Efficient Lookup Scheme for Non-aggregatable Name Prefixes and Its Evaluation

Masaki FUKUSHIMA (a), Atsushi TAGAMI, and Toru HASEGAWA††, Members

SUMMARY Content-Centric Networking (CCN) employs a hierarchical but location independent content naming scheme. While such a location independent naming brings various benefits including efficient content delivery, mobility, and multihoming, location independent name prefixes are hard to aggregate. This poses a serious scaling issue on the efficiency of looking up content names in a huge Forwarding Information Base (FIB) by longest prefix matching, which requires seeking the longest matching prefix through all candidate prefix lengths. We propose a new scheme for efficiently looking up non-aggregatable name prefixes in a large FIB. The proposed scheme is based on the observation that the bottleneck of FIB lookup is the random accesses to the high-latency off-chip DRAM for prefix seeking and this can be reduced by exploiting the information on the longest matching prefix length in the previous hop. Our evaluation results show that the proposed scheme significantly improves FIB lookup latency with a reasonable traffic parameters observed in today’s Internet.

key words: CCN, forwarding, FIB, longest prefix match

1. Introduction

Sustainability of the society has emerged as a pressing issue that needs to be addressed by information technologies. Recently, Information-Centric Networking (ICN) [1] is getting more and more attention in the future internet research community, as a means for improving the sustainability of the society [2], [3] as well as enabling scalable and cost-efficient content distribution, intrinsic mobility, and multihoming. Content-Centric Networking (CCN) [4] pursued in Named-Data Networking (NDN) project is a promising ICN architecture that employs a hierarchical content naming scheme.

In CCN, content names are location independent and there is no notion of locator like IP address. Instead of name resolution like DNS, the bindings between a content name and its content source locations are gradually resolved by routers in a hop-by-hop basis. Each CCN router has a Forwarding Information Base (FIB), which binds every content name prefix to the next hops (i.e., the outgoing faces) toward its content sources. When a router needs to forward an Interest packet for a content name, it looks up the name in its FIB by longest prefix match, retrieves the next hop information, and forwards the packet to the next hop routers. Such hop-by-hop content locating naturally supports efficient content distribution, mobility, and multihoming. A popular content can be hosted by many content sources without managing many locators like in current CDNs. Moreover, each content source can change/add its attachment points to the network without globally advertising the new locators of these points to its content consumers or routing packets via an indirect point like in Mobile IP.

On the other hand, such location independent names raise scalability issues on FIB [5], [6]. Since location independent names are assigned to content sources regardless of their topological locations, name prefixes are hard to aggregate and thus the FIBs of CCN routers will be far larger than those of current IP routers. Thus, it is crucial to efficiently store and lookup such large FIBs. Fortunately, the downside trend of DRAM cost will enable CCN routers to store large FIBs on memory. FIB lookup latency issue is, however, more challenging due to the large latency of DRAM access and the complexity of longest prefix matching on variable-length names. Software-based FIB mechanisms employ a hash table [5] or trie [7]. Regardless of the underlying data structures, they need to seek the longest matching prefix through all candidate prefix lengths (in descending order with a hash table, or ascending order with a trie), and thus the number of random accesses to the DRAM per lookup is proportional to the name length, which makes the FIB lookup latency proportional to the name length and FIB throughput inversely proportional to the name length. In order to eliminate this limitation, hardware-based FIB mechanisms are proposed [6], [8], which store a Bloom filter in a low-latency on-chip (SRAM) memory and populate it with prefixes. For each prefix length, the fast Bloom filter is checked first, and only if it gives a positive result, a slow hash table in an off-chip DRAM is probed to retrieve the next hop information. If given a sufficiently large on-chip memory, a name lookup involves only a single DRAM access regardless of the name length. However, this hardware-based FIB requires an expensive on-chip memory of the size proportional to the number of prefixes, which makes its Internet-scale deployment in backbone routers infeasible.

We propose a new scheme to improve the efficiency of FIB lookup, which can be applied to the software-based FIBs for faster lookup and to the hardware-based FIBs to reduce on-chip memory. The proposed scheme is motivated by the observation that Interest packets matching a non-aggregatable prefix are forwarded by the same prefix length at every hop. Therefore, by exploiting the information on the longest matching prefix length in the previous hop, each
CCN router could find the longest matching prefix without prefix seeking. Although, in the current CCN protocol, a router cannot learn the prefix length matched in the previous hop, it is easy to add some link-local header that carries the prefix length information along with each Interest packet. This does not violate the hop-by-hop principle of CCN, and can be incrementally deployed in the network.

In our previous workshop paper [9], we propose the new FIB lookup scheme and show a preliminary evaluation results based on a simple model and theoretical parameters. In this paper, we conduct more thorough evaluation based on empirical data from the current Internet. These new results suggest that the proposed scheme is a promising approach to improve the efficiency of the CCN-enabled future Internet.

2. Background

This section briefly reviews the design of CCN routers’ forwarding plane and its bottleneck, and describes typical scenarios where non-aggregatable prefixes pose serious scaling issues.

2.1 Packet Forwarding in CCN

In CCN [4], contents are identified by a variable length name (like foo/bar/file1name), which consists components (foo, bar, and filename) and “/” as delimiters between components. The length of a name is counted by the number of components in it. CCN defines two packet types. A content consumer sends an Interest packet carrying the name of the desired content, and the packet is forwarded to some content sources. Upon its reception, the content sources respond by sending a Data packet carrying the requested content, and the packet is forwarded back to the consumer. In order to forward these packets, each router has three tables. When a router receives an Interest packet for a content name, it first looks up the name in the Content Store (CS), which caches the Data packets it forwarded. If the CS has any matching Data, it is forwarded back to the previous hop. Otherwise, the router looks up the name in the Pending Interest Table (PIT), which stores the binding between pending interest names (names of the Interest it forwarded but the response Data packet is not yet received) and their incoming faces. If the PIT has any matching entry, the incoming face is added to the entry. Otherwise, the router looks up the name by longest prefix match in the Forwarding Information Base (FIB), which stores the bindings between name prefixes and one or more outgoing faces. If the FIB has any matching entry, the packet is forwarded according to the FIB entry, and a new PIT entry is created. When a Data packet is received, it is stored in the CS, and if the PIT has any matching entry, the Data packet is forwarded back to the incoming faces of the PIT entry.

As Perino et al. [6] pointed out, FIB lookup is the most critical bottleneck of CCN routers, especially in high-speed backbone routers. This is because it requires longest prefix matching, and to make matters worse, location independent names are generally hard to aggregate, and thus a router’s FIB in a large scale network is loaded by hundreds of millions of non-aggregatable prefixes. In contrast to FIB lookup, CS and PIT lookups are not so serious bottleneck, because they do not necessarily require longest prefix matching. Besides, since CS and PIT entries are created on demand and eventually expire, their numbers are limited by the volume of active traffic, while FIB entries are maintained even for all potential, not necessarily active content sources.

2.2 Problems of Non-aggregatable Prefixes

If CCN is deployed at Internet scale, we must face a challenge of designing CCN routers that meet forwarding latency requirements, while accommodating a large number of non-aggregatable prefixes in their FIBs.

One obvious origin of non-aggregatable prefixes is the Internet backbone routing table advertised in CCN-capable BGP [10], because many contents are published under provider-independent prefixes. According to a rough estimate, a full routing table of BGP includes 620 million prefixes, which is the number of web server hostnames as of December 2012 [11]. This is a far more challenging number compared to 0.43 million prefixes of the current BGP [12]. CCN-based backbone network operators need hardware-based high-speed backbone routers that can forward Interest packets at tens or hundreds of Mpps at a moderate cost.

Another potential origin of non-aggregatable prefixes is user-supplied contents. In CCN, users communicate with each other by bi-directional Interest exchange like Voiceover-CMN [13]. Such applications publish contents under user-specific prefixes like /AccessProvider.com/User. While the BGP routing table stores only the aggregated prefix /AccessProvider.com, user-specific prefixes cannot be aggregated within the access provider’s network, because the name User is flat and location independent. In order to forward Interests to the location where User is attached to the access provider, its internal routers need to store millions or tens of millions of non-aggregatable user-specific prefixes.

Similar problems could occur in the cloud infrastructure of Online Social Network (OSN) providers. For example, Interest packets for /OSN.com/User need to be forwarded to the server hosting User’s contents such as blog posts, tweets, movies, and photos. Currently, such contents are located in randomly chosen servers by some distributed key-value store mechanism like distributed hash table. Recently, however, inefficiency of random placement has been pointed out [14], [15], because interests in OSN have strong spatial locality. In future, OSN providers might need to locate contents of strongly related users in a specific server located close to those users. This requires internal routers of large-scale OSN providers to store hundreds of millions of non-aggregatable user-specific prefixes.
3. Related Work

Some FIB mechanisms for CCN routers have been proposed, each of which suffers from scaling issues due to memory access latency when they are populated with a large number of non-aggregatable prefixes.

Software-based FIBs for general purpose CPUs are classified by their underlying data structures. Hash table provides exact matching with $O(1)$ comparisons. Longest prefix matching is possible by seeking through all candidate prefix lengths in descending order. At the worst case, for a name of length $B$, this requires $O(B)$ comparisons. CCNx implements a Name Prefix Hash Table (NPHT) [5] that combines FIB and PIT functions into a single hash table. Each NPHT entry has a “parent pointer”. If an Interest’s filename component (excluding its sequence number component) hits the entry added by its preceding Interest for the same file, the parent pointer enables faster prefix seeking by reducing the number of name comparisons required for longest prefix matching. Trie is another well-known data structure supporting longest prefix matching, which seeks through the candidate prefix lengths in ascending order. Wang et al. [7] proposed a trie-based FIB for CCN with an efficient name component encoding scheme. (Although they claim that their scheme can support a few million lookups per second on a PC-based software implementation, their evaluation only considers the domain name parts of content names, and does not fully consider variable- and unbounded-length names.)

Regardless of the underlying data structures, these software-based FIBs need to seek through the candidate prefix lengths. Since CCN FIB is generally too large to fit into a small on-chip (SRAM) cache of a general purpose CPU, a random access to a high-latency off-chip (DRAM) memory is required for every prefix length. Consequently, the number of random accesses to the off-chip memory is proportional to the name length, which makes the FIB lookup latency proportional to the name length. Parallel processing does not address this issue, because the bottleneck is the access port of the off-chip memory, not the processor. Also, note that the parent pointer of NPHT does not address this issue, because following a parent pointer is also a random access to the off-chip memory.

In order to eliminate this limitation, hardware-based FIBs are proposed, which can be implemented as an ASIC or FPGA chip. Unlike IP forwarding engines, it is infeasible to implement a large CCN FIB by TCAM, due to its high cost and energy consumption. Therefore, these hardware-based FIBs exploit low-latency on-chip memory to minimize access to high-latency off-chip memory. Bloom filter based FIB was first proposed by Dharmapurikar et al. [16] for IP routers. Bloom filter [17] is a randomized data structure for storing a set of items. It is more space-efficient than a hash table because, firstly, it stores only keys and no related value, and secondly, it supports only an approximate set membership query function. Namely, it might return a false positive result when the queried item does not belong to the set. The false positive probability depends on the size of the Bloom filter. By exploiting this space efficiency, Dharmapurikar et al. [16] use a Bloom filter on an on-chip memory to minimize access to a hash table on an off-chip memory. The Bloom filter is populated with prefixes. For each prefix length, the fast Bloom filter is checked first, and only if it gives a positive result, the slow hash table is probed to retrieve the next hop information. If given a sufficiently large on-chip memory, each lookup involves only a single access to the off-chip memory regardless of the address length. Perino et al. [6] study the feasibility of the Bloom filter based FIB for CCN routers. They conclude that today’s technology is not ready to support an Internet-scale CCN deployment in backbone routers, mainly because a Bloom filter based FIB requires an expensive on-chip memory whose size is proportional to the number of prefixes. Otherwise, the false positive probability of the Bloom filter increases, and thus prefix seeking is required as in the software-based FIBs. Based on the conceptual design of Perino, et al. [6], Varvello, et al. [8] proceed one step further into a more concrete CCN router design, which distributes FIB to many line cards. The number of line cards required by this design is proportional to the number of prefixes. Since each line card needs a large on-chip memory, this design also suffers from the similar cost issue.

4. Exploiting Neighbors’ Prefix Length Information

We propose a new scheme to improve the efficiency of FIB lookup, which is based on the prefix length information from the previous hop router. It can mitigate the scaling issue that the existing FIB mechanisms suffer in the scenarios described in Sect. 2. The proposed scheme is sufficiently generic to be applied to the existing hardware/software FIB mechanisms described in Sect. 3.

The proposed scheme is inspired by two architectural invariants [18] observed in the CCN architecture. One is the latency of off-chip memory, which is the critical bottleneck of FIB lookup and its drastic improvement will not be expected in near future, because an enormous number of non-aggregatable prefixes prevents routers from effectively caching FIB entries in a low-latency on-chip memory. The other is the length of a non-aggregatable prefix. If a router forwards an Interest packet by a non-aggregatable prefix, the next hop router is likely to forward the packet by the same prefix length. The proposed scheme minimizes the impact of the off-chip memory latency by exploiting the length of non-aggregatable prefixes.

4.1 Conventional FIB Lookup

The conventional FIB lookup function, denoted by $\text{faces} \leftarrow \text{FIBLookup}(N)$, takes a name $N$ as an input, seeks the longest matching prefix length $L$ through the candidate prefix lengths, retrieves the set of outgoing faces (to be precise, IDs of faces), and returns the faces.
Although this interface seems a very natural design choice from a single router viewpoint, it could be inefficient from a collective viewpoint. Suppose routers on a path have an almost same set of non-aggregatable prefixes in their FIBs. As described in Sect. 2, this is rather the norm than the exception. Suppose the name $N$ of an Interest packet matches a prefix of length $L$ in the non-aggregatable prefixes, and the packet happens to be forwarded along the path. Then, every router on the path redundantly looks up the same name $N$ in the almost same set of prefixes, and seeks the same prefix length $L$. Obviously, this is a waste of resources, especially for large FIBs.

4.2 Proposed Fast FIB Lookup

If a router has a large FIB similar to its neighbor’s FIBs, it can reduce prefix seeking by exploiting the longest matching prefix length $L$ in the previous hop. The proposed fast FIB lookup function, denoted by $(\text{faces, } L) \leftarrow \text{FastFIBLookup}(N, L)$, has additional input $L$ and output $L'$, the longest matching prefix length of the previous hop router and the current router, respectively. The former $L$ is forwarded from the previous hop and the latter $L'$ will be forwarded to the next hop along with the Interest packet.

Figure 1 illustrates the algorithm of FastFIBLookup. The two shaded boxes access to the high-latency off-chip DRAM. One shaded box SlowFIBLookup is the same as the conventional $FIB_{lookup}$, except for having an additional output $L'$. This modification is trivial. In the following, we assume SlowFIBLookup is implemented by a Bloom filter in an on-chip memory and a hash table in an off-chip memory like Perino et al. [6], because it is the most promising approach to implement high-end CCN routers.

The other shaded box is $HashTable(N[L])$, where $N[L]$ denotes the length-$L$ prefix of the name $N$, namely, the first $L$ components of the name $N$ (e.g., if $N = /foo/bar/filename$, then $N[2] = /foo/bar$.) This box probes the FIB hash table in the off-chip memory only for the prefix length $L$. Note that we can use the result of this probing only if the found prefix of length $L$ is a leaf prefix, namely, the prefix has no child prefix in the (hypothetical) prefix tree. This is because, if the prefix has some child prefixes, there is a possibility that one of them or their further descendant gives a match longer than $L$.

FastFIBLookup has two design objectives. Most importantly, it should avoid calling SlowFIBLookup as much as possible. Secondly, it should avoid unnecessarily calling $HashTable(N[L])$ if we can predict the hash table probing result is useless. In the best case, this scheme requires only a single access to the off-chip memory. These objectives are achieved as follows.

First, FastFIBLookup quickly filters out $N[L]$ if it is obviously a non-leaf prefix. For example, if the previous hop router forwarded a packet by the default prefix ("/"), the prefix of the same length in the current router (namely, the default prefix) is probably a non-leaf. This filtering is done by checking whether the FIB has any leaf prefix of length $L$, by using a small Leaf Prefix Count Table (LPCT), which holds the number of leaf prefixes for every prefix length. If there is no such prefix, we fall back to SlowFIBLookup.

Next, BloomFilter($N[L]$) is called in order to avoid unnecessary hash table probing if we have no such prefix. If its result is negative, we fall back to SlowFIBLookup. If the proposed scheme is applied to a software-based FIB without an on-chip Bloom filter, this step is simply omitted.

If the preceding two steps are positive, $HashTable(N[L])$ is called. By carefully choosing hash table implementation [19], this step can be done with a single access to the off-chip memory. If a prefix entry is found, and its $num\_child$ field equals zero, we are sure that it is the longest matching prefix. Otherwise, we fall back to SlowFIBLookup.

4.3 Additionally Required Information

In the above description, it is assumed that some pieces of information ($L$, $num\_child$, and LPCT) are available, which are not required by the conventional FIBs. They can be efficiently obtained as follows.

Firstly, in order to obtain the longest matching prefix length $L$ in the previous hop, we need to modify the protocol between neighboring routers. Since the size of prefix length information is very small, the overhead of adding this information is negligible from the viewpoint of link bandwidth. Another concern than efficiency is the impact of modifying CCN protocol from an architectural perspective. We discuss this issue in Sect. 6.1.

Secondly, the number of child prefixes should be maintained in each prefix entry’s $num\_child$ field, even if the FIB is dynamically updated. This is easily done with a trie-based FIB entry that has explicit child pointers, and can be done with a hash table based FIB entry that has only a parent pointer, by the following procedures. When adding a new
leaf prefix to the FIB, its \textit{num\_child} is set to zero, and its parent’s \textit{num\_child} is incremented. If the new prefix has no direct parent (e.g., adding /foo/bar to a FIB including only the default prefix “/”), all of the intermediate prefixes (/foo, in the above case) are added as dummy placeholders with \textit{num\_child} = 1. When deleting an existing prefix from the FIB, its parent’s \textit{num\_child} is decremented. Any dummy prefix is deleted when its \textit{num\_child} becomes zero. Although these update procedures need multiple accesses to the off-chip memory, it is not the bottleneck because FIB update is less frequent than packet forwarding [5].

Lastly, the \textit{LPCT} can be easily maintained by incrementing/decrementing \textit{LPCT}(L) whenever any non-dummy prefix of length \(L\) changes its \textit{num\_child} field from one-to-zero/zero-to-one, respectively.

5. Evaluation

We evaluate the efficiency of the \textit{FastFIBLookup} scheme compared to that of the conventional \textit{FIBLookup} scheme.

5.1 Evaluation Model

The efficiency of FIB lookup is evaluated by its latency, the time required by a router to lookup a name in its FIB. The latency of the conventional FIB lookup is evaluated by the model proposed by Perino et al. [6].

The latency \(L_{fast}\) of \textit{FastFIBLookup} is determined by the latencies of \textit{SlowFIBLookup} and \textit{HashTable}, and how often these functions are called. Figure 2 shows five possible execution paths (path 1, \ldots, 5) in \textit{FastFIBLookup}. Let \(P_i\) (\(i = 1, \ldots, 5\)) denote the probability that a packet follows execution path \(i\). Then, the latency \(L_{fast}\) is defined by

\[
L_{fast} = L_{slow}(P_1 + P_2 + P_3 + P_4) + L_{hash}(P_3 + P_4 + P_5)
\]

where \(L_{slow}\) and \(L_{hash}\) are the latencies of \textit{SlowFIBLookup} and \textit{HashTable}, respectively, and derived as shown in Appendix A. The probabilities \(P_1, \ldots, P_5\) depend on the characteristics of the network topology and traffic forwarded by the router.

In order to determine reasonable values of these probabilities experienced by a typical backbone router, we consider a network model of an Autonomous System (AS) shown in Fig. 3. This AS has a \textit{backbone network} consisting of backbone routers, each of which is located in a Point-of-Presence (PoP). These backbone routers are connected by backbone links, and running both of BGP and IGP. Each backbone router has a link to an \textit{aggregation router}, which is in charge of forwarding packets between the backbone router and customers accommodated by the PoP. Each backbone router also has links to other ASes.

In this model, a packet from a customer or another AS enters the backbone network at a PoP, then forwarded along a sequence of backbone routers, called \textit{backbone path}, and finally exits to a customer or another AS at a PoP. (Note that backbone routers are neither of sources or sinks of traffic.) When forwarding a packet, the backbone routers use either of the following two types of prefixes. One is \textit{AS-level prefixes} globally advertised by BGP, which is used to forward packets destined for another AS. The other is \textit{PoP-level prefixes} advertised by IGP, which is used to forward packets destined for customers within the AS. Besides, the aggregation router in a PoP forwards packets destined for outside the PoP by the \textit{default prefix} (i.e., a zero-length prefix “/”) to its upstream backbone router.

In this evaluation, we assume PoP-level prefixes are aggregatable at AS-level. At first glance, this assumption contradicts with our motivation that name prefixes are non-aggregatable. Actually, this assumption is reasonable by the following reasons. In general, prefix aggregation is used to reduce the number of prefixes. On the other hand, in our evaluation, the number of prefixes is given as a parameter, not the result of aggregation. If the number of prefixes is given, the assumption that prefix length is variable (i.e., prefixes are aggregatable) results in more conservative evaluation (in a sense, the worst-case evaluation), because the proposed scheme can reduce FIB lookup latency only if the
Each packet forwarded by a backbone router is classified into eight types, depending on the router’s position on the backbone path and the packet’s source AS/destination AS.

<table>
<thead>
<tr>
<th>Packet type ( j )</th>
<th>The current router’s position on the backbone path</th>
<th>Source AS</th>
<th>Destination AS</th>
<th>Prefix used by the previous hop</th>
<th>Prefix used by the current router</th>
<th>Same prefix?</th>
<th>Path in Fig. 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>first</td>
<td>same AS</td>
<td>same AS</td>
<td>default</td>
<td>PoP-level</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>first</td>
<td>same AS</td>
<td>another AS</td>
<td>default</td>
<td>AS-level</td>
<td>no</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>first</td>
<td>another AS</td>
<td>same AS</td>
<td>AS-level</td>
<td>PoP-level</td>
<td>no</td>
<td>2 or 3</td>
</tr>
<tr>
<td>4</td>
<td>first</td>
<td>another AS</td>
<td>another AS</td>
<td>AS-level</td>
<td>AS-level</td>
<td>yes</td>
<td>4 or 5</td>
</tr>
<tr>
<td>5</td>
<td>not first</td>
<td>another AS</td>
<td>same AS</td>
<td>PoP-level</td>
<td>PoP-level</td>
<td>yes</td>
<td>4 or 5</td>
</tr>
<tr>
<td>6</td>
<td>not first</td>
<td>same AS</td>
<td>another AS</td>
<td>AS-level</td>
<td>AS-level</td>
<td>yes</td>
<td>4 or 5</td>
</tr>
<tr>
<td>7</td>
<td>not first</td>
<td>another AS</td>
<td>same AS</td>
<td>PoP-level</td>
<td>PoP-level</td>
<td>yes</td>
<td>4 or 5</td>
</tr>
<tr>
<td>8</td>
<td>not first</td>
<td>another AS</td>
<td>another AS</td>
<td>AS-level</td>
<td>AS-level</td>
<td>yes</td>
<td>4 or 5</td>
</tr>
</tbody>
</table>

From these observations, each packet forwarded by a backbone router (hereafter called the current router) can be classified into eight types shown in Table 1. As shown in the leftmost four columns of the table, each packet type is defined by three attributes: the current router’s position on the backbone path followed by the packet, the source AS of the packet, and the destination AS of the packet. If the current router is the first router in the backbone path, its previous hop is an aggregation router or a router in a neighboring AS. Otherwise, its previous hop is a neighboring backbone router.

Each packet type uniquely determines the prefix types at the previous hop and the current router. Packets of type 1 and 2 are traffic from customers accommodated by the current router’s PoP, and forwarded from the aggregation router by the default prefix, and thus follow the execution path 1 in Fig. 2. Namely,

\[
P_1 = T_1 + T_2
\]

where \( T_j \) denote the probability that a packet is type \( j \). Packets of type 3 are traffic from a neighboring AS and destined for customers within the AS. These packets are forwarded by AS-level prefixes in the previous hop, and forwarded by PoP-level prefixes in the current router. Thus, they follow the execution path 2 or 3, depending on whether the Bloom filter results in false positive or not, namely

\[
P_2 = T_3(1 - P_{fpos}),
\]

\[
P_3 = T_3P_{fpos},
\]

where \( P_{fpos} \) is the false positive probability shown in Appendix A. Packets of type 4, 5, 6, 7, and 8 are forwarded by the same prefix in their previous hop router and current router, and thus follow the execution path 4 or 5, depending on whether the prefix is a leaf or not, namely

\[
P_4 = (T_4 + T_5 + T_6 + T_7 + T_8)(1 - P_{leaf}),
\]

\[
P_5 = (T_4 + T_5 + T_6 + T_7 + T_8)P_{leaf},
\]

where \( P_{leaf} \) is the probability that a prefix is a leaf.

The remaining parameters, \( T_j \) and \( P_{leaf} \), are determined from empirical data. Unfortunately, there is currently no large-scale commercial CCN network. As a second-best way, we investigated these values of the current Internet.

The packet type distribution \( T_j \) (\( j = 1, \ldots, 8 \)) is estimated from empirical data. For each backbone topology of six ASes published by Mahajan et al. [20], we synthesize a traffic matrix by using the gravity model proposed by Roughan [21], and route this traffic by shortest path routing. See Appendix B for the details on how \( T_j \) is derived. The results are summarized in Table 2. We use the average over the six ASes as the values of \( T_j \).

The probability \( P_{leaf} \) is determined from the current IPv4 prefixes advertised in BGP, available from CAIDA [22]. The dataset includes 460441 IPv4 prefixes. We randomly choose an address from the address space covered by the dataset, lookup the longest matching prefix, and investigate whether the prefix has any child prefix (i.e., any prefix covered by the prefix) or not. As a result of sampling one million addresses, we obtain \( P_{leaf} = 0.685 \). For reference, we also evaluate the latency for \( P_{leaf} = 0.5, 0.75, \) and 1.

### 5.2 Evaluation Results

The proposed scheme is evaluated in two scenarios; the application to a hardware-based FIB like Perino et al. [6], and the application to a software-based FIB implemented by a hash table. Unless otherwise stated, the parameters not discussed in Sect. 5.1 are set as shown in Table 3.

Figure 4 shows the latencies of hardware-based FIB as functions of the number of prefixes \( n \) for different values of \( P_{leaf} \). The entire on-chip memory is dedicated to the Bloom filter and its size is fixed to 1 Gbits. For small FIBs (\( n < 100 \) million), both of the conventional and proposed scheme achieve nearly minimum latency of 15 ns, the latency of a single access to off-chip Reduced Latency DRAM (RLDRAM). Namely, each lookup requires only a single hash table probing. As \( n \) increases, the latency of the conventional scheme rapidly increases and converges to \( B \) times larger than the minimum, where \( B \) is the name length, be-
Table 3 Other evaluation parameters.

<table>
<thead>
<tr>
<th>Parameters</th>
<th>Hardware-based FIB</th>
<th>Software-based FIB</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of prefixes ((n))</td>
<td>620 million [11]</td>
<td>-</td>
</tr>
<tr>
<td>On-chip memory size ((M))</td>
<td>1 Gbits</td>
<td>-</td>
</tr>
<tr>
<td>On-chip memory latency ((L_{on}))</td>
<td>0.45 ns [6]</td>
<td>-</td>
</tr>
<tr>
<td>Off-chip memory latency ((L_{off}))</td>
<td>15 ns [6]</td>
<td>60 ns [23]</td>
</tr>
<tr>
<td>Name length ((B))</td>
<td>30 comp. [6]</td>
<td>1, ... , 30 comp.</td>
</tr>
<tr>
<td>Prefix length ((L))</td>
<td>-</td>
<td>1, ... , 15 comp.</td>
</tr>
<tr>
<td>Name component length ((K))</td>
<td>-</td>
<td>10 bytes [24], [25]</td>
</tr>
<tr>
<td>Hash table load factor ((\alpha))</td>
<td>-</td>
<td>0.5</td>
</tr>
<tr>
<td>CPU clock frequency ((F))</td>
<td>-</td>
<td>3 GHz</td>
</tr>
<tr>
<td>CPU word length ((W))</td>
<td>-</td>
<td>8 bytes</td>
</tr>
</tbody>
</table>

Fig. 4 The latencies of hardware-based FIBs. The on-chip memory size \(M\) is fixed to 1 Gbits, which is a fairly high standard for today’s technology.

Fig. 5 The latencies of hardware-based FIBs. The number of prefixes \(n\) is fixed to 620 million, which is the same as the number of web server hostnames of today.

decause the false positive probability of the Bloom filter converges to 1 due to the excessive amount of registered prefixes. The latencies of the proposed scheme increase slower. For \(n = 620\) million (the number of web server hostnames of today [11]) and \(P_{leaf} = 0.685\), the latency of the proposed scheme is less than half of the conventional scheme.

Figure 5 shows the latencies as functions of the on-chip memory size \(M\) (i.e., Bloom filter size). For any latency target, the proposed scheme significantly reduces the required on-chip memory size, which is the primary source of the cost increase to support CCN in a backbone router [6].

Although it is currently unrealistic to use a software-based router for forwarding backbone traffic, it would be worthwhile to evaluate the performance of the proposed scheme applied to software-based FIBs, due to the following reasons. Firstly, software-based router technologies are rapidly improving in recent years [27], [28]. Secondly, CCN is rather long term research direction and thus the applicability of software-based router in the era of CCN deployment could be broader than that of today. Therefore, based on these assumptions, we evaluate the performance of the proposed scheme applied to software-based FIBs. As a reference, we suppose software-based backbone router’s FIB lookup latency must be kept below 230 ns, which is the average interval of packets of 4-KB size (the default packet size of the current CCN implementation [29]) arriving at the rate of 140 Gbps (the line card throughput of state-of-the-art backbone routers [30]).

The performance of the software-based FIB is evaluated, where the FIB is too large to fit into the on-chip cache of a CPU. Figure 6 shows the latencies as functions of the name length \(B\). The minimum latency is 60 ns, the latency of a single access to off-chip DRAM. The latency of the conventional scheme increases linearly to the name length, because every prefix length requires an off-chip DRAM access. The latencies of the proposed scheme increase slower. Even if \(B = 30\), the 99.9-percentile of URL length in a traffic trace [6], the latency of the proposed scheme with \(P_{leaf} = 0.685\) is less than half of the conventional scheme. In order to keep the lookup latency below 230 ns, software-based FIBs can not support names with arbitrary length. The conventional scheme can support only name with 5 components, whereas the proposed scheme with \(P_{leaf} = 0.685\) can increase this limitation to 10 components.
6. Discussion

6.1 Impact on CCN Architecture

In the standard CCN architecture, there is no protocol to exchange prefix length information with neighboring routers. In order to support this feature, we have two options. One is to modify the networking layer protocol of CCN and add a prefix length field in the Interest packet format. Although this is easy from the viewpoint of implementation cost, it might violate the principle of CCN architecture, because prefix length information is per-hop information and inappropriate to be embedded in the networking layer. Probably, adding some link layer protocol is better solution, which installs a tiny shim header carrying the prefix length information in front of every Interest packet. Such a link layer protocol can be easily designed by extending the existing link layer protocol for NDN [31]. Note that such a link layer modification affects neither of the CCN network layer protocol nor its hop-by-hop content locating principle, and thus it is easy to incrementally deploy the proposed scheme.

6.2 Security Consideration

We need to discuss whether exchanging prefix length information between neighbors involves any security issue or not, because such information is not exchanged in conventional networking protocols. First of all, since the proposed scheme is a link local protocol, there is no security issue between non-neighboring routers.

As for integrity concerns, there could be a possibility that a router sends an invalid prefix length other than \( L' \) to its next hops, intentionally or accidentally. From a receiver’s viewpoint, this is rather an efficiency issue and not a security issue, because the proposed scheme strictly checks whether the prefix of that length is a leaf prefix or not. If a name matches with a leaf prefix, it must be the longest matching prefix. Otherwise, the proposed scheme falls back to the conventional scheme. Therefore, no router needs to trust its neighbor to correctly send the longest matching prefix length.

As for confidentiality concerns, some network operators might think prefix length is sensitive information. For example, suppose an access provider has a confidentiality business contract with a content provider (e.g., /Content-Provider.com). This contract requires the access provider to perform some policy-based routing for a specific prefix (e.g., /ContentProvider.com/Movie/PayPerView). This is implemented by adding the specific prefix to routers’ FIBs. If the proposed scheme sends the length of the prefix to another network operator who provides transit between the access provider and the content provider, the transit provider could guess the existence of the confidential business contract. In order to prevent such undesired information disclosure, the access provider can add a “length to export” attribute to the prefix, which is used instead of the actual longest matching prefix length for the packets forwarded to the transit provider’s routers. Such a workaround is required for a small fraction of prefixes in the BGP table, and thus it does not affect the overall efficiency of the proposed scheme.

As for availability concerns, it is technically possible for a owner of a router to mount a DoS attack on a neighboring router by intentionally sending invalid prefix length information. From a practical perspective, however, such an attack is not effective in commercial ISP networks, because the attacker must be a neighbor of the victim ISP, and thus the victim ISP can easily identify the attacker.

7. Conclusion

We proposed a new scheme for efficiently looking up non-aggregatable name prefixes in a large FIB. The proposed scheme is based on the observation that the bottleneck of FIB lookup is the random accesses to the high-latency off-chip DRAM for prefix seeking and this can be reduced by exploiting the information on the longest matching prefix length in the previous hop. Our evaluation results show that the proposed scheme significantly improves FIB lookup latency with a reasonable traffic parameters observed in today’s Internet.

Acknowledgments

The research leading to these results has partially received funding from the EU-JAPAN initiative by the EC Seventh Framework Programme (FP7/2007-2013) Grant Agreement No. 608518 and NICT under Contract No. 167 (the GreenICN project).

References

[12] “Bgp routing table analysis reports;” bgp.potaroo.net/
[31] “Routeviews prefix to as mappings dataset (pfx2as);” bgp.potaroo.net/
pfx2as.xml

Appendix A: Latency Model

For a hardware-based FIB, the latency $L_{slow}$ of $SlowFIBLookup$ follows the equation (1) of Perino et al. [6] and the latency $L_{hash}$ of $HashTable$ equals the off-chip memory latency $L_{off}$. The false positive probability $P_{ppos}$ of an $M$-bit Bloom filter populated by $n$ prefixes is given by

$$P_{ppos} = \left(1 - \frac{1}{2^M}\right)^n.$$  
(A-1)

In contrast to the hardware-based FIB, a software-based FIB probes each length one by one in descending order, from the name length $B$ until finding the longest matching prefix length $L$. Therefore, the latency of $SlowFIBLookup$ in a software-based FIB is given by

$$L_{slow} = \sum_{\ell=B}^{L} L_{hash}(\ell).$$  
(A-2)

The effect of the on-chip cache memory is omitted, because the FIB is too large to fit into the small cache of a CPU. Thus, we have no on-chip Bloom filter, namely, $P_{ppos} = 1$. Also, by carefully implementing $HashTable$ with open addressing [19], its latency $L_{hash}$ for a prefix of length $\ell$ is given by

$$L_{hash}(\ell) = L_{off} + \frac{1}{1 - \alpha} \cdot \frac{\ell K}{WF},$$  
(A-3)

the sum of the latency of an off-chip memory access and the time required to compare hash keys, where $1/\alpha$ is the number of key comparisons required to probe the hash table of load factor $\alpha$ [26] and $\frac{\ell K}{WF}$ is the time required to compare equality of two prefixes of the average component length $K$ bytes, at the rate of one word length $W$ per CPU clock of frequency $F$.

Appendix B: Packet Type Distribution

Given one of the backbone topologies published by Mahajan et al. [20], [32], the probability $T_j$ that a packet forwarded by a backbone router is type $j$ is derived as the weighted average over all backbone routers in the topology, namely,

$$T_j = \sum_r T_{jr} \frac{V_j}{\sum_u V_u},$$  
(A-4)

where $T_{jr}$ is the probability that a packet forwarded by a specific backbone router $r$ is type $j$, and $V_j$ is the relative frequency of a packet visiting the router $r$. 

Washington.edu/research/networking/rocketfuel/
Let $C_{ix}^{in}$ and $C_{ix}^{out}$ be the probability that a packet respectively enters from and exits to a customer accommodated by the PoP of a backbone router $x$. Similarly, let $A_{ix}^{in}$ and $A_{ix}^{out}$ be the probability that a packet respectively enters from and exits to other ASes linked with a backbone router $x$. We follow the gravity model [21], namely, the entering location and exiting location of a packet is assumed to be statistically independent. Then,

$$V_x = \sum_x \sum_y (C_{ix}^{in} + A_{ix}^{in})(C_{yx}^{out} + A_{yx}^{out})R_{xry}$$  \hspace{1cm} (A-5)$$

where $R_{xry}$ is the probability that a packet entering at a backbone router $x$ and exiting at a backbone router $y$ visits a backbone router $r$. By assuming equal-cost multipath routing, this equals to the number of shortest path(s) from $x$ via $r$ to $y$ divided by the total number of shortest path(s) from $x$ to $y$.

For packet type $j = 1, 2, 3, \text{and} 4$ (i.e., the router is the first hop in the backbone path), $T_j$ are given by

$$T_{1,r} = \sum_y C_{ix}^{in} C_{yx}^{out},$$  \hspace{1cm} (A-6)$$

$$T_{2,r} = \sum_y C_{ix}^{in} A_{yx}^{out},$$  \hspace{1cm} (A-7)$$

$$T_{3,r} = \sum_y A_{ix}^{in} C_{yx}^{out},$$  \hspace{1cm} (A-8)$$

$$T_{4,r} = \sum_y A_{ix}^{in} A_{yx}^{out}. \hspace{1cm} (A-9)$$

For type $j = 5, 6, 7, \text{and} 8$ (i.e., the router is not the first hop in the backbone path), $T_j$ are given by

$$T_{5,r} = \sum_{x\neq r} \sum_y C_{ix}^{in} C_{yx}^{out} R_{xry},$$  \hspace{1cm} (A-10)$$

$$T_{6,r} = \sum_{x\neq r} \sum_y C_{ix}^{in} A_{yx}^{out} R_{xry},$$  \hspace{1cm} (A-11)$$

$$T_{7,r} = \sum_{x\neq r} \sum_y A_{ix}^{in} C_{yx}^{out} R_{xry},$$  \hspace{1cm} (A-12)$$

$$T_{8,r} = \sum_{x\neq r} \sum_y A_{ix}^{in} A_{yx}^{out} R_{xry}. \hspace{1cm} (A-13)$$

We assume that the entering/exiting probabilities, $A_{ix}^{in}, A_{ix}^{out}, C_{ix}^{in}, \text{and} C_{ix}^{out}$, are proportional to the normalized population $D_x$ of the city where the backbone router $x$ is located, namely,

$$C_{ix}^{in} = C_{ix}^{in} D_x, \quad C_{ix}^{out} = C_{ix}^{out} D_x, \quad A_{ix}^{in} = A_{ix}^{in} D_x, \quad A_{ix}^{out} = A_{ix}^{out} D_x. \hspace{1cm} (A-14)$$

For each backbone router $x$, its city name is annotated in the topology data [32]. Thus, we determine the value of relative population $D_x$ from the city population statistics by United Nations [33]. All values of $D_x$ are normalized so that $\sum_x D_x = 1$. If multiple backbone routers are located in a city, the population is equally distributed among those routers.

The total probabilities, $C_{ix}^{in}, C_{ix}^{out}, A_{ix}^{in}, \text{and} A_{ix}^{out}$, are derived by solving the following equations from (A-15) to (A-17). The entering probabilities and exiting probabilities should add up to unity, namely,

$$C_{ix}^{in} + A_{ix}^{in} = C_{ix}^{out} + A_{ix}^{out} = 1. \hspace{1cm} (A-15)$$

The ratio of upload and download by customers within the AS is given by a parameter $\alpha$, i.e.,

$$\frac{C_{ix}^{in}}{C_{ix}^{out}} = \alpha. \hspace{1cm} (A-16)$$

We set $\alpha = 2.63$ from the traffic statistics of broadband users in Japan [34]. The probability that a packet entering from other ASes is destined for a customer within the AS is given by a parameter $\beta$, namely,

$$\frac{C_{ix}^{out}}{C_{ix}^{in} + C_{ix}^{out}} = \beta. \hspace{1cm} (A-17)$$

We set $\beta = 0.264$, which is the inverse of the average AS hop length 3.7856 [12] as of Nov. 2012. These equations yield a solution

$$C_{ix}^{in} = \alpha \beta, \quad C_{ix}^{out} = \beta, \quad A_{ix}^{in} = 1 - \alpha \beta, \quad A_{ix}^{out} = 1 - \beta. \hspace{1cm} (A-18)$$

Masaki Fukushima received the B.E. and M.E. degrees in information and computer science from Waseda University, Tokyo, Japan, in 1998 and 2000, respectively. He joined KDDI in 2000, where he has been engaged in research and development on the performance evaluation and traffic control techniques for communication networks. His current research interests include network virtualization and information-centric networking. He is currently a research engineer at Future Communication Architecture Laboratory in KDDI R&D Laboratories, Inc. He received the Young Researchers’ Award of the IEICE in 2006.

Atsushi Tagami received the M.E. degrees in Computer Science from Kyushu University, Japan in 1997. He joined the KDDI R&D Laboratories Inc. in the same year, where he has been engaged in research and development on Performance Measurement of Communication Networks and Overlay Networking. And he received the Ph.D. degree in Informatics from the same university in 2010. He is currently a senior manager at Future Communication Architecture Laboratory of KDDI R&D Laboratories, Inc. He received the Best Paper Award of IEICE Communications Society in 2010.
Toru Hasegawa received the B.E., the M.E. and Dr. Informatics degrees in information engineering from Kyoto University, Japan, in 1982, 1984 and 2000, respectively. Since joining KDD (currently KDDI) in 1984, he has been working in the field of formal description technique (FDT) of communication protocols, ATM and high-speed protocols. From 1990 to 1991, he was a visiting researcher at Columbia University. His current interests are future Internet, Information Centric Networking, mobile computing and so on. He is currently a professor at Osaka University. He received the Meritorious Award on Radio of ARIB in 2003. He is a fellow of IEICE and IPSJ.