Efficient Lookup Solutions for Named Data Networks

Swetha B., S. V. Uma

Abstract*: Named Data Networking (NDN) is afast growing architecture, which is proposed as an alternative to existing IP. NDN allows users to request the data identified by a unique name without any information of the hosting entity. NDN supports in-network caching of contents, multi-path forwarding, and data security. In NDN, packet-forwarding decisions are driven by lookup operations on content name of the NDN packets. An NDN node maintains set of routing tables that aid in forwarding decisions. Forwarding the NDN packets depend on lookup of these NDN tables and performing Longest Prefix Matching (LPM) against these NDN tables. The NDN names are unbounded and of variable length. These features along with large and dynamic NDN tables pose several challenges that include increased memory requirement and delayed lookup operations. To this end, there is a need for an efficient data structure that support fast lookup operations with low memory overhead. Several lookup techniques are proposed in this direction. Traversing trie structures would be slow since every level of trie require a memory access. Hash tables incur additional hash computations on names and suffer from collisions. Bloom filters suffer from false positives and do not support deletions. Improving the performance of these structures can lead to a better lookup solution.This survey paper explores different lookup structures for NDN networks. Performance is measured with respect to lookup rate and memory efficiency.

Keywords: Cache store (CS), Forwarding Information Base (FIB), Longest Prefix Matching (LPM), Pending Interest Table (PIT).

I. INTRODUCTION

Named Data Networking (NDN) has evolved to be a favorable architecture for the applications of current network scenario. At recent times the Internet's client-server communication model has evolved into peer-to-peer mode of data sharing. IP facilitates to create a communication network, where packets are recognized by source and destination addresses [11]. There is a substantial rise in user-generated data, with growing demand for applications like YouTube, Bit Torrent, and social networking sites [12]. The IP architecture fails to handle the large data traffic with desired quality of service. Mobility and security is not in-built in Internet but offered as multiple patches, which sometimes may fail. To overcome these shortcomings of IP architecture, NDN is proposed as an alternate architecture where communication takes place by the exchanging Interest and Data packets.

NDN leverages in-network caching, support mobility and multicasting, content level security and scalability. NDN names are hierarchical in structure with variable lengths comprising of sequence of delimited components. These characteristics present challenges to the fast lookup operations that rely on longest prefix matching (LPM) of content names. Due to frequent changes in the content distribution, NDN routers need to perform incremental route updates. In addition to this, the need for large table size for FIB implementation call for a careful design of memory efficient lookup structures.

Several NDN forwarding structures are proposed to meet the need of fast table lookups without increasing the storage requirements. These lookup methods make use of one or more combination of data structures that include hash tables, Bloom filters, trie structures, bitmaps, and state transition arrays.

The paper is arranged into 5 sections. The general idea of NDN naming and packet forwarding architecture are described in section II. Section III describes different NDN lookup data structures. The simulation results are presented for different lookup techniques in section IV. We conclude the paper in Section V.

II. NDN OVERVIEW

This section explains NDN naming, NDN communication and NDN forwarding.

A. NDN naming

Each NDN packet is assigned a unique application dependent name. Based on the names, NDN packets are forwarded. NDN names are hierarchically structured delimited components. As per NDN design, name is a reversed domain name, e.g. in/icdecs2019/pdf/Conclave.pdf where in/icdecs2019 is reversed domain name of icdecs2019.in, and pdf/Conclave.pdf is the directory path of the content [1]. Hierarchical structure allows name aggregation and longest prefix matching.

B. NDN communication

To initiate the data exchange, it is required that the consumer would send an *Interest packet* with the header carrying the desired content name. This name is used by the routers to dispatch the Interest packet in the network. Nodes that contain requested data forward the *Data packet* that contains the desired data and the name.

Revised Manuscript Received on December 05, 2019.
***** Correspondence Author
Swetha B., Department of Electronics and Communication, RNS Institute of Technology, Bengaluru, India. Email: swetab26@gmail.com
Dr. S. V. Uma, Department of Electronics and Communication, RNS Institute of Technology, Bengaluru, India.
Email: umakeshav2000@gmail.com

C. NDN Forwarding Architecture

NDN forwarding plane is made up of 3 tables for packet forwarding. A copy of previously answered data packets is cached to answer similar requests by Cache Store (CS). A list of unanswered Interest requests that are forwarded upstream towards the data sources is held in Pending Interest Table (PIT). Forwarding Information Base (FIB) holds the information regarding the name prefixes and their corresponding next hop interfaces.

Upon receiving the Interest request, the CS is checked to see if similar entry exists. If available, the Data packet is returned directly. If not found in CS, it looks PIT for an entry with similar Interest packet. If similar entry already exists, the requesting interface is appeneded to that PIT entry and rejects the incoming Interest packet. If not, a new PIT entry is created and recordes the requesting interface. Interest packet is dispatched to next hop faces by FIB lookup.

When Data packet is recieved, it is dispatched to all requesting interfaces. This particular PIT record is erased and caches the data in CS. Fig. 1. shows the data communication at an NDN node.

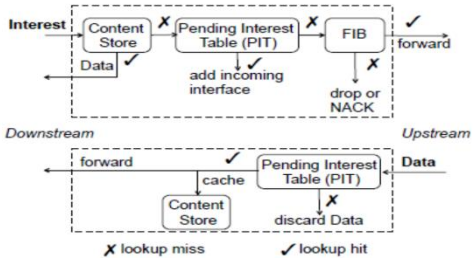


Fig. 1. NDN forwarding process.

III. NAME LOOKUP SOLUTIONS

NDN forwarding solutions for high-speed table lookup with challenges of data structures for efficient memory.

A. *Adaptive prefix Bloom Filter (NLAPB)*

NLAPB technique is proposed by Wei Quan, et al. which splits the NDN name into B-prefix and T-suffix [2]. Prefixes with a fixed size can be processed by Counting Bloom Filters (CBF). Suffixes with variable sizes can be stored using Trie structures. A hash is used to attach T-suffix Tries to B-prefix. NLAPB data structure is depicted in Fig. 2.

B-Prefixes of each length are assigned one filter. The filter size is computed according to the number of prefixes with varied lengths contributed by it.While adding or removing the prefixes, the counters analogous to hash values are incremented or decremented. Also the hash structure mapping prefixes to suffixes is updated.

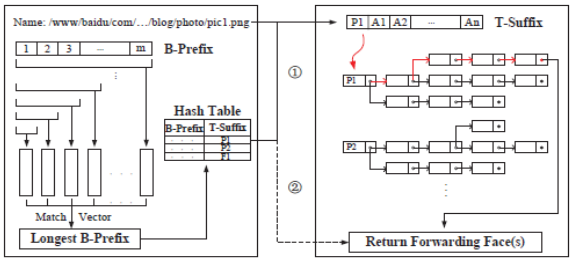


Fig. 2. NLAPB lookup structure.

While adding or removing the prefixes, the counters analogous to hash values are incremented or decremented. Also the hash structure mapping prefixes to suffixes is updated.

Each suffix tree component maps to one entry of trie. The root of the trie is connected to prefix through hash table. The components are inserted or removed along the trie knowing the root node.

During lookup operation, for B-prefix matched by CBFs, the hash table provides the suffix tree position. For short prefixes, the outgoing faces will be returned directly. With the knowledge of root information, the corresponding next hop information is returned.

By reducing the number of entries of CBFs, probability of false positive rate can be reduced. The lookup processing rate, update rate and false positive rate of NLAPB outperform that of Bloom hash, Hash table and Component trie methods.

B. Bloom Filter Assisted Hash Table

Huichen Dai, et al. proposed a structure that employs BF with hash table for LPM. It proposes the idea that a unified index can address all 3 tables namely CS, PIT and FIB [1]. This reduces the multiple lookups to only once. BFAST data structure is depicted in Fig. 3.

The *pointer* field of the entry in the index point to the entry of one of CS, PIT or FIB table. The table to which the entry belongs to is indicated by its associative *type*.

For prefix insertion, the item is inserted into CBF and then into least loaded hash bucket.

While inserting an item, it employs k auxillary CBFs to identify the hash function used to calculate the hash bucket with the smallest counter.

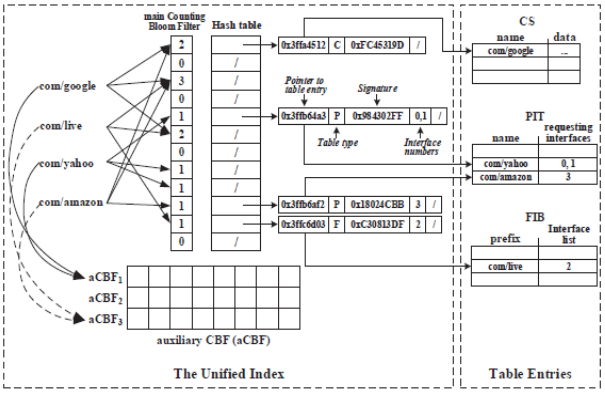


Fig. 3. BFAST data structure for NDN forwarding.

Major benefits of this structure are better LPM lookup performance, improved sclability and support to high speed incremental updates.

BFAST lookup throughput under average workload of 10M FIB reaches 2.33 M/s, obtaining 12.40x, 1.27x, 4.49x speedup over CCNx, NameFilter, and NCE, respectively.

C. Name Filter

Name Filter proposed by Yi Wang, et al. suggest the use of two stage Bloom filter [3]. To avoid performance degradation in lookup speed and memory consumption, the hash tables are replaced with Merged Bloom filter.

Bloom filters are used as identifiers for names to the same outward ports. The N Bloom filters are attached to m forwarding ports, with a bit sequence made up of aggregated m bits in the same position. Each entry in second stage filter stores a bit sequence from MSB to LSB. The AND operation on n-bit strings analogous to n hash functions decide the outgoing port. Proposed data structure is depicted in Fig. 4. Advantages of this structure include high lookup performance with reduced memory utilization, better scalability and improved update efficiency.
Compared to CharacterTrie and BlooomHash, it achieves 17 times and 1.8 times speedup.

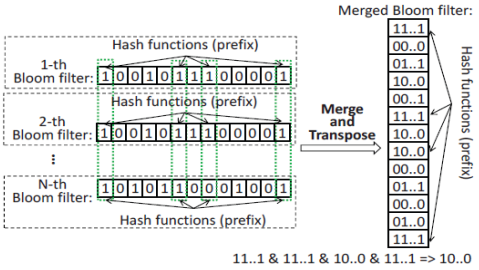


Fig. 4. Merged Bloom filter.

D.Parallel Name lookup (PNL)

PNL structure proposed by Yi Wang, et al. builds name prefix trie (NPT) of component granularity [4]. A name component is represented as an edge of NPT and each node of NPT represents lookup state. The name prefix lookup starts at the root to see if the beginning component equals any of the edges that begin from the root. If it matches, the lookup process shifts from the root to the next level node. This lookup continues until the it reaches one of the leaf nodes or the transfer condition be unsuccessful. Then the lookup process ends to output the port's number. The name lookup for NPT is shown in Fig. 5.

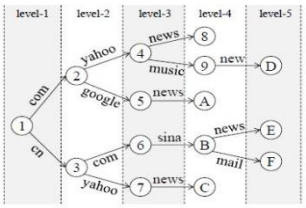


Fig. 5. NPT structure.

Different physical modules are assigned group of states of different levels in the NPT. Knowing the name's access probability, each state's access probability is calculated. Any state having access probability higher than threshold is duplicated. Original and duplicated states are assigned to different physical modules. The PNL structure is shown in Fig. 6. The dotted circles indicate duplicated states.

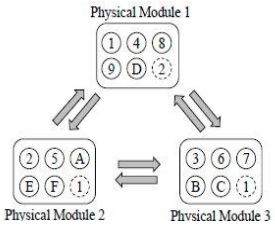


Fig. 6. PNL structure.

E. Lookup based on Name Component Encoding

This structure proposed by Yi Wang, et al. employs a code assignment mechanism where total number of codes are reduced to reduce the bytes used to denote a code [5]. Unique codes are assigned to all components in NPT. Code of the component is set as the maximum code plus one, if different codes are assigned to a component.
State Transition Arrays (STA) are used to design the Encoded Name Prefix Trie (ENPT) to accelerate longest prefix lookup and compress memory size. There are 3 types of arrays. The transition array which stores state information is indicated by the Base entry's first two bits. Transition Array has 2 types of entries, namely Indicator and transition. State's transition number is denoted by indicator. The free entries are represented by Manage Array. Code allocation mechanism and ENPT are shown in Fig. 7 and Fig.8 respectively.

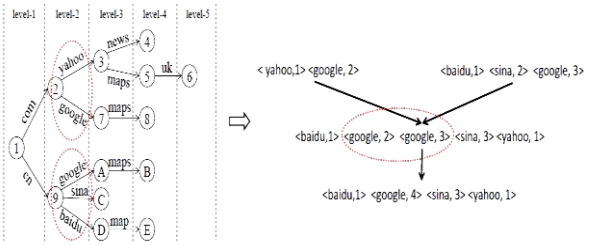


Fig. 7. Illustration of Code Allocation Mechanism.

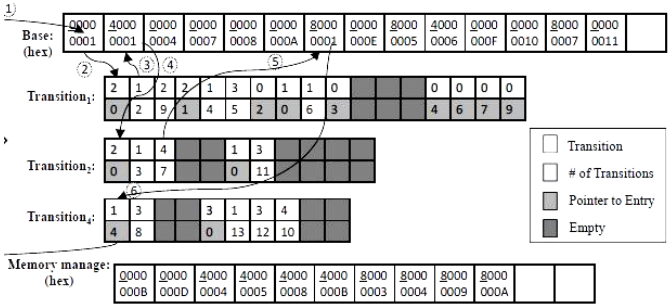


Fig. 8. State Transition Arrays for Encoded NPT.

F. Name lookup using FPGA

Yanbiao Li, et al. proposed FPGA based name lookup that employs compact data structure called Hierarchical Aligned Transition arrays (HATA) [6]. Parallelism on FPGA is added by parallel reading of different RAMs. The transitions of the same state, which are in the same level and same length are stored in different transition arrays. Based on the offset value resulting from the source state and input number,the state transition is calculated. The valid transitions are identified by the transitions stored in all ATAs at that position. An illustration of 4-stride name trie and corresponding HATA is shown in Fig. 9 and Fig. 10, respectively.

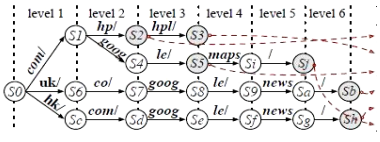


Fig. 9. A 4-stride name trie.

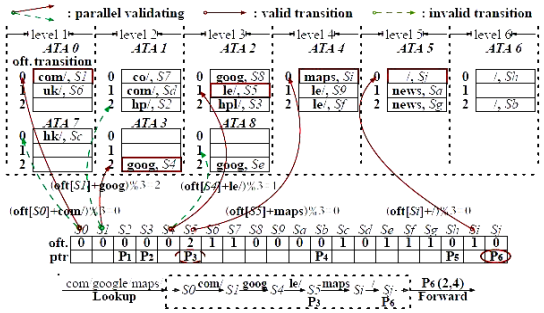


Fig. 10. HATA corresponding to name trie.

G. Name lookup using Iterative Bloom filters (IBFs)

Cristina Munoz, et al. proposed name lookup using IBFs that take advantage of iterative hash functions properties [7]. To reduce the number of elements, I(FIB)F technique splits the standard BF into m number of bit positions. A standard BF is split into d number of IBFs representing n elements and m/d bit-positions. The iterated hash outputs that belong to each field of the name serve as input to individual IBFs. Thus, name is inserted in four IBFs using a single hash function. While standard BF requires 4 hashes per element, the IBF technique require only one hash to each of the four Iterated BFs. If the top leaves of names coincide, it reduces the probability of false positives.

This technique proposes to map each interface with a separate iterated BF made of IBFs. When an interest is received, it checks the iterated hash values contained in the name with every possible outgoing interface. The IBFs of every FIB is tested for membership. Fig. 11 compares the standard BF with I(IBF)F considering four different names which employ four IBFs. Fig. 12 shows the forwarding plane at node C.

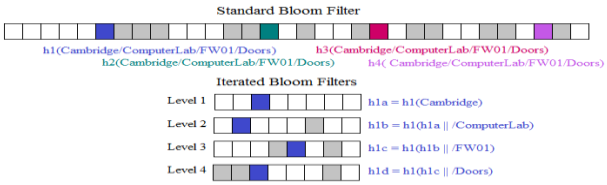


Fig. 11. Comparison of Standard Bloom filter with Iterated Bloom filter.

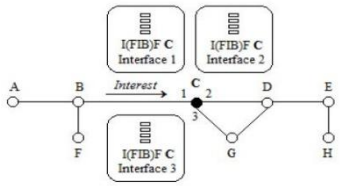


Fig. 12. Interfaces at node C that receives interest through interface 1.

H. Split name trie (SNT)

Name lookup using split name trie proposed by YunTan, et al. split the existing FIB into two reduced ones to perform longest prefix matching on them separately [8]. By eliminating the redundancies resulting from the similarity in the prefixes, the memory cost is reduced. Before performing lookup, each name component is hashed to 32-bit integer. For name lookup, the name need to be hashed to an integer string, and then perform longest prefix matching the NDN table against this integer string. These integers are used as state transitions to construct character trie stored as a Aligned Transition Array. By using all names having length smaller than n, character trie T_1 is constructed. For content names longer than *p* components, character trie T_2 is constructed after the name is split at the position *p*. Front prefix (FP) is made of components from 1 to n, and components from n+1 to the end make up Rear prefix (RP). Character trie T_2 is constructed using all RPs. Each RP in T_2 is assigned to a hash table with its associated front prefix as key. It stores FP in its hash table.

Experiments prove that SNT generates fewer transitions when split position is chosen to be 2, 3 or 4. SNT technique is illustrated in Fig. 13.

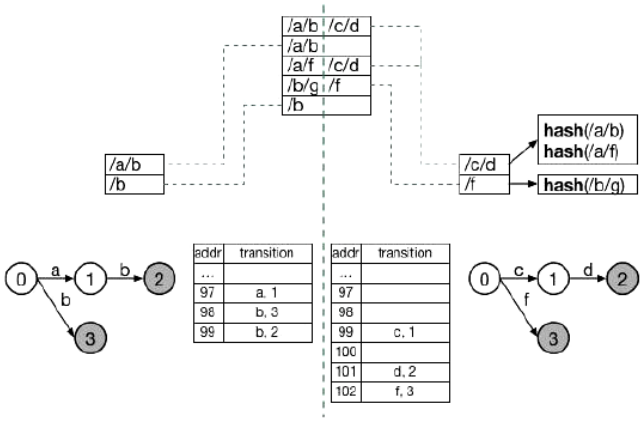


Fig. 13. An illustration of Split name trie.

I. Optimized Name trie using minASCII encoding

ChavooshGhasemi, et al. proposed Name trie using minASCII encoding which demonstrates the idea of *NameTrie*to overcome the node partitioningexisting in the trie [9]. All pointers of the trie are stored in hash table which eliminates the need of saving the pointers of leaf nodes. The control information of trie structure is encoded using ASCII characters. The lookup structure consists of minASCII name encoding and the *NameTrie* data structure. The minASCII codes are stored in *NameTrie* data structure.

The *NameTrie* design is depicted in Fig. 14.

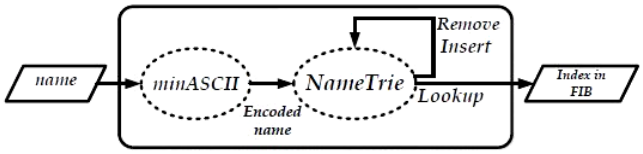


Fig. 14. *NameTrie*design.

In *NameTrie*, a dummy node is used to connect several tries starting with dissimilar characters. Every name is stored as an array of bytes. The set of names in *NameTrie* is shown in Fig. 15. The name lookup in the *NameTrie*is shown in Fig. 16.

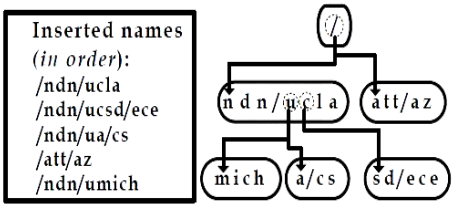


Fig. 15. NameTrie for a set of input names.

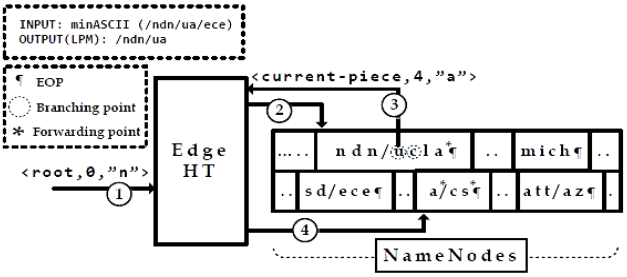


Fig. 16. Name lookup in *NameTrie.*

The characters of NDN name are stored contiguously in an array. Names are stored using Dynamic memory allocation. *NameTrie*edges connect a node's diverging point to the initial byte of a child node. Hash table,*EdgeHT* stores all edges of *NameTrie.* The keys of hash table made of *<parent node, offset, next byte>*. *Parent node* is the address of the parent node. The byte in the name piece at which the edge starts is indicated by the *offset.* The *next byte* of this edge is the beginning character of the child node. The return valueis the address of child node.NameTrie's memory usage outperforms NCE, HT, and NPT by an average of 40.9%, 84.9%, and 88.9%, respectively.

J. Mapping Bloom Filter (MBF)

MBF proposed by Zhuo Li, et al. is made up of 2 components- a Packet store and an Index table. Packet store is implemented on an off-chip memory. Index table is implemented as an on-chip memory in order to access the elements of the Packet store [10]. Proposed data structure is shown in Fig. 17 and Fig. 18.The index table is made up of 2 data structures- a Bloom filter (BF) and a Mapping array (MA). They are implemented as1-dimensional bit arrays. Bloom Filter aids to answer membership queries in the Mapping Bloom filter. In order to retrieve the Packet store the MA provides the offset address.BF array is divided into i equivalent divisions such thateach division of BF is mapped to 1 bit of i bits Mapping array. For every incoming interest request, every bit's initial state of MA is set to 0. While inserting the elements into MBF, based on k hash values some bits of Mapping array are set to 1. This is used as the offset address of the element in Packet store.

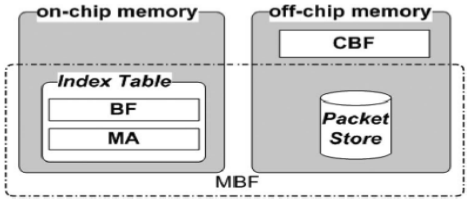


Fig. 17. MAPIT data structure

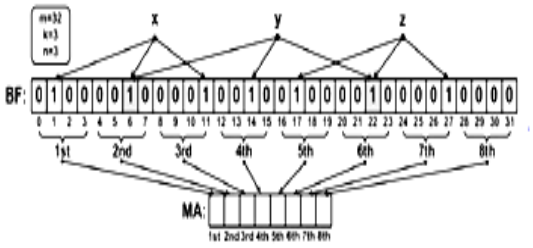


Fig. 18. Mapping Bloom Filter

IV. RESULTS AND DISCUSSION

Performance of lookup structure is measured in terms of lookup time and memory requirement for a given set of FIB name prefixes. Another performance criterion is false positive rate of Bloom filter for those structures employing Bloom filter. Experimental results obtained with different lookup structures are plotted in Fig. 19 to Fig. 28, [1] - [10]. Based on the simulation results, it is noted that BFAST achieves better speedup performance and reduced false positive rate compared to all other techniques, but at the cost of increased memory consumption.

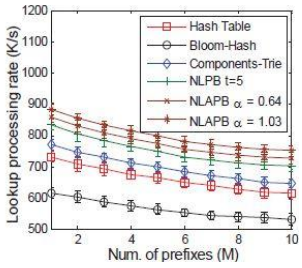


Fig. 19. NLAPB Lookup rate

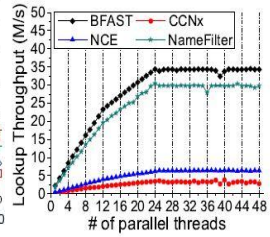
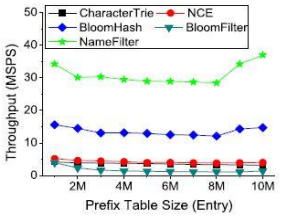


Fig. 20. BFAST Lookup rate



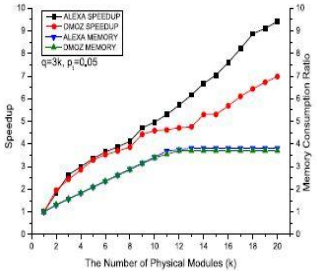


Fig 21. Name Filter ThroughputFig. 22. PNL Speedup rate

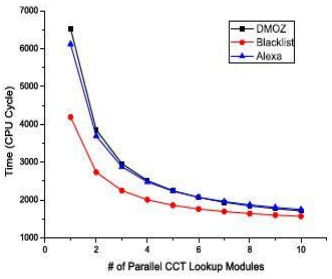


Fig. 23. NCE Lookup Time

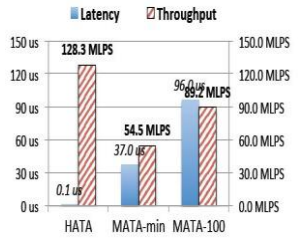


Fig. 24. FPGA Lookup throughput

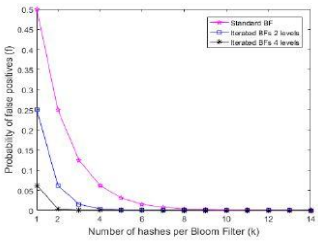


Fig. 25. IBF False positive rate

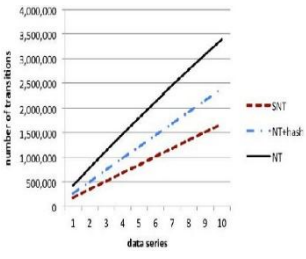


Fig. 26. SNT Transitions

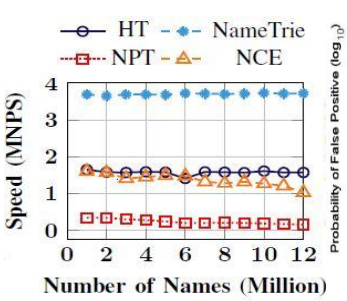


Fig. 27. NameTrie Lookup Speed

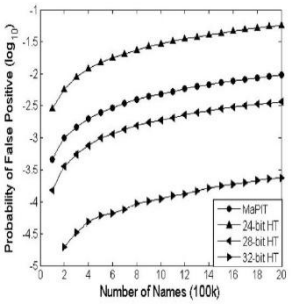


Fig. 28. MaPIT false positive rate

V. CONCLUSION

The implementation of memory efficient high-speed table lookup is the most essential feature for the optimized forwarding of the NDN packets. This faces several challenges due to large NDN tables and longer NDN names. Existing lookup solutions in implementing NDN table lookup are discussed and their performances are compared in this paper.

REFERENCES

1. Huichen Dai, Jianyuan Lu, Yi Wang, and Bin Liu, "BFAST:Unified and scalable index for NDN Forwarding Architecture," 2015 IEEE Conference on Computer Communications (INFOCOM).
2. Wei Quan, Changqiao, Jianfeng Guan, Hongke Zhang, and Luigi Alfredo Grieco, "Scalable Name lookup with Adaptive Prefix Bloom filter for Named Data Networking," IEEE COMMUNICATION LETTERS, VOL. 18, NO. 1, JANUARY 2014.
3. Yi Wang, Tian Pan, Zhian Mi, Huichen Dai, Xiaoyu Guo, Ting Zhang, Bin Liu, and Qunfeng Dong, "NameFilter: Achieving fast name lookup with low memory cost via applying two-stage Bloom filters," 2013 Proceedings IEEE INFOCOM.
4. Yi Wang, Huichen ai, Junchen Jiang, Keqiang He, Wei Meng, and Bin Liu, "Parallel Name Lookup for Named Data Networking," IEEE Globecom 2011 proceedings.
5. Yi Wang, Keqiang He, Huichen Dai, Wei Meng, Junchen Jiang, Bin Liu, and Yan Chen, "Scalable Name Lookup in NDN Using Effective Name Component Encoding," 2012 32nd IEEE International Conference on Distributed Computing Systems.
6. Yanbiao Li, Dafang Zhang, Xian Yu, Wei Liang, Jing Long, and Hong Qiao, "Accelerate NDN Name Lookup using FPGA: Challenges and a scalable Approach". Cristina Munoz, Liang Wang, Eduardo Solana and
7. Jon Crowcroft, "I(FIB)F: Iterated Bloom Filters for routing in Named Data Networks".
8. Yun Tan and Shuhua Zhu, "Efficient name lookup scheme based on hash and character trie in Named Data Networking," 2015 12th Web Information System and Application Conference.
9. ChavooshGhasemi, Hamed Yousefi, Kang G. Shin, and Beichuan Zhang, "A Fast and Memory-Efficient Trie structure for Name-based Packet Forwarding," 2018 IEEE 26th International Conference on Network Protocols.
10. Zhuo Li, Kaihua Liu, Yang Zhao, and Yongtao Ma, "MaPIT: An Enhanced Pending Interest Table forNDNwith Mapping Bloom Filter", IEEE COMMUNICATIONS LETTERS, VOL. 18, NO. 11, NOVEMBER 2014.
11. Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, CAIDA, Patrick Crowley, Christos Papadopoulos, Lan Wang, andBeichuan Zhang, "Named Data Networking", ACM SIGCOMM Computer Communication Review.
12. DivyaSaxenaa, VaskarRaychoudhurya, Neeraj Surib, Christian Beckerc, and JiannongCaod, "Named Data Networking: A survey", Computer Science review, 2016.

AUTHORS PROFILE

Swetha B is a research scholar at R.N.S Institute of technology, Bengaluru. She received M.Tech degree in VLSI and Embedded systems, in 2007 from K.L.E Engineering College under Visvesvaraya technological university. She received B.E degree in Electronics and Communication Engineering, in 2005 from BDT College under KuvempuUniversity.Her research area is Named data networks (NDN). She has a teaching experience of 10 years. Her subjects of interest include Computer networks, Operating systems, Data structures, and Computer architecture. She has presented and published paper titled "A new design methodology for distributed EDS applications using .NET remotingarchitecture" in national Conference on "Emerging trends in IT'. She is currently working on developing efficient algorithms for NDN table lookups.

Dr. S V Uma is working as Associate professor at R.N.S Institute of technology, Bengaluru. She received Ph.D. degree in the area "Congestion control and improved QOS in multimedia networks" from Bangalore University, in 2015. She received M.Tech in Digital Communication from BMSCE under Visvesvaraya technological University, in 2002. She received B.E degree in Electronics and Communication Engineering from Malnad College of engineering, in 1996. She has a teaching experience of 22 years. Her research interests include Communication Networks, Network security, Cryptography, Signal and Image Processing. She has authored book on "Constraint based network design using Genetic algorithm" under Lambert Academic publishing, Germany. She has authored and co-authored several technical publications in National Conferences, International Conferences and International Journals. She received "Best paper Award" for paper titled "A cross Layer UDP-IP protocol for efficient Congestion Control in wireless Networks", in International Journal of Computer Science and Information Security. She has guided more than 50 UG projects and more than 12 PG students for their final year projects. She has delivered invited talks on "Advanced Communication Technologies", "Basics of Engineering", etc. She is currently guiding 5 doctoral students in different areas of Computer Networks.