the Internet, and the corresponding storage requirements.

Toward our goal, we use data from the DMOZ Open Directory Project [4], which contains user submitted links, and the Route Views Project [5], which makes route announcements available to the research community. We implement various *longest prefix algorithms* used by IPv4 routers in software. The analysis produces encouraging results. While the lookup, creation, and update times of IPv4 are faster than name-based routing, the name-based routing results are of the same order and certainly not beyond the scope of feasibility. The biggest obstacle for name-based routing is the size of the routing table, which requires several orders of magnitude more storage than the corresponding IPv4 tables. We explore the viability of caching to reduce the number of entries in the routing table and also highlight an approach to perform domain aggregation to further reduce the number of entries.

The rest of this paper is organized as follows. In Section II, we outline currently used approaches for IP routing. In Section III, we discuss name-based routing and analyze its performance. Section IV discusses approaches to optimize the memory required for the name-based routing. Finally, Sections V and VI outline related work and open issues respectively.

II. Background

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 prefixes contained in the FIB. This operation must be performed quickly to accommodate gigabit

[1]Routing would still have to be secured. This issue would remained unchanged from today.

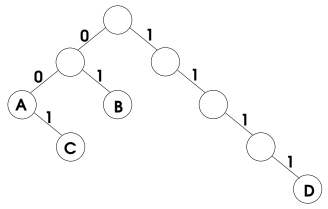

Fig. 1. A traditional trie.

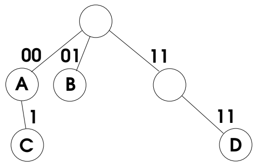

Fig. 2. Multibit trie with stride length of 2.

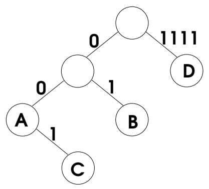

Fig. 3. Path compressed trie.

routing speeds. Accordingly, a variety of algorithms exist for fast longest-prefix matches [6]. Below, we outline the prominent ones.

The classical longest prefix match approach uses a trie data structure. 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 required to reach the leaves decreases.

Another approach to optimize the trie data structure is perform path compression. Tries employing path compression, *path compressed tries*, simply collapse one-way branches. This reduces the number of memory accesses required and limits the memory required to store the trie. PATRICIA [7] first introduced path compression. Modification were later made to the PATRICIA approach, allowing it to be used in longest prefix matching [8]. 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.

III. Name-based Routing

To route on DNS names instead of IPv4 addresses, inter-domain routers would have to maintain an equivalent of a border gateway protocol (BGP) FIB. We refer to this table as the *name-based routing table* subsequently and routers employing this table as *name-based routers*. For common cases, it is sufficient that this table for core Internet routers contain an entry for 1) each DNS second-level domain, e.g., "university.edu" and 2) each third-level domain for domain names that contain countries as the top level domains (TLDs), e.g., "university.ac.in", along with their corresponding outgoing interfaces. For simplicity of subsequent description, we refer to all entries of the name-based routing table as *domain names*. Notice that finer granularity domains names, e.g., "cs.university.edu", do not need to be exist in name-based routing tables for core Internet routers since they can be taken care of by the intra-domain routing. To forward packets toward their destination, name-based routers will use the domain name of the destination and perform an equivalent of today's longest prefix match on the name-based routing table.

We compare the performance of IPv4 routers with name-based routers for traditional and path compressed tries. We leave out multibit tries from our comparison because an even-handed comparison is hard to do when the optimal stride sizes differ, which is likely to be the case because IPv4 prefixes and domain names have fundamentally different characteristics.

A. Test Data

In order to model realistic name-based routing tables, we collected data from the DMOZ Open Directory Project [4]. The project contains user submitted links and is the largest and most comprehensive directory of the Web. Our input data, collected on October 28, 2006, has $9,633,835$ unique URLs and $2,711,181$ unique second and third-level domain names, as described earlier. We compare this data with the July, 2006 results from the Internet Systems Consortium (ISC) Internet Domain Survey [9]. The ISC data indicates there are $3,105,760$ second-level domains. Thus, our data includes approximately 73.38% of the second-level domains. This gives us confidence that we are working with a representative sample of the Internet's domains. For comparison with IPv4 routing tables, we obtained a BGP FIB from one router in the Route Views Project [5] on November 15, 2006. The FIB contained $155,854$ entries, fewer than expected, possibly because the chosen vantage point does not have all the announced IPv4 prefixes. As a result, the performance of IPv4 that we measure is actually slightly better than it would be with complete records.

B. Implementation of Longest Prefix Match Algorithms

We begin by parsing the links contained in the DMOZ data into DNS host names. We then aggregate these host names into domain entries, which are used to populate both traditional and path compressed tries. To do so, we use a simple heuristic, in which generic TLDs are grouped by their second-level domains and most country code TLDs are grouped by their third level TLDs. Some country codes have second level domains, in which case an individual host name is considered to be a domain, introducing a small overestimate in the number of domains if there are multiple hosts in the same domain in our data.

In each of the trie implementations, we hierarchically reverse the DNS names when storing entries and when performing lookups. For example, "www.university.edu" is translated to "edu.university.www." This allows us to take advantage of the hierarchical structure of DNS names to obtain better branching.

The BGP FIB from Route Views is also parsed into AS-specific prefixes, which are then used as input to the corresponding traditional and path compressed tries. Next, we describe the implementation of various tries.

1) Traditional Trie: The trie should support three basic functionalities: insertion, search, and update. In the case of a name-based routing table, the unit of insertion, search, and update is a domain name while for a IPv4 FIB, the unit is a prefix. The names are made up of 37 characters, 0-9, A-Z, and a '-' (the '.' is treated as a special value) while the prefixes can only be made of bits '0' and '1'. The subsequent discussion describes the routines for insertion, search, and update in a name-based routing table where a *character* is consumed at a time. The traditional trie for IPv4 is populated similarly except that a *bit* is consumed at a time and bit comparisons are used instead of character comparisons.

When storing an entry, the insertion routine recursively adds one character at a time from left to right, starting at the root. At each hop, the routine finds the child node that matches the first character in the input domain name. The insertion routine then removes the first character of the input and recursively calls itself using the child node as the new insertion point. Upon encountering a null child, the insertion routing creates a new node for the child, inserts it into its parent node, removes the first character of input, and recursively calls itself. Once all the input has been consumed, the next hop and output interface are stored at a terminal node off the last child.

The search routine also proceeds recursively, consuming a character of input at each hop. In the name-based approach, the search routine checks for the existence of the next-hop and output interface information at each "." entry and records it if it exists. In the IPv4 approach, the search routine checks for next-hop and output interface information at every hop. Upon encountering a null child, the search process aborts and returns the next-hop and output interface it last recorded.

An update is simply a deletion and insertion paired together. The deletion routine proceeds similarly to the search routine. Upon encountering a null child, the deletion process aborts without changing the structure, since no exact match is found in the structure. When the deletion has consumed all of the input data, the deletion routine removes the next hop and output interface information from the current node. The routine then completes.

When looking at a traditional trie analytically, we note that the worst case lookup time is O(L), where L is the length of the input. This is because the trie traversal is based on this length, consuming one character at each node. Similarly, the worst case for an update is O(L). The memory requirements are O(L*N), where N is the number of entries than must be stored.

2) Path Compressed Trie: In a path compressed trie, each node can contain multiple characters or bits that it represents, in addition to the characters/bits represented from its placement in the trie. Accordingly, the search and deletion routines compare these additional characters/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 a null child was encountered, since the input cannot exist in the trie.

The insertion routine is most affected by path compression. Upon encountering a null child when inserting, the insertion creates a new node, stores the remainder of the input in it, and stores the next hop and interface information. Additionally, if the insertion encounters a node, node A, which is storing multiple characters, it attempts to match its entry with the stored characters/bits. Upon finding characters/bits that do not match, the stored character/bit string is split. The matching characters/bits are retained in node A. Two new nodes are then created: one for the remaining part of the split string, node B, and one for the rest of the input in the entry being inserted, node C. All of the children on node A are then moved to node B. Nodes B and C are then added as children on node A. This process of building the trie takes advantage of compression whenever possible while avoiding any special compression heuristics.

When looking at a path compressed trie analytically, we again note that the worst case lookup and update times are O(L), where L is the length of the input. However, the memory requirements are O(N), where N is the number of entries that must be stored. Note that the storage requirements are independent of the input length, since the entire input can be compressed into a single node.

C. Comparison with IPv4

To compare the performance of name-based routing with IPv4 for both traditional and path compressed tries, we examined for each approach: 1) the time required to create routing tables, 2) the time required to lookup entries during packet forwarding, 3) the time required to update tables when entries get added or deleted, and 4) the storage requirements for 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.

1) Routing Table Creation Times: In Table I, we show the average time required to create the name-based routing table which had $2,711,181$ entries and the IPv4 FIB, which had $155,853$ entries. We make comparisons both for traditional and path compressed tries. Though the name-based routing tables take orders of magnitude more time to load, these times are unlikely to impact forwarding speeds since the tables typically need to be loaded only every few minutes.

	Traditional Trie	Path Compressed Trie
Name-based	24.383	19.231
IPv4	0.612	0.384

TABLE I

AVERAGE ROUTING TABLE CREATION TIMES (IN SECONDS).

The ISC Internet Domain Survey indicates that there has been a growth of roughly $50,000$ second-level domains every six months over the last 3 years. If this trend continues, there will be roughly 4.25 million second-level domains in January, 2018. This time-frame seems sufficiently large to determine the scalability of our approach. Projecting to 4.25 million domains, we find that the creation time for the name-based routing table becomes 40.86 seconds for the traditional trie and 31.73 seconds for the path compressed trie. Both of these times fall well within the typical update times for modern routers. We conclude that routing table creation times are a non-issue for name-based routing.

2) Lookup Times: To determine lookup performance, we searched a randomly sampled 1% of the unique domains for both traditional and path compressed tries. Table II shows the results for the name-based routing table and the IPv4 FIB. Though lookups in the name-based tables cost more than IPv4 lookups for both types of tries, they are of the same order.

	Traditional Trie			Path Compressed Trie		
	Avg	Min	Max	Avg	Min	Max
Name-based	6,842	2,070	54,140	6,460	2,290	51,238
IPv4	2,618	1,070	10,455	2,525	910	5,283

TABLE II
LOOKUP TIMES (IN NANOSECONDS).

Next, we looked at the distribution of lookup times obtained above. The cumulative distribution functions (CDFs) of the lookup times are shown in Figure 4. These CDFs indicate that the average lookup times for name-based routing are worse than those for IPv4 because a larger percentage of lookups for IPv4 finish in a small amount of time. In particular, 50% of the name-based lookups take $4,531$ nanoseconds or less and about 90% of the lookups take $7,656$ nanoseconds or less. For IPv4, the corresponding percentage of lookups take $2,656$ nanoseconds or less and $2,969$ nanoseconds or less respectively. Thus, name-based routing can benefit from optimizing lookup times for popular entries.

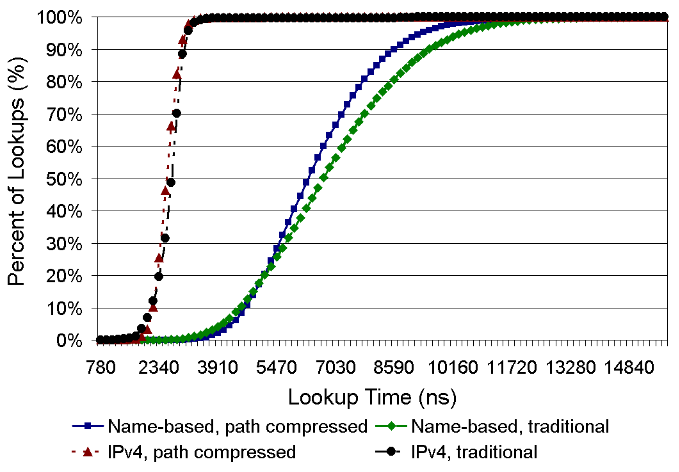

Fig. 4. CDFs for distribution of lookup times for name-based and IPv4 approaches.

Projecting the average lookup times to 4.25 million second-level domains projected by the ISC Internet domain survey in January, 2018, we get $7,629$ nanoseconds for traditional trie and $7,644$ nanoseconds for the path compressed trie. These numbers seem to indicate that the lookup times do not increase in proportion to the number of entries in the name-based table.

3) Updating the Routing Table: Next, we observed the times required to update the routing tables. We randomly updated 1% of the routing table entries. Table III shows the results for the name-based routing table and the IPv4 FIB. Though updates in the name-based tables cost more than IPv4 lookups for both types of tries, they are still reasonable, since updates occur much less regularly than lookups.

	Traditional Trie			Path Compressed Trie		
	Avg	Min	Max	Avg	Min	Max
Name-based	16,013	5,790	41,508	14,603	7,425	36,520
IPv4	5,949	3,365	19,810	5,358	3,808	14,855

TABLE III
UPDATE TIMES (IN NANOSECONDS).

When projecting the average update times to 4.25 million second-level domains, we get $16,749$ nanoseconds for traditional trie and $16,529$ nanoseconds for the path compressed trie. These numbers again seem to indicate that the update times do not increase in proportion to the number of entries in the name-based table.

4) Memory Requirements: To determine the amount of memory required to store the name-based routing table, we multiplied the number of entries by the size of each entry. The first two columns of Table IV show the storage requirements of the name-based routing table and the IPv4 FIB. Clearly, the path compressed tries fare much better for both name-based and IPv4 tables and the memory requirements of the name-based routing table are very demanding.

	IPv4	Name-based (full trie)	Top 16% of name-based entries	Top 4% of name-based entries
Traditional Trie	13.0	400.8	83.2	19.5
Path Compressed Trie	8.9	163.7	29.1	7.2

TABLE IV
COMPARISON OF STORAGE REQUIREMENTS (IN MBYTES).

When projecting the storage requirements to 4.25 million second-level domains, we get 642.18 MBytes for traditional trie and 264.11 MBytes for the path compressed trie.

IV. OPTIMIZING MEMORY REQUIREMENTS FOR NAME-BASED ROUTING TABLES

The greatest difficulty for the name-based approach seems to be the storage requirements, which are two orders of magnitude greater than IPv4 for both traditional and path compressed tries. As a result, the memory required to store the entire name-based trie may be too much to fit into the faster SRAM on routers. Previous work indicates that a significant portion of traffic is destined to a small subset of destinations [10], [11], [12]. Guided by this observation, we now explore the trade-offs of storing high usage domains in fast memory and using the slower memory, such as DRAM, for cache misses.

We estimate domain popularity using the number of times a domain appears in the unique URLs contained in the DMOZ data. For each link, we determine the domain associated with it. We then add the number of times each domain appears in the list and take this as an indication of domain popularity. As shown in Figure 5, the DMOZ data revealed a heavy tail distribution. In particular, the top 4% most popular domains account for 35.75%

of URLs and the top 16% most popular domains account for 50.06% of URLs.

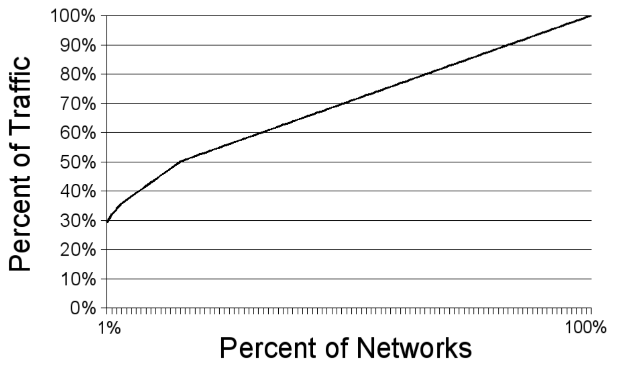

Fig. 5. Cumulative distribution function for domain popularity.

To evaluate the performance of caching tries corresponding to popular domains, we modify our code to include a smaller, cached trie as well. Entries are either inserted into the cached trie or the regular trie, depending on its popularity. Lookups and updates are first conducted on the cached trie and proceed to the regular trie only if no match is found in the cached trie.

We perform these tests for both path-compressed and traditional tries. We experimented with two cases: when the smaller trie contains 16% of the most popular unique domains and when it contains 4% of the most popular unique domains. Table IV shows the storage requirements for the caches. The cached tries containing 4% entries come very close to the corresponding traditional IPv4 tries. For the path compressed trie, the cached trie with 4% entries is well within the bounds of the SRAM in modern routers.

As shown in Figure 6, caching 4% of the entries reduces the lookup times for more than 60% of the lookups. Caching 16% of the entries does not yield as great results, indicating costs of traversing a larger cache trie offset the higher cache hit percentage. These caching benefits come at the cost of increased lookup times for the less popular domains, since they must look through two tries. We note this analysis is all performed in software and DRAM, which does not show the advantage of caching in higher speed memory. As future work, we will examine caching in hardware with faster memory.

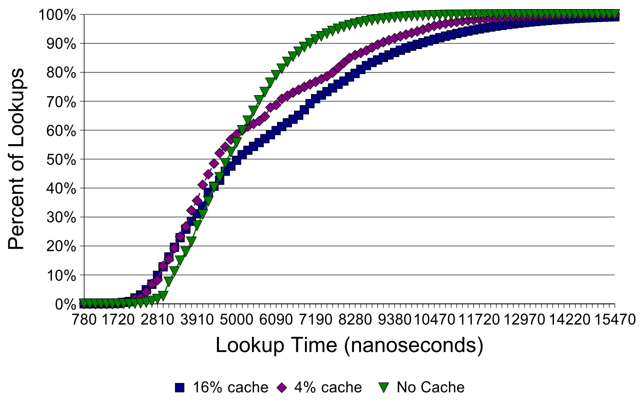

Fig. 6. Comparison of CDFs for lookups in name-based path-compressed tries with and without caching.

A. *Future Work*

Our present caching analysis only accounts for link popularity, which is different from the amount of traffic destined to a particular domain. Also, traffic other than web traffic is not taken into account. To overcome these limitations, as part of future work, we plan to analyze Netflow [13] traffic logs from the Internet2 backbone to determine the distribution of traffic volume across domains.

Thus far, we have assumed that individual domains are stored in the name-based routing table. In practice, multiple domains are often co-located in the same network. In fact, previous work indicates that more “than 87% of active domain names are found to share their IP addresses ... with one or more additional domains, and more than two thirds of active domain names share their addresses with fifty or more additional domains” [14]. Accordingly, groups of domains can be aggregated together into a single identifier. This observation can significantly reduce the number of entries in the core routing tables. We plan to investigate the benefits of this compaction as part of our future work.

V. Related Work

Many research works have been considering alternate Internet architectures. In TRIAD [15], the authors focus on content routing, a much different goal from our own. However, to transparently cache content, the architecture requires a subset of routers to store forwarding tables based on host names. TRIAD differs from our approach in that the host names are only used during connection establishment, not for actual data packets. In IPNL [16], the authors propose to solve the Internet address exhaustion issue by making network address translation (NAT) a first class citizen. They leave the core of the Internet unchanged and instead propose to change the edge networks to route on fully qualified domain names (FQDNs) during connection establishment. In contrast to ours, their scheme leaves the DNS infrastructure unchanged. By allowing the core of the network to change, our approach will eliminate the DNS infrastructure as well.

Many other works have focused on outstanding Internet issues, including address exhaustion and mobility. IPv6 [17] was designed mainly to increase the address space available to end-hosts. In the face of traffic engineering, multi-homing, and prefix de-aggregation, its scalability remains questionable [2]. FARA [18] and i3 [19] focus on mobility and utilize rendezvous mechanisms to facilitate communication. In Nimrod [20], the authors propose an architecture to support varying quality of service requirements and restrictions. In HIP [21], the authors use public key cryptography to create secure identities. In Layered Naming [22], the authors use separate identifiers to distinguish between services and hosts, allowing for delegation of duties, which benefits load balancing. Finally, in ROFL [23], the authors demonstrate that routing on flat labels cannot be casually dismissed, even if the current performance is inferior to IPv4 routing. Our approach differs from ROFL in that we propose to route on DNS names, eliminating the need for the DNS infrastructure. ROFL on the other hand, will require a DNS-equivalent in order to allow human-friendly host identifiers.

VI. Discussion

Our goal in this paper was to determine the feasibility of routing on DNS names. Through a software implementation of longest prefix match forwarding algorithms, we determined that while modern IPv4 routing has better performance, the performance of our name-based routing implementation is of the same order of magnitude, indicating it may be a feasible approach for routing. Our results encourage further exploration into the named routing realm.

In our analysis, we considered effective second level domains to estimate the size of routing tables for core name-based routers. Implicitly, this assumed one entry in the routing table per organization. While this assumption is true for most domains, some domains may be an exception. In particular, a large international company may have several sites. For example, `company.com` may be an entry that goes to the company's corporate headquarters, while `uk.company.com` may be a route to the company's site in the United Kingdom. The effect of this de-aggregation is similar to the de-aggregation in the CIDR approach for IPv4. Effectively, there would be more entries in the name-based FIBs, which would increase table sizes. (Forwarding such entries is a non-issue because the tries already support longer prefix match.) We note that the domain aggregation approach we mentioned in Section IV-A would have the opposite effect on the size of name-based FIBs. The net effect of the extra entries and domain aggregation on the size of name-based routing tables remains to be explored.

There are several other open issues. A redesign of the routing protocols and IP header is required to enable a transition to the proposed name-based scheme. Partial deployment scenarios necessitate using legacy infrastructure between deploying sites. This could be accomplished by adding a *name-layer* on top of the IPv4 header. The issue of encoding domain names in the network layer would also require careful consideration. Clearly, DNS host names are longer than IPv4 addresses and encoding them in each packet's header will cause the packet header size to increase. However, the extra overhead may not be worse than IPv6. This can be seen in Figure 7, where we plot a CDF of the character length of the full host names from our DMOZ data set. We note that 99.59% of host names are 36 characters or shorter. Further, 67.62% are 21 characters are shorter. Since each domain name character can be encoded in 6 bits, a 21 character name would require only 15.75 bytes whereas an IPv6 address would require 16 bytes. If an efficient variable-length encoding were used, it is possible that the name-based headers would actually be shorter than IPv6 for the majority of traffic.

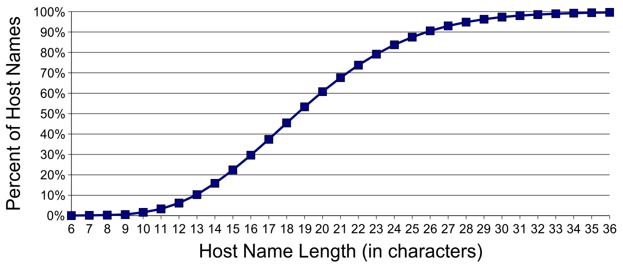

Fig. 7. CDF of the percentage of hosts with given number of characters.

Further, our paper leverages the limited character set in DNS, allowing our nodes to have a set branch factor of 37 within labels (26 letters, 10 digits, and the "-" character). A series of works introduce an approach to map international domain name characters into the current character set, avoiding changes to the actual DNS infrastructure [24], [25], [26]. This approach could also be used in our architecture, though its ramifications merit further study.

Finally, we note that multi-homing is as natural to incorporate in name-based routing as it is in the case of IP. Finally, the name-based approach can support host mobility because names are not tied to location by design.

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] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS Security Introduction and Requirements," RFC 4033 (Proposed Standard), 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4033.txt
[4] DMOZ, "Open directory project."
[5] U. o. O. Advanced Network Technology Center, "Route views project," http://www.routeviews.org/.
[6] M. Ruiz-Sanchez, E. Biersack, and W. Dabbous, "Survey and taxonomy of IP address lookup algorithms," *IEEE Network*, vol. 15, no. 2, pp. 8–23, 2001.
[7] D. Morrison, "Patricia - practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM*, vol. 15, no. 4, pp. 514–534, October 1968.
[8] K. Sklower, "A tree-based packet routing table for Berkley Unix," in *Winter Usenix*, January 1991.
[9] I. S. Consortium, "Internet domain survey," http://www.isc.org/index.pl?/ops/ds/.
[10] W. Fang and L. Peterson, "Inter-as traffic patterns and their implications," GLOBECOM, 1999.
[11] J. Rexford, J. Wang, Z. Xiao, and Y. Zhang, "Bgp routing stability of popular destinations," ACM SIGCOMM Workshop on Internet measurement, 2002.
[12] N. Taft, S. Bhattacharyya, J. Jetcheva, and C. Diot, "Understanding traffic dynamics at a backbone pop," SPIE, 2003.
[13] B. Claise, "Cisco systems netflow services export version 9," IETF RFC 3954, October 2004.
[14] B. Edelman, "Web sites sharing IP addresses: Prevalence and significance," September 2003, http://cyber.law.harvard.edu/people/edelman/ip-sharing/.
[15] D. Cheriton and M. Gritter, "TRIAD: A new next generation Internet architecture," Stanford Computer Science, Tech. Rep., March 2000.
[16] P. Francis and R. Gummadi, "IPNL: A NAT-extended Internet architecture." ACM SIGCOMM, 2002.
[17] S. Deering and R. Hinden, "Internet protocol, version 6 (IPv6) specification," IETF RFC 2460, December 1998.
[18] D. Clark, R. Braden, A. Falk, and V. Pingali, "FARA: Reorganizing the addressing architecture," in *ACM SIGCOMM Computer Communication Review*, vol. 33, no. 4, October 2003, pp. 313–321.
[19] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure." ACM SIGCOMM, 2006.
[20] I. Castineyra, N. Chiappa, and M. Steenstrup, "The Nimrod routing architecture," IETF RFC 1992, August 1996.
[21] R. Moskowitz and P. Nikander, "Host identity protocol (HIP) architecture." IETF RFC 4423.
[22] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A layered naming architecture for the Internet," *ACM SIGCOMM Computer Communication Review*, vol. 34, no. 4, pp. 343–352, October 2004.
[23] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker, "ROFL: Routing on flat labels." ACM SIGCOMM, 2006.
[24] P. Faltstrom, P. Hoffman, and A. Costello, "Internationalizing Domain Names in Applications (IDNA)," RFC 3490 (Proposed Standard), 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3490.txt
[25] P. Hoffman and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)," RFC 3491 (Proposed Standard), 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3491.txt
[26] A. Costello, "Punycode: A Bootstring encoding of Unicode for Internationalized Domain Names in Applications (IDNA)," RFC 3492 (Proposed Standard), 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3492.txt