A Name-to-Hash Encoding Scheme for Vehicular Named Data Networks

Hakima Khelifi, Senlin Luo, Boubakr Nour, Hassine Moungla

A Name-to-Hash Encoding Scheme for Vehicular Named Data Networks

Hakima Khelifi*, Senlin Luo*, Boubakr Nour‡, and Hassine Moungla§¶
*School of Information and Electronics, Beijing Institute of Technology, Beijing, China
‡School of Computer Science, Beijing Institute of Technology, Beijing, China
§LIPADE, Paris Descartes University and Sorbonne Paris Cite University, France
¶CNRS, UMR 5157, Mines Telecom Institute, Telecom SudParis, CEA Nano-Innov Saclay, France
Emails: {hakima, luosenlin, n.boubakr}@bit.edu.cn, hassine.moungla@parisdescartes.fr

***Abstract*—In contrast to the host-centric model where the communication is directed using the destination address, Information-Centric Networking (ICN) adopts the content name as the pillar network element to provide data discovery and delivery process, as well as in other network functionalities. Named Data Networking (NDN) is an active ICN project that uses hierarchical unbounded names. These names are used in both interest and data packets and other data structures that may consume more memory with long lookup time. This paper targets the naming aspect in vehicular named data networks and proposes a Name-to-Hash Encoding scheme. The idea consists of hashing each name components separately to a fixed length, then perform a heuristic Wu-Manber-like algorithm lookup process. The former process enhances the NDN to consume less memory compared to hierarchical names, the latter process provides a fast lookup time. We have evaluated the proposed scheme against different related solutions using real domain datasets. Both theoretical analysis and experiments prove that the proposed scheme is efficient in terms of complexity, memory consumption, and lookup time.**

***Index Terms*—Information-Centric Networking (ICN), Vehicular Named Data Networks (VNDN), Naming Scheme**

I. Introduction

Vehicular Ad hoc Network (VANET) [1] has been developed to improve driving safety and manage traffic condition. However, several challenges have been found due to the use of the TCP/IP communication model, especially from mobility, routing, and security perspective. Thus, several researchers show their efforts to propose various architectures such as Information-Centric Networking (ICN) [2]. ICN is a new communication paradigm that uses names as the main component to retrieve the content instead of IP addresses.

Named Data Networking (NDN) [3], [4] is one of the most active ICN implementations. NDN uses human-readable hierarchical names to identify and deliver content. It uses two types of packets: interest and data packets. Interest packet is a request, triggered by the consumer to find content; The data packet is a response of an interest packet. In NDN, each node maintains three tables: Pending Interest Table (PIT) – to keep trace of unsatisfied interests, Forwarding Information Base (FIB) – to select the most convenient interface to forward interest upstream, and Content Store (CS) – to locally store the content.

Applying NDN on top of VANET has several advantages [5]–[8]. By using the content name instead of the host identifier, NDN can support a huge amount of information that may be exchanged between vehicles and roadside units. The mobility is also facilitated using the Interest-Data exchange model [9], and enhanced by integrating in-network caching feature [10]. In ICN-based networks, the network layer is aware of the content delivery. Indeed, any node may cache and serve the content from its local cache store careless of the original producer reachability. Moreover, NDN secures the content itself rather than securing the communication channel [11].

It is important to recall that all these features are built upon the content name. The efficiency of the naming scheme (lookup process, memory consumption, etc.) has a direct impact on network performance. NDN may have long names to identify services and content, which leads to unbounded (variable length) names. These long names may consume the memory of both interest and data packet as well as PIT, FIB, and CS tables. In addition, NDN has a search intensive architecture, where the three different tables are consulted before forwarding interest packets upstream or data packets downstream. FIB lookup using Longest Name Prefix Match (LNPM) requires to search in at hundreds of characters to find a match. This is not suitable for delay-sensitive VANET applications. Hence, the time to search in NDN data structures and the occupied memory to store names must be optimized to a high degree.

The aforementioned issues are the main motivation to derive this research. Indeed, we propose in this paper an efficient name compression and high-performance naming lookup method, namely *N*ame-to-*H*ash *E*ncoding (NHE) scheme. NHE aims at reducing the memory cost and speeding up the name lookup process, it consists of two steps: name compression, and name lookup process.

The remainder of this paper is organized as follows. Section II reviews the NDN naming efforts and the related lookup research. Section III details the proposed NHE scheme, principles, and its components. In Section IV, we present the simulation and the experimental results in terms of memory compression and lookup time using three different datasets. Finally, we conclude the paper in Section V.

II. Background & Related Works

A. Naming Overview

NDN uses the content name in all network-related features (e.g., content discovery, delivery, mobility management, security, etc.). Thus, it assigns to each piece of content a human-readable, unique, and location-independent name. The forwarding plane uses this name to forward and deliver data back to requesters without knowing host addresses. ICN architecture can support various naming schemes including hierarchical, flat or hash names, attribute-value, and hybrid names. Focusing on NDN-based VANETs, the existing naming schemes can be categorized into hierarchical and hybrid names. Hierarchical names include a list of component to identify content separated by a delimited element (e.g., slash ('/'), dot ('.')); as an example, `/maps/cn/beijing/` is a hierarchical name refers to the map of *China, Beijing*, where `cn`, `maps`, and `beijing` are three components that construct the name. Hybrid names combine at least two of the aforementioned naming schemes to provide efficient names [12]. However, the selection of the used scheme and how to merge them is a challenging task and may affect the naming performance.

Due to the fact that names are the main element to derive the communication in the NDN-based networks, different challenges and issues may be raised, including:

- *Processing Time and Throughput:* contrary to fixed-length IP addresses, NDN provides unbounded names without imposing any design regulations to name designing. Arbitrarily long names lookup process is a challenging task that may require a linear time instead of a fixed time.
- *Longest Name Prefix Matching:* In contrast to classless IP addresses, NDN names maintain a group of delimited textual components. The longest prefix matching must match the entire prefix (all components until the end character) rather than any digit in IP. Consequently, implementing traditional prefix matching in NDN may not produce effective results.

The use of long, variable, and unbounded names may consume more memory and affect the lookup process, hence affecting the network scalability, especially in highly mobile networks such as VANET. Thus, it is important to design a careful naming scheme that takes the name length into consideration with a fast lookup process.

B. NDN Lookup Research

Various efforts have been shown in order to provide an efficient NDN lookup process. The proposed methods can be regrouped as follows:

(1) Hash Table-based Methods: works in [13] and [14] store hierarchical names in a a hash table. The lookup process consists of finding a match for for a key against a set of key's prefixes in the hash table. However, these solutions can work only for specific names, and consume the memory of NDN tables caused by the redundant information in the hash table. Moreover, the performance of these methods is highly dependent on the names' length, which is challenging to apply to long names such as those used in vehicular networks.

(2) Bloom Filter-based Methods: are an alternative methods to describe set of names and a generalized form of hashing with trade-offs among memory occupation. The prefix Bloom filter proposed in [15] aims to store prefixes that share the same first-level component in the same cache-line sized Bloom filter. The Bloom filter is increased if the number of suffixes exceeds the Bloom filter capacity. There are cases where multiple Bloom filter expansions are required to store all the prefixes. Although a dataset that requires Bloom filter expansion at every name component level can be generated (unlikely to happen in practice), a certain number of prefixes have large numbers of suffixes.

(3) Trie-based Methods: efforts in [16] and [17] introduce name encoding solutions with a Trie-based approach in order to compress the name characters with a state transition array, represent the prefix tree structure, and optimize the traverse operations. However, a trie-based approach has several issues: because the tree uses hashing codes for entity storage, this may cause collision; and searching name in the tree has the time $O(n)$, which is the time complexity of tree searching that is long and unsuitable for applications that require a fast name lookup.

As stated in this section, the biggest weakness of hashing the whole name is that it cannot perform a prefix matching especially for long names. Although the method of name decomposition tree has the ability to do so, it suffers from long lookup time. Hence, we propose to use a hash function in order to reduce the length of NDN names and thus provide a short lookup time while keeping the prefix matching ability.

III. NHE: Name-to-Hash Encoding Scheme

In the following section, we present our proposed solution called NHE scheme. NHE aims to reduce the memory consumption in the packets as well as in PIT, CS, and FIB tables; and accelerate the lookup process. NHE is based on two main steps: The first step is name compression, and the second step is name lookup, which is based on multiple string matching concept (*Wu-Manber-like algorithm*) [18].

We must highlight that the compression and lookup steps are local to the node. Thus, the collision of hashed names is limited and becomes negligible, also the application dependent name design principles remain intact.

A. System Model

We consider a connected vehicular network represented by a directed graph $G = (V, L)$, comprising set vehicle nodes V, and links L. We denote l_{v_i,v_j}, the link connection $l \in L$ between two vehicle nodes $(v_i, v_j) \in V$. Each vehicle node can be a consumer S, producer R, or intermediate node I with caching abilities. Nodes may exchange two types of packets: interest and data packets. Both interest and data packets carry the requested/delivered content name. NDN uses a hierarchical naming scheme $n = (n_i \cdot /)$, where n_i is a name component of

TABLE I: Summary of notations used in the paper.

Notation	Meaning
$V = \{S \cup I \cup R\}$	Vehicles (Consumer, Intermediate, Producer)
L	Set of links
l_{v_i,v_j}	Connection link between two vehicles
$n = (n_i \cdot /)$	Name of the content
n_i	Name component of n
c	Compressed name
c_i	Compressed name component
x	Number of decomposed name components
N	Number of names
$P_{collision}$	Probability of two hashed names
m	Length of the smallest name
B	List of last character of the compressed name
B'	List of first character of the compressed name

n. Table I illustrates a summary of the used notations in this paper.

Interest and Data Forwarding Process: When a vehicle node receives an interest packet, it checks first if the requested content exists in its local CS. If a match is found, it will forward back the data through the incoming interface, and discard the interest packet. Otherwise, the vehicle node checks if the requested name exists in the PIT table. If a match is found, which means that a similar interest has already been forwarded but no data has been returned yet, the vehicle node will add the incoming interface to the PIT entry and discard the interest packet. If no match is found, the vehicle node will consult the FIB table to find the next-hop, and add a new PIT entry. Otherwise, the interest packet is dropped.

Similarly, when a vehicle node receives a data packet, it checks if the name already exists in the PIT. If a match is found, it stores the data in the local CS, forwards this data to all interfaces listed in the PIT entry, then deletes the associated entry. Otherwise, the vehicle node considers the packet unsolicited and discard it.

B. Name Compression Algorithm

As mentioned before, NDN names are variable length and unbounded, using such long names consume more memory space, and slow down the lookup process. To end-up with short names and a high-performance lookup process, we propose to use a hash function to reduce the length of NDN names.

First of all, we decompose our name n into a collection of name components separated by the slash delimiter. We use CRC32 (a 32-bit CRC algorithm) to hash each unequal-length name component n_i into a fixed-length string c_i, that represented by a binary code called: *compressed name component*. Then, these compressed components are concatenated successively to make a single string c, named: *compressed name* that represents the original complete compressed name. Algorithm 1 shows the name compression process.

The reason to use the CRC32 algorithm is a trade-off between the computational complexity and the memory requirement of the hash codes. Moreover, it has a simple implementation and can be done at the line rate. Hence, the computation of CRC32 can be performed while reading the packet header, without adding any extra delay. In addition, the result of hashing a component using CRC32 is a $4-byte$ fixed-length string. The compressed name is represented by $4x\ byte$ string, where x is the number of decomposed name components. The probability of two hashed names to be the same in CRC32 presented as follow:

$$P_{collision} = 1 - (\frac{2^k - 1}{2^k})^{N-1}$$

where N is the number of names.

However, in our work we treat the name by components, therefore the collision probability becomes:

$$P_{collision} = \sum_{j=1}^{J_{max}} (1 - (\frac{2^k - 1}{2^k})^{P_j^{N-1}})^j$$

where j is the number of name components, P_j is the proportion of names composed of j name components, J_{max} is the maximum number of decomposed name components of the name in a list of N names, and $k = 32$ (CRC32). In this way, a lower collision probability (false positive) is obtained.

To illustrate a naming compression example, detailed in Figure 1, we have the following name `/traffic/Highway101/north/{400, 410}/{1323201600, 1323205200}/speed/19375887` to fetch information from the north part of a highway. We decompose the textual name to seven components that are separated by the slash delimiter. Then, we use the CRC32 algorithm to hash each component, and concatenate them to make a single hashed string that contains only 28 *bytes* represented by hexadecimal code of `0x55602630f3238601dc2a30c29642aff8d31f0ac80f26fef6c2fed8d1`. This compressed name is used in all of CS, PIT, and FIB tables to reduce their sizes, as well as used in received interest and data packet to speed up the lookup process.

Algorithm 1 Name Compression Algorithm

Input: n: Content name
Output: c: Compressed name
1: **for** (n_i in $split(n, /)$) **do**
2: $c_i := CRC32(n_i)$;
3: $c := c \cdot c_i$;
4: **end for**
5: **return** c;

C. Name Lookup Algorithm

Finding a name in the entire table is not an easy task from the view of memory and the time consumption, especially if the table has a large set of names, this process is similar to the multiple string matching problems. The second step in our proposed scheme is the name lookup algorithm that is based on using the Wu-Manber-like algorithm for multiple string matching.

The Wu-Manber algorithm [18] aims to find the occurrence pattern P in given text T, where P and T are all strings,

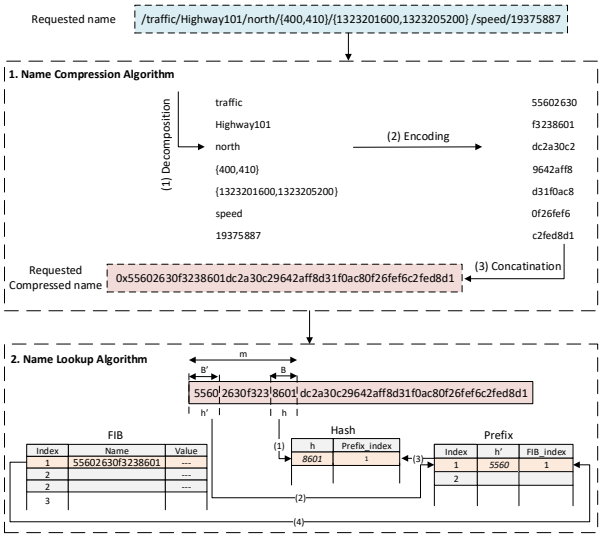


Fig. 1: NHE Process

using heuristics to skip portions of the input text. However, we have simplified this algorithm and made changes so it can be adapted to finding names in a large name set. We have used only two tables from three tables of the Wu-Manber algorithm:

1) HASH TABLE: contains a list of last B character of the compressed name. Where each Hash entry is associated with a list of indexes of PREFIX TABLE.
2) PREFIX TABLE: contains a list of first B' character of the compressed name. Where each entry also holds an associated list of indexes of FIB entries in witch B' exists.

After extracting the name from the NDN packet, and compressing it using the first step of the name compression algorithm, the matching stage is executed. A brief description of the matching stage is shown in Figure 1, and illustrated in Algorithm 2, where m is the length of the smallest name in FIB. This is updated every time a new packet with shorter name length is processed. Hence, for entries which are larger than m, only first m characters are used for matching initially. The lengths of substrings h and h' are pre-configured. h is listed into the HASH TABLE, and h' is listed in the PREFIX TABLE. List of $prefix\_indexes$ in the HASH TABLE is appended to contain the index at which h' is inserted in PREFIX TABLE. List of FIB_index in PREFIX TABLE is appended with the index of FIB table where the complete compressed name is stored.

For each new entry in FIB, if h is found in the HASH TABLE, the entry is not inserted again, its h' is inserted in PREFIX TABLE and $prefix\_index$ is updated. Similarly, for each h', if the entry in PREFIX TABLE already exists, then it is not inserted again, rather its FIB_index is updated.

The searching principle is to look for small hashes in the HASH TABLE from where the subset of FIB is further reduced by matching selected prefixes only. Where for any name that needs to be found, substring C of size m is extracted from the compressed name, and further broken into substrings h and h'. If h is not found in the HASH TABLE, then the name does not exist. If found, then the associated $prefix\_index$ list is used to find entries in the PREFIX TABLE, which has the same value as h'. If h' is not found in the PREFIX TABLE, the insertion algorithm was not executed properly. If found, the associated FIB_index list is used to find a subset of FIB entries which are then compared with the full compressed name. Hence, the actual number of strings matched is reduced to very few possible entries. This lookup algorithm can be applied and used in all three tables (CS, PIT, and FIB) without changing the core design of NDN routing.

Algorithm 2 NHE Lookup Algorithm

Input: N : requested name

```
C := compressed_name(N);
h := C.substring(m − B + 1, m);
h' := C.substring(0, B);
Subset := NULL;
for (each entry i in Hash) do
    if (hash[i].h == h) then
        prefix_index = Hash[i].prefix_index;
        for (each element j in prefix_index) do
            if (prefix_index[j].h' == h') then
                Subset := prefix_index[j].FIB_index;
            end if
        end for
    end if
end for
if (Subset != NULL) then
    for (each n in Subset) do
        if (Subset[n].Name == C) then
            return FOUND_FIB_index;
        end if
    end for
else
    return NOT FOUND;
end if
```

IV. IMPLEMENTATION & EVALUATION

We have implemented NHE and compared it against LNPM [3] and Name Character Trie (NCT) [17] in terms of time complexity, lookup time, and memory compression. The evaluation has been performed on an Intel Core 5 Duo CPU at 2.4 GHz and DDR3 SDRAM of 8 GB. Moreover, we have used three datasets: BLACKLIST[1], ALEXA[2], and SHALLALIST[3]. We have extracted 25×10^5 different domains and URLs from BLACKLIST, top 10^6 URLs form ALEXA, and 17×10^5 from SHALLALIST. Also, we have cleaned these domains and URLs to be represented as NDN names format, e.g., `google.com/map` represented as `com/google/map`.

[1] BLACKLIST: www.urlblacklist.com
[2] ALEXA: www.alexa.com
[3] SHALLALIST: www.shallalist.de

TABLE II: Memory Compression.

Dataset	Total Names	Total length (MBytes)	NCT (MBytes)	NHE (MBytes)	Compression (NCT)	Compression (NHE)
ALEXA	1000000	47.41	47.41	4	0%	91.56%
SHALLALIST	1700000	95.96	84.14	7.80	12.31%	91.86%
BLACKLIST	2500000	224.86	118.36	18.53	47.36%	91.76%

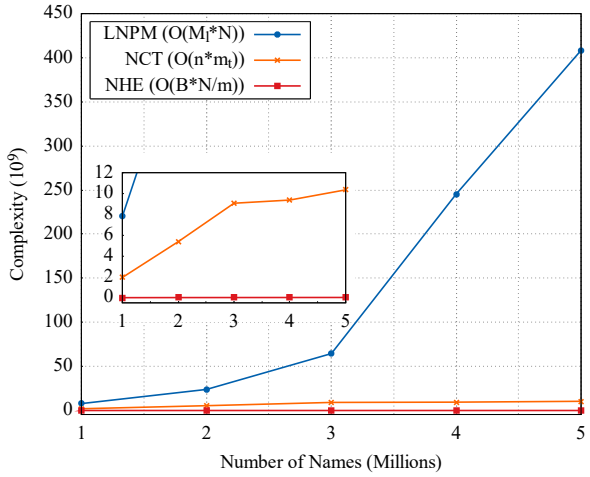


Fig. 2: Time complexity

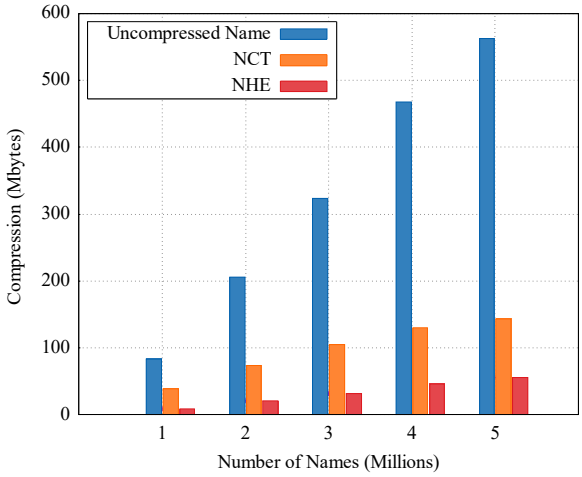


Fig. 3: Memory consumption after compression per number of names in three datasets.

A. Time Complexity Analysis

Figure 2 represents the time complexity for all of NHE, NCT, and LNPM. We can observe that the complexity in NHE is very small compared to NCT and LNPM because NHE is divided into two main algorithms: Name Compression and Name Lookup. The average complexity of both algorithms is the sum of the complexity of the two algorithms.

The complexity of Name Compression algorithm is determined by the complexity of the $CRC32$, which is of degree $O(|M|)$, where $|M|$ is the length of the name. The processing of CRC32 can be done at line rate, hence no additional time delay is required, and parity speed is improved. The complexity of Name Lookup algorithm is the complexity of Wu-Manber-like algorithm, which is given by $O(B*N/m)$, where B is the length of the character block, while N is the length of all names, and m is the shortest length of the pattern.

In NCT, the time complexity equals to $O(n * m_t)$, where n denotes the average length of a name in the whole dataset and m_t denotes the average number of descendants per node. Similarly, the time complexity in LNPM is described by $O(M_l * N)$, where M_l is the longest length of the patterns.

B. Compression Rate

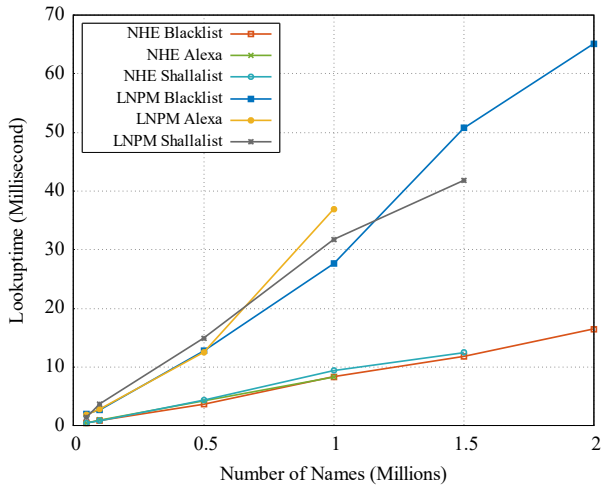
We define the compression rate by $1-s/TotalNames$, where s denotes the memory usage to store the compressed names from three datasets by NHE, and $TotalNames$ is the sum of the string length of the original names in the datasets. Table II presents the compression effect of the three domain name datasets. The analysis shows NHE compression method can achieve about 90% compression rate while NCT achieves 50% especially when the number of names increases. NHE achieves more compression overall because of restricts all name components to $4-bytes$. Because NHE is a local process on NDN node, the compression is also local, hence the global collision is not a challenge.

Figure 3 illustrates the memory consumption after compression the three domain name datasets. We can notice that when the number of domains increases, the number of compressed names in NHE increases quite slowly compared to NCT. Therefore, the memory cost in NHE grows slower than NCT, which proves that NHE is efficient to maintain the memory despite the domain set (either small or large size set).

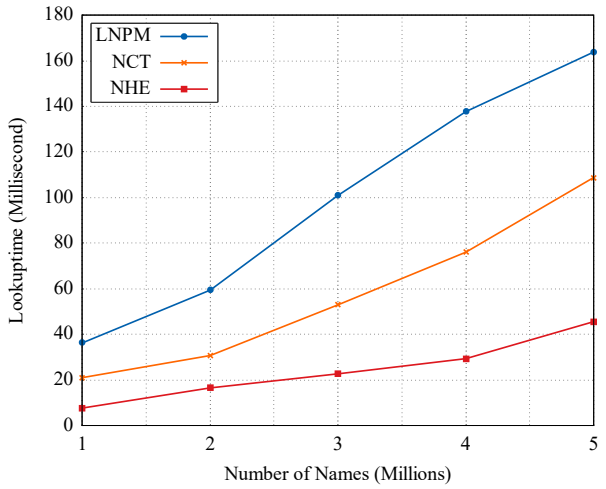
C. Lookup Time

Figure 4a shows the lookup time for NHE and LNPM in three datasets BLACKLIST, ALEXA, and SHALLALIST. We can observe that NHE is faster than LNPM in all of the tested datasets. Also, when the number of names increases, the lookup time increases slowly in NHE compared to LNPM.

Figure 4b demonstrates the previous results. It shows the lookup time in NHE, LNPM, and NCT, where we treat the three datasets as individual FIB tables. For small datasets, the performance is almost comparable with NHE, LNPM, and NCT. However, when the size of the tables grows, the difference becomes significantly large, where NHE increases linearly compared to LNPM and NCT. Thus, and due to the fact that NHE utilizes a heuristic method to select only the necessary parts in the name by applying two tables and performing a better prefix matching. While LNPM matches names character by character, and NCT uses the tree searching algorithm.



(a) NHE and LNPM lookup time.



(b) NHE, NCT and LNPM lookup time.

Fig. 4: Lookup time calculation

V. Conclusion

The name and name lookup process are considered as a critical part in NDN. The performance of other functionalities is based on the efficiency of the naming scheme. In VANET, many applications have a time-critical to receive and forward the data in optimal time. As NDN names can be unbounded and variable length and the lookup time based on these names, this will end up with several problems in the memory consumption and the lookup time. In this paper, we proposed a new Name-to-Hash Encoding (NHE) scheme for vehicular named data networks. NHE aims at reducing the memory overhead while accelerating lookup speed regardless of the length of names. The process is based on two steps: (1) a name compression process, and (2) name lookup process based on the Wu-Manber-like algorithm. We evaluated the proposed scheme against related solutions using real domain datasets, where both theoretical analysis and simulation experiments prove the efficiency of NHE in terms of complexity, memory consumption, and lookup time.

Acknowledgements

The work of S. Luo was supported by the National 242 Project under Grant No. 2017A149. Dr. Luo is corresponding author.

References

[1] S. Boussoufa-Lahlah, F. Semchedine *et al.*, "Geographic routing protocols for Vehicular Ad hoc NETworks (VANETs): A survey," *Vehicular Communications*, 2018.

[2] C. Fang, H. Yao *et al.*, "A survey of mobile information-centric networking: Research issues and challenges," *IEEE Communications Surveys & Tutorials*, 2018.

[3] D. Saxena, V. Raychoudhury *et al.*, "Named data networking: a survey," *Computer Science Review*, 2016.

[4] B. Nour, K. Sharif *et al.*, "A Survey of Internet of Things Communication using ICN: A Use Case Perspective," *Computer Communications*, 2019.

[5] H. Khelifi, S. Luo *et al.*, "Named Data Networking in Vehicular Ad hoc Networks: State-of-the-Art and Challenges," *IEEE Communications Surveys and Tutorials*, 2019.

[6] H. Khelifi, S. Luo *et al.*, "An Optimized Proactive Caching Scheme based on Mobility Prediction for Vehicular Networks," in *IEEE Global Communications Conference (GLOBECOM)*, 2018, pp. 1–6.

[7] H. Khelifi, S. Luo *et al.*, "LQCC: A Link Quality-based Congestion Control Scheme in Named Data Networks," in *IEEE Wireless Communications and Networking Conference (WCNC)*, 2019.

[8] H. Khelifi, S. Luo *et al.*, "Reputation-based Blockchain for Secure NDN Caching in Vehicular Networks," in *IEEE Conference on Standards for Communications and Networking (CSCN)*. IEEE, 2018, pp. 1–6.

[9] B. Nour, K. Sharif *et al.*, "A Distributed ICN-Based IoT Network Architecture: An Ambient Assisted Living Application Case Study," in *IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2017, pp. 1–6.

[10] B. Nour, K. Sharif *et al.*, "NCP: A Near ICN Cache Placement Scheme for IoT-based Traffic Class," in *IEEE Global Communications Conference (GLOBECOM)*, 2018.

[11] H. Khelifi, S. Luo *et al.*, "Security and Privacy Issues in Vehicular Named Data Networks: An Overview," *Mobile Information Systems*, 2018.

[12] B. Nour, K. Sharif *et al.*, "M2HAV: A Standardized ICN Naming Scheme for Wireless Devices in Internet of Things," in *International Conference Wireless Algorithms, Systems, and Applications (WASA)*. Springer, 2017, pp. 289–301.

[13] W. So, A. Narayanan *et al.*, "Named data networking on a router: Fast and dos-resistant forwarding with hash tables," in *Symposium on Architectures for networking and communications systems*. IEEE, 2013.

[14] H. Yuan and P. Crowley, "Reliably scalable name prefix lookup," in *ACM/IEEE Symposium on Architectures for networking and communications systems*, 2015.

[15] D. Perino, M. Varvello *et al.*, "Caesar: A content router for high-speed forwarding on content names," in *ACM/IEEE symposium on Architectures for networking and communications systems*, 2014.

[16] Y. Wang, K. He *et al.*, "Scalable name lookup in NDN using effective name component encoding," *International Conference on Distributed Computing Systems*, 2012.

[17] S. Feng, M. Zhang *et al.*, "A Fast Name Lookup Method in NDN Based on Hash Coding," pp. 575–580, 2015.

[18] Y. Lu, Y. Liu *et al.*, "A Data-Deduplication-Based Matching Mechanism for URL Filtering," in *IEEE International Conference on Communications (ICC)*, 2018.