An Efficient Prefix Caching Scheme for Fast Forwarding in Named Data Networking

Jinsoo KIM[1], Junghwan KIM[1]*

[1] Division of ICT Convergence Engineering, Konkuk University,
268 Chungwon-daero, Chungju-si, Chungcheongbuk-do, 27478, Korea
jinsoo@kku.ac.kr, jhkim@kku.ac.kr (**Corresponding author*)

Abstract: Named data networking (NDN) has recently received much attention as a promising structure for information-centric networking. The name lookup in NDN is a fairly complicated process because of the unbounded name space and its variable string matching characteristics. This paper proposes a prefix caching scheme for fast name lookup in the forwarding information base (FIB) of NDN. It is based on the critical distance that is the longest distance between a prefix and its child prefixes. Caching a non-leaf prefix can lead to incorrect lookup results. In the proposed scheme the non-leaf prefix is expanded for caching based on the critical distance to avoid such incorrect results. This scheme can be applied to any kind of underlying FIB to employ the spatial locality. We also simulate and evaluate our scheme in comparison with other schemes. Experiment results show that our scheme has a higher cache hit ratio than other schemes.

Keywords: Named data networking, Prefix caching, Fast forwarding, Critical distance, Prefix expansion.

1. Introduction

The Internet has evolved over the past several decades as an essential infrastructure for the social, economic and cultural aspects of everyday life. It has employed the host-centric paradigm in which packets are communicated according to host addresses. However, this paradigm of the Internet becomes an obstacle in accepting the rapidly growing information-oriented services such as Internet of Things [4]. Information-centric networking (ICN) is a promising future Internet provided to overcome the limitations of current host-centric Internet, and one of the representative ICNs is the *named data networking* (NDN) [12, 13, 23].

The NDN router looks up a prefix in its *forwarding information base* (FIB) that matches the name in the packet to forward the incoming packet. The delivery of packets in an NDN is name-based, unlike an IP network based on an IP address. NDN names are human-readable and hierarchical in structure to provide intrinsically efficient forwarding [2].

There can be multiple matching prefixes in FIB of NDN, and the longest matching one is chosen as the final result. This process of name lookup is called the *longest prefix matching* (*LPM*) similarly to IP address lookup. To provide stable service in an NDN, the router should reach the same rate of packet delivery as the link speed. Unfortunately, the LPM is a very complicated and time-consuming task because of variable string matching. Moreover, the length of the name is not fixed and the name space is unbounded. So the name lookup is very challengeable, and many researches have focused on reducing the lookup time.

Caching is one of the common techniques used to improve the performance in various information processing applications. It is usually accomplished by exploiting temporal locality or spatial locality. The name lookup time can be reduced in FIB by using the caching technique [3]. The temporal locality can be exploited in the name lookup when the identical name is repeatedly referenced within a certain period of time. Likewise, the spatial locality can be exploited if the names with the same prefix as the LPM results are referenced repeatedly. Since the NDN provides in-network caching, it is probably not significant to utilize the temporal locality in the FIB. On the other hand, prefix caching can improve the forwarding performance by using spatial locality.

The prefixes in FIB can be classified as either leaf or non-leaf. While the non-leaf prefix has some child prefixes, the leaf prefix does not. If a name can be matched to both a non-leaf prefix and its child prefix, caching the non-leaf prefix can cause an incorrect lookup result. In this paper, we propose a prefix caching scheme to solve this problem with a small overhead so that the router can perform an efficient and rapid name lookup for the FIB.

The rest of this paper is organized as follows. Section 2 outlines packet forwarding in NDN. In section 3 we introduce the prefix caching and study the related works. Section 4 explains the proposed prefix caching based on the information

https://doi.org/10.24846/v27i2y201805

about the distance between a prefix and its farthest child prefix. The performance of our scheme is evaluated using simulation in terms of cache hit ratio, in section 5. Finally, we conclude the paper in section 6.

2. Packet forwarding in NDN

There are two types of hosts in the NDN: consumers and producers. While the consumer is sending the *Interest* packet to get some content, the producer responds with the *Data* packet for the Interest. A packet contains the name to identify some content. A name is represented by a series of components separated by slashes [2]. For example, a name '/com/google/www/video' comprises four components which are '/com', '/google', '/www', and '/video'. The NDN router forwards the incoming packet based on its name.

The NDN router forwards packets using three data structures: content store (CS), pending interest table (PIT), and forwarding information base (FIB). The CS is used to cache Data packets for later use. The PIT stores the input faces of the second and subsequent Interest packets with the same name. The FIB is a repository containing name prefixes. Figure 1 shows the algorithms used to forward Interest and Data packets.

Algorithm Forward_Interest(*name*)
name: the name in *Interest* packet
1. **if** (*name* is in **CS**)
2. reply corresponding *Data* packet.
3. exit.
4. **if** (*name* is in **PIT**)
5. pend the incoming face to **PIT**.
6. exit
7. find *LPM prefix* to *name* in **FIB**.
8 forward the packet to outgoing face.
9. initiate a new **PIT** entry by *name*
10. exit.

Algorithm Forward_Data(*name*)
name: the name in *Data* packet
1. forward *Data* packet to each incoming face in **PIT**.
2. cache *Data* packet into **CS**.
3. exit

Figure 1. Packet forwarding algorithm

Several name lookup schemes have been proposed to forward packets at the link speed. These are usually based on Ternary Content-Addressable Memory (TCAM), tries, hash structure, or a combination of these.

A TCAM-based name lookup has been proposed in [15]. TCAM is an associative memory that searches all items simultaneously. Each cell in TCAM can store a * (do not care) value in addition to 0 or 1, so it is easy to store the prefixes and find the LPM prefix. However, TCAM is restrictively used in the name lookup by contrast to the IP address lookup, because it is difficult to accommodate an unlimited-length prefix and a large number of prefixes.

Trie is a tree-like structure, and its degree for name lookup is irregular and large. The name component encoding (NCE) scheme [19] has been proposed to resolve the irregularity of the component length by encoding one component as a unique integer. Several trie-based name lookup techniques have been proposed to reduce the actual number of memory accesses per lookup using parallelization [18], path compression [7], and pruning sub-tries [9].

Several hash-based name lookup techniques have been presented to compare an Interest name with candidate prefixes in the FIB by hashing a series of components. Most of these schemes use Boom filters together for efficiency, such as on-chip Bloom filter [8], mapping Bloom filter [10], two dimensional filter [14], and NameFilter [20]. An efficient hash table was designed by using not the real name components but their fingerprints [17], and it has been extended to collision free fingerprint-based hash table [16]. Some other hash-based schemes improved the lookup performance by reducing the search space [21, 22].

3. Prefix caching and related works

3.1 Prefix caching and prefix expansion

The forwarding engine in the FIB can improve the speed of name lookup if it stores the latest lookup results by using the cache in front of the original forwarding table. The prefix caching utilizes a kind of spatial locality. For incoming packets with names that result in the same LPM prefix, it can increase the cache hit ratio.

Figure 2 shows a forwarding table with four prefixes from *p* to *s* in the form of a tree-like trie structure. Assume that the four Interest packets, whose names are *name1, name2, name3*, and

name4 in Figure 2, are input in order. Since the name in the first packet is *name1* = /house/h1, the LMP prefix is *p* = /house. The prefix caching scheme caches the prefix *p* which is the first lookup result. The LMP prefix for *name2* = /house/h2 is *p* as well. Thus, the name lookup result of the second packet can be used to get fast results using *p* in the cache.

Name prefix	Output face
p = /house	1
q = /auto	2
r = /food/fruit	3
s = /auto/body/window	4

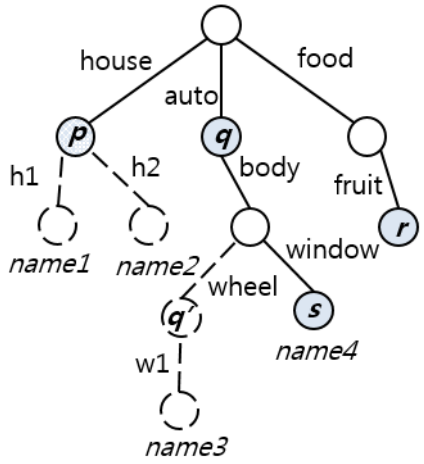

Figure 2. An example of prefix caching

Figure 2 also illustrates that prefix caching can yield incorrect lookup results. Since the third Interest packet contains the name *name3* = /auto/body/wheel/w1, the prefix *q* = /auto could be cached as the matched prefix. Note that the prefix *q* in the cache could be matched for *name4* = /auto/body/window as well. Thus, when name lookup is performed for the fourth packet, the prefix *q* is considered as the LMP owing to the cache hit. However, the actual LMP in the forwarding table is *s* = /auto/body/window, so *q* is the incorrect result. In Figure 2, prefixes *p, r*, and *s* are leaves, whereas prefix *q* is a non-leaf. The incorrect result of the prefix caching is caused by caching non-leaf prefixes such as prefix *q* in this figure. To avoid such wrong results, either only leaf prefixes should be cached, or non-leaf prefixes together with all their descendants should be cached. To cache only leaf prefixes is fairly restrictive in cache performance, and also it is not easy to manage all descendants of non-leaf prefixes in the cache**.**

Prefix expansion is one of the easiest ways to resolve the incorrect results of prefix caching. If we cache the proxy prefix *q′* = /auto/body/wheel instead of the prefix *q* = /auto for the third name lookup, the correct result can be yielded for *name4*. Since *name4* = /auto/body/window does not match prefix *q′*, a cache miss occurs. The process of obtaining prefix *q*’ from prefix *q* is called *prefix expansion*. The expanded prefix is chosen between the name and its LPM prefix. The closer the expanded prefix is to the LPM prefix, the more efficient it can be.

3.2 Previous prefix caching schemes

Prefix caching in the name lookup can exploit spatial locality by caching not the individual name but the prefix to be shared by some names. Several prefix caching techniques have already been proposed to speed up the IP address lookup, and more recently there have been some studies on applying the above to the name lookup of the FIB. Most of these methods cache the expanded prefix to solve the problem of caching the non-leaf prefix.

Liu [11] has presented three prefix extension methods, which are no prefix expansion (NPE), complete prefix tree expansion (CPTE), and partial prefix tree expansion (PPTE), to solve the non-leaf prefix caching problems in prefix caching for IP lookup. While NPE doesn’t cache the non-leaf prefix, CPTE and PPTE cache the completely and partially expanded prefixes. CPTE and PPTE increase the size of the forwarding table. Furthermore, it is not feasible to apply such prefix extensions to name lookups where the degrees are not uniform.

Kasnavi *et al*. [5] have developed a multi-zone pipelined cache (MPC) to use the prefix caching and IP address caching for short and long prefixes, respectively. The reverse routing cache with minimal expansion (RRC-ME) [1] has been proposed to expand non-leaf prefixes on the fly without modifying the original forwarding table. However, it is only applicable to the trie-based IP lookup engine. Apart from that, a new type of prefix caching called a bitmap-based prefix cache (BMCache) [6] has been proposed without any additional prefix entries in the forwarding table.

Chen *et al*. [3] have recently proposed a technique for caching an expanded prefix on the fly in the name lookup of the FIB, in a similar way to the

RRC-ME of the IP address lookup. This scheme is also only applicable to the trie-based name lookup engine. Each component that could constitute a name corresponds to a node in the trie for FIB. Thus, unlike IP address lookup, the original trie for name lookup is inherently irregular in its degree and its implementation may be infeasible.

4. Proposed prefix caching

4.1 Critical distance-based prefix expansion

Prefix caching incurs incorrect cache hits if there is no appropriate restriction. To avoid such incorrect results we adopted the *critical distance*-based prefix expansion.

Definition: The critical distance of a prefix is the highest value among distances to all its child prefixes.

For each prefix, critical distance is calculated and stored in FIB. Figure 3 shows an example of critical distance for a given set of prefixes. For example, the critical distance of prefix *p* is 2 since the farthest child is *s* among its children (*q*, *r*, and *s*) and the distance to *s* is 2. If a prefix has no child, i.e., it is a leaf prefix, then its critical distance should be zero.

Name prefix	Critical Distance
p = /com	2
q = /com/house	0
r = /com/auto	2
s = /com/food/fruit	0
t = /com/auto/body/window	0

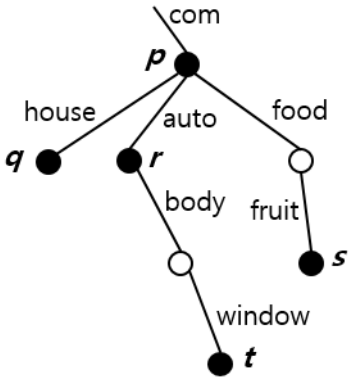

Figure 3. An example of critical distance

When FIB is looked up for a given input name, the critical distance of the matched prefix is exploited for prefix expansion. Suppose that a name *n*1 = /com/food/meat/beef is given as an input for FIB lookup. Then, the prefix *p* is returned as LMP. Since its critical distance is 2, *p* is expanded by two levels. The expanded prefix /com/food/meat will be stored in the cache. Critical distance-based prefix expansion guarantees that the cache lookup result is always correct. For any input name which should be matched with *q*, *r*, or *s*, it cannot be matched with /com/food/meat in the cache.

The *component level* of a prefix or a name is the number of components in it. For example, the component level of /a/b/c/d is 4. If the component level of a given Interest name is shorter than that of the matched prefix plus its critical distance, the expansion actually does not occur. For example, suppose that a given Interest name /com/food is matched with *p* = /com. But /com cannot be expanded to the component level 3 because the component level of /com/food is 2. In that case, /com/food itself should be cached and that entry will be marked as *exact matching* instead of *prefix matching*. If the matched entry in a cache is marked as exact matching, then its component level should be the same as that of the input name.

4.2 Lookup procedure

```
Lookup(name)
name: an Interest name
1.  e = cache_lookup(name);
2.  if (e == NULL or
3.     (e.flag == exact_match and
4.      e.level != name.level) ) {   // miss
5.      e = FIB_lookup(name);
6.      if (name.level < e.level + e.cd) {
7.          e.pr = name.pr;
8.          e.flag == exact_match;
9.      } else {
10.         e.pr = expand(name, e.cd);
11.         e.flag == prefix_match;
12.     }
13.     cache_insert(e);
14. }
15. return e.face;
```

Figure 4. Lookup procedure

Figure 4 describes the lookup procedure for a given Interest name. First, this procedure looks up the cache. If the cache hit occurs, it checks whether the hit entry is marked as an exact_match

or not. If so, the cache hit is finally accepted only when the component level of the Interest name is the same as that of the retrieved cache entry. In case of cache miss, FIB is looked up and the matched entry is inserted into the cache. The matched entry should be marked either as exact_match or not before being inserted.

4.3 Organization of name lookup

Figure 5 shows an overall name lookup structure which has a prefix cache. On a cache miss FIB is looked up with the given Interest name. As lookup results, a name prefix and the associated critical distance are extracted. Then, the expansion logic expands the given Interest name with the critical distance. If no expansion occurs, the match type should be marked as E (Exact match). Otherwise, it should be marked as P (Prefix match).

The prefix cache consists of TCAM and SRAM. TCAM is an associative memory which facilitates the function for finding the LMP for a given Interest name. For a matched entry, the corresponding entry in SRAM is consulted. In case that the match type is E, it is finally decided as matching only when its level is the same level as the component level of the Interest name.

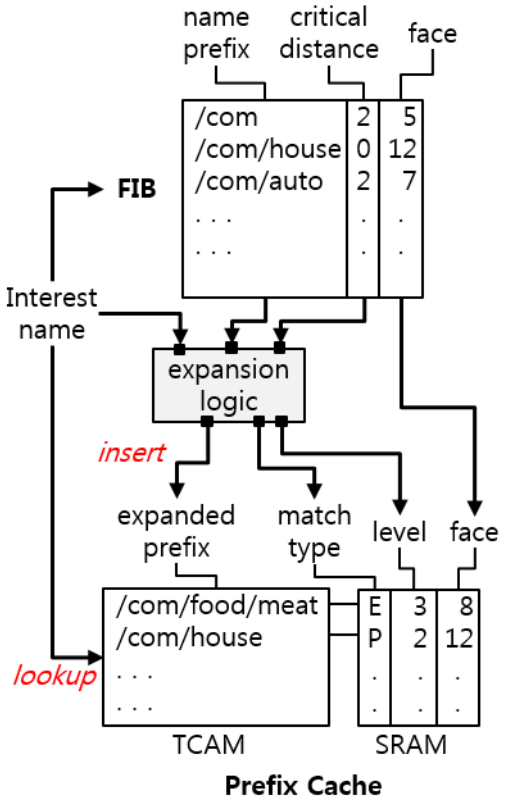

Figure 5. Lookup structure with prefix cache

The critical distance-based prefix expansion has the advantage that there is no restriction on the organization of FIB. Whatever the FIB is constituted from, it is sufficient if it contains critical distance for each name prefix to determine how long the prefix should be expanded. FIB may be constituted by hashing or even TCAM. Unless it has critical distances, it is necessary to use the trie-based structure to determine the length of prefix expansion.

5. Performance evaluation

5.1 Simulation environments

The proposed scheme was simulated and evaluated with a FIB and several random traces. Since there is no real-world FIB for NDN, we used dmoz name set instead [24]. The dmoz name set is a large set of urls for open content directory. We extracted name prefixes from a dmoz name set of March 12/2017. The urls were transformed into name prefixes. For example, http://auto.com/body/window/ was transformed into /com/auto/body/window/. Table 1 shows the characteristics of prefixes in the FIB which was used in simulation. The *prefix level* of a prefix is the number of ancestor prefixes (including itself), which should be distinguished from the component level. For example, if there exist two prefixes, /a and /a/b/c/d, then the prefix level of /a is 1 and the prefix level of /a/b/c/d is 2. However, the component level of /a/b/c/d is 4.

Table 1. Characteristics of FIB

# prefixes	3,055,734
Max. component level	18
Max. prefix level	9
Highest frequent component level	3
Highest frequent prefix level	1

The input traces were randomly generated using the prefixes in FIB. Each Interest name was made by the process of appending several random suffixes to a randomly selected prefix. The traces were generated so as to follow Zipf's law. For our experiment, 4 traces each of which had one million names were generated on $\alpha = \{ 0.7, 0.8, 0.9, 1.0 \}$.

5.2 Experiment results

Figure 6 shows the distribution of prefix level and component level. Most prefixes are at prefix level 1 (72.3%), and the prefixes at level 2 and level 3 account for 13.9% and 8.4%, respectively.

Figure 7 shows the distribution of critical distance. Most prefixes are at critical distance = 0, i.e., leaf prefixes (92.5%). The prefixes at critical distance = 1 and 2 account for 5.7% and 1.2%, respectively.

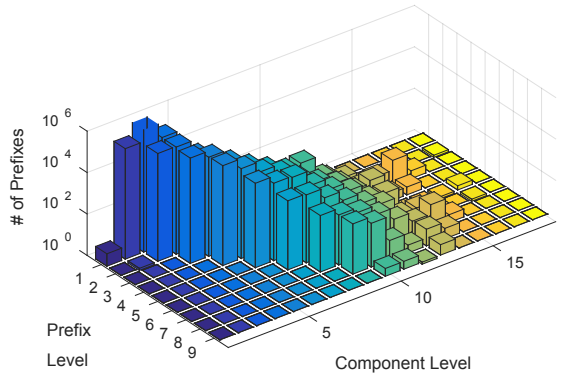

Figure 6. Distribution of prefix level and component level

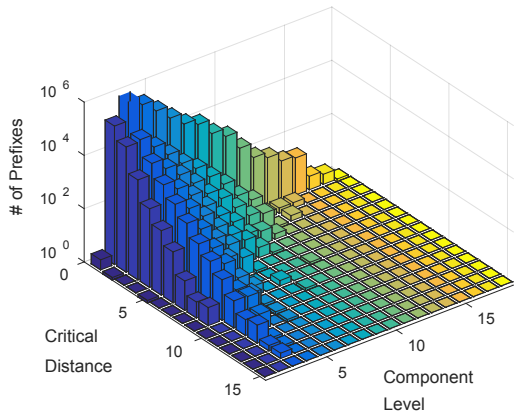

Figure 7. Distribution of critical distance

In Figure 8, the cache hit ratios at $\alpha = 0.8$ are compared as the cache size is increased. EPC denotes *Expanded Prefix Caching* which is the proposed scheme. LPCO denotes *Leaf Prefix Caching Only* and NPC denotes *No Prefix Caching*. In LPCO only the leaf prefixes are cacheable and non-leaf prefixes should not be cached to avoid incorrect hits. In NPC the Interest names are cached instead of the matched prefixes, and cache lookup is performed as exact matching. NPC shows the worst hit ratio among them since it does not exploit spatial locality. The proposed scheme, EPC, shows the best hit ratio, and it achieves nearly 0.999 at 8K of cache size.

The cache hit ratios of EPC, LPCO, and NPC are presented in Figure 9, 10, and 11, respectively. The higher α-based trace, the higher cache hit ratio is observed. As the cache size is increased, EPC and LPCO converges to some hit ratio irrespective of α whereas there is no such effect in NPC. It can be explained by the fact that NPC does not exploit spatial locality.

When a cache hit occurs, the matched entry is either E (Exact Match) or P (Prefix Match). The Prefix Match is either Leaf Prefix Match or Non-leaf Prefix Match. As shown in Figure 12, there are some Non-leaf Prefix Matches in case of EPC, which accounts for the effectiveness of critical distance-based prefix expansion. LPCO and NPC do not have any Non-leaf Prefix Match, and even NPC has only Exact Matches.

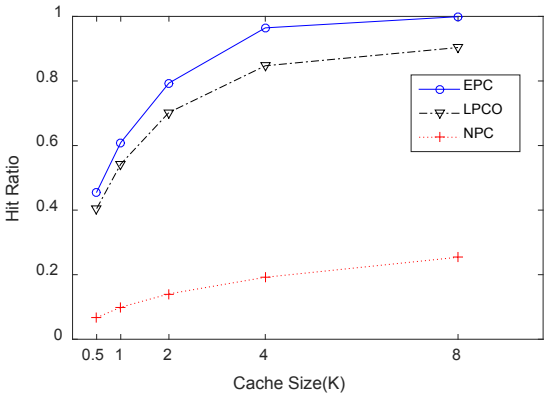

Figure 8. Cache hit ratio at α=0.8

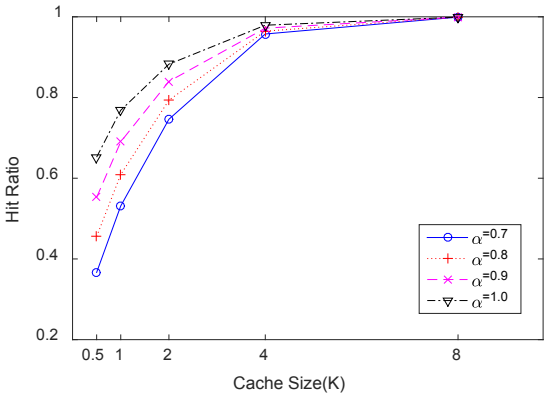

Figure 9. Cache hit ratio of EPC

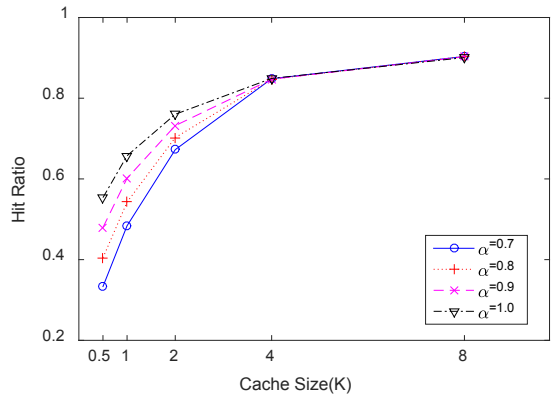

Figure 10. Cache hit ratio of LPCO

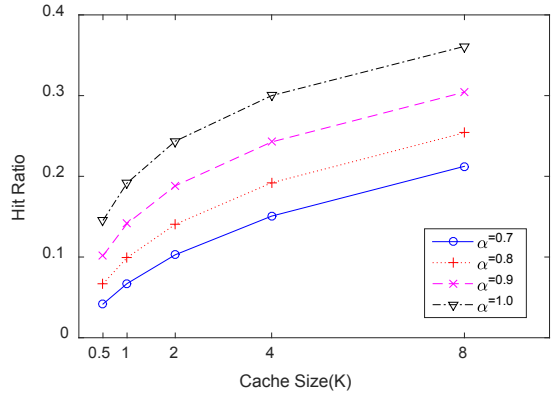

Figure 11. Cache hit ratio of NPC

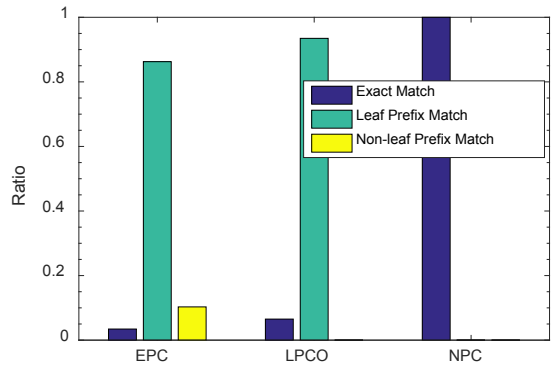

Figure 12. Ratio of matching type

6. Conclusion

The name lookup for Interest packet forwarding can be improved by prefix caching. The latest FIB lookup result is stored in a prefix cache and the next lookup can be performed faster in the cache. Instead of storing a lookup key, that is, an Interest name, the cache stores a prefix which covers some name space.

Though the prefix cache gives better performance than a name cache, it has a problem that produces incorrect results when caching non-leaf prefixes. We proposed a technique which expands matched prefixes based on critical distances to avoid incorrect results. The technique can be adopted for any forwarding engine using a simple expansion logic irrespective of the FIB lookup scheme. The simulation results show that our prefix caching scheme has a higher hit ratio than leaf prefix caching or no prefix caching.

REFERENCES

1. Akhbarizadeh, M. J. & Nourani, M. (2004, August). Efficient prefix cache for network processors. In *Proceedings. 12th Annual IEEE Symposium on High Performance Interconnects*, *2004* (pp. 41-46). IEEE.

2. Bari, M. F., Chowdhury, S. R., Ahmed, R., Boutaba, R. & Mathieu, B. (2012). A survey of naming and routing in information-centric networks, *IEEE Communications Magazine*, *50*(12).

3. Chen, X., Zhang, G. & Cui, H. (2018). Investigating Route Cache in Named Data Networking, *IEEE Communications Letters*, *22*(2), 296-299.

4. Florian, V. & Neagu, G. (2018). Towards an IoT Platform with Edge Intelligence Capabilities, *Studies in Informatics and Control*, *27*(1), 65-72.

5. Kasnavi, S., Berube, P., Gaudet, V. & Amaral, J. N. (2008). A cache-based internet protocol address lookup architecture, *Computer Networks*, *52*(2), 303-326.

6. Kim, J., Ko, M. C., Nam, J. & Kim, J. (2014). Bitmap-based Prefix Caching for Fast IP Lookup, *KSII Transactions on Internet and Information Systems* (*TIIS*), *8*(3), 873-889.

7. Lee, J. & Lim, H. (2016). A new name prefix trie with path compression. In *IEEE International Conference on Consumer Electronics-Asia (ICCE-Asia)* (pp. 1-4).

8. Lee, J., Shim, M. & Lim, H. (2016). Name prefix matching using bloom filter pre-searching for content centric network, *Journal of Network and Computer Applications*, *65*, 36-47.

9. Li, D., Li, J. & Du, Z. (2016, June). An improved trie-based name lookup scheme for Named Data Networking. In *IEEE Symposium on Computers and Communication (ISCC)*, *2016* (pp. 1294-1296).

10. Li, Z., Liu, K., Liu, D., Shi, H. & Chen, Y. (2017). Hybrid wireless networks with FIB-based Named Data Networking, *EURASIP Journal on Wireless Communications and Networking*, *2017*(1), 54.

11. Liu, H. (2001). Routing prefix caching in network processor design. In *Proceedings. Tenth International Conference on Computer Communications and Networks, 2001* (pp. 18-23). IEEE.

12. Pan, J., Paul, S. & Jain, R. (2011). A survey of the research on future internet architectures, *IEEE Communications Magazine*, *49*(7).

13. Saxena, D., Raychoudhury, V., Suri, N., Becker, C. & Cao, J. (2016). Named data networking: a survey, *Computer Science Review*, *19*, 15-55.

14. Shubbar, R. & Ahmadi, M. (2017). Efficient name matching based on a fast two-dimensional filter in named data networking, *International Journal of Parallel, Emergent and Distributed Systems*, 1-19.

15. Sun, Y., Egi, N., Shi, G. & Wu, J. (2012, December). Content-based route lookup using CAMs. In *Global Communications Conference (GLOBECOM)*, *2012 IEEE* (pp. 2677-2682).

16. Yuan, H., Crowley, P. & Song, T. (2017). Enhancing Scalable Name-Based Forwarding. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems* (pp. 60-69).

17. Yuan, H. & Crowley, P. (2015, May). Reliably scalable name prefix lookup. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, *2015* (pp. 111-121).

18. Wang, Y., Dai, H., Jiang, J., He, K., Meng, W. & Liu, B. (2011, December). Parallel name lookup for named data networking. In *Global Telecommunications Conference (GLOBECOM 2011), 2011 IEEE* (pp. 1-5).

19. Wang, Y., He, K., Dai, H., Meng, W., Jiang, J., Liu, B. & Chen, Y. (2012). Scalable name lookup in NDN using effective name component encoding. In *Proc. of the International Conference on Distributed Computing Systems (ICDCS)* (pp. 688-697).

20. Wang, Y., Pan, T., Mi, Z., Dai, H., Guo, X., Zhang, T., Liu, B. & Dong, Q. (2013, April). Namefilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters. In *Proceedings IEEE INFOCOM*, *2013* (pp. 95-99).

21. Wang, Y., Xu, B., Tai, D., Lu, J., Zhang, T., Dai, H., Zhang, B. & Liu, B. (2014, May). Fast name lookup for named data networking. In *IEEE 22nd International Symposium of Quality of Service (IWQoS)*, *2014* (pp. 198-207). IEEE.

22. Wang, Y., Qi, Z., Dai, H., Wu, H., Lei, K. & Liu, B. (2017, May). Statistical Optimal Hash-based Longest Prefix Match. In *Proceedings of the Symposium on Architectures for Networking and Communications Systems* (pp. 153-164).

23. Xylomenos, G., Ververidis, C. N., Siris, V. A., Fotiou, N., Tsilopoulos, C., Vasilakos, X., Katsaros, K. V. & Polyzos, G. C. (2014). A survey of information-centric networking research, *IEEE Communications Surveys & Tutorials*, *16*(2), 1024-1049.

24. http://www.dmoz.org/.