# A Review on Recent NDN FIB Implementations for High-Speed Switches

Eduardo Castilho Rosa and Flávio de Oliveira Silva

**Abstract** Forwarding Information Base (FIB) plays an essential role in Named-Data Networking (NDN) since it allows contents identified by unique hierarchical names to be reachable anywhere. Over the last few years, the advances in programmable switches have become possible to implement data structures for FIB in hardware to run at line rate. However, such implementations are not trivial in these devices, taking into account its architectural constraints and some NDN features like the complexity of dealing with variable-length names and the FIB size being orders of magnitude larger than the current IP routing tables. Despite all the benefits that high-speed switches may bring to NDN as a whole, the literature has been missing a survey that covers the data structures for FIB designed specifically to run in physical switches. To this end, we present a review on recent FIB implementations for both fixed-function and programmable high-speed switches. Our main contribution includes a fair and new comparative analysis among different approaches to implement the FIB highlighting its features and limitations. We also provide new insights and future research directions in this field.

## 1 Introduction

Information-Centric Networking (ICN) [1] is a clean-slate approach aiming to address the limitations of the current Internet. By using concepts like in-network caching, name-based forwarding, and data-centric security, ICN can better support emerging applications in Vehicular Ad Hoc Network (VANET), the Internet of Things (IoT), and 5G/6G.

Eduardo Castilho Rosa
Goiano Federal Institute, Catalão-GO, Brazil, e-mail: eduardo.rosa@ifgoiano.edu.br

Flávio de Oliveira Silva
Federal University of Uberlândia, Uberlândia-MG, Brazil, e-mail: flavio@ufu.br

Among several ICN-based architectures in literature, Named-Data Networking (NDN) [2] is currently the most mature. In contrast to IP, NDN forward packets by using hierarchical and variable-length names. The communication is based on the exchange of two types of packets: 1) Interest Packet (Ipckt) and 2) Data Packet (Dpckt). The former contains a unique name the consumer uses to retrieve the data. A Dpckt stored at any intermediate node is sent back to the consumer following the reverse path. The routing is performed through a data structure called Forwarding Information Base (FIB). FIB plays an essential role in NDN because it guarantees the reachability of any data content in the NDN domain. Such importance of FIB has been motivating the research community to adapt existing data structures like Trie, Bloom-Filter (BF), and Hash Tables (HT), to store name prefixes taking into account parameters such as lookup speed and memory footprint.

There are many FIB implementations in software such as NFD [3], NDN-DPDK [4], and YaNFD [5]. However, to scale up the NDN, it is necessary to move forward and implement its main data structures in hardware to enable forwarding packets at high speed. Recent developments in hardware and programmable switches make this possible, even though in most cases, native features in NDN such as caching and name-based forwarding do not fit naturally with the constraints of such devices like the limited on-chip SRAM/TCAM memory. Many solutions have been proposed recently to address these issues that use different techniques in several target architectures. However, although some surveys in the literature cover how NDN forwarding strategies are implemented in software [6, 7, 8], we still lack a comprehensive survey on specific FIB implementations for physical switches.

This work provides a review and a comparative analysis of recent FIB implementation for high-speed switches. Our goal is to answer the following questions: *What do we know about how to implement the FIB in both programmable and fixed-function switches? What is the most used method to do so?*, and *What are the gaps in this field?*. To answer these questions, the papers we reviewed are classified according to what switch technology the FIB is designed for and what method they used. We do not intend to provide a full systematic literature review. Instead, the idea is to present an overview of the most used methods to implement the FIB that is deployable in hardware and provide a fair comparison among relevant aspects of each paper, highlighting its main features and limitations.

The remain of this paper is organized as follows: In §2 we present a brief overview of the NDN architecture. In §3 we present 15 papers that focus on FIB implementations for both programmable and fixed-function switches. §4 brings a discussion and future research directions. Finally, we conclude the paper in §5.

## 2 NDN Overview

The NDN project was funded by the National Science Foundation (NSF) as part of the Future Internet Architecture Program. The central concept behind NDN is to replace the thin waist of today's Internet with a content-oriented network layer. In

contrast to IP, the core communication of the NDN layer-3 protocol is based on the asynchronous exchange of Ipckts and Dpckts. The NDN Routers deliver contents based on hierarchical variable-length names carrying on both Ipckt and Dpckt. To do so, NDN Routers implements three data structures: Content Store (CS), Pending Interest Table (PIT), and Forwarding Information Base (FIB).

Overall, CS plays an essential role in providing a Quality-of-Service (QoS) to applications because it can significantly alleviate the traffic in the core network, improving both latency and throughput. The CS is responsible for storing Dpckts in buffers to support in-network caching, one of the main features of an ICN-based architecture. In-network caching is used to ensure the independent-location property in NDN, seeing as not only the producer can serve the consumer with data but also any intermediate node storing a copy of it. The NDN specification foresees the need for CS to support a cache replacement policy to avoid overflowing the buffer capacity with too many Dpckts. The replacement policy uses the metadata information in Dpckt that includes the freshness time.

From a perspective of an incoming Ipckt, when the CS cannot find a Dpckt that matches exactly with the name in the Ipckt, the PIT table needs to be checked. The PIT stores all pending Ipckt that have been forwarded but not been satisfied yet, to avoid sending unnecessary information to the network. In other words, only the first Ipckt needs to be sent out to the network, and all the subsequent requests for the same data are aggregated in PIT together with the incoming port. Thus, similarly to CS, PIT can alleviate the traffic in the network.

FIB is the most important data structure in NDN because it is responsible for interconnecting consumers and producers worldwide. When a given Ipckt arrives at the NDN router for the first time, and both CS and PIT can not satisfy that request, FIB comes into play and forwards such Ipckt to one or more interfaces configured by a routing strategy. The FIB stores name prefixes and performs a Longest Name Prefix Matching (LNPM) whose definition is the same as the longest prefix matching of IP, except in FIB, the entries are variable length. Even though FIB can be implemented as HT [9] and BF [10], trie-based data structures are primarily used to reduce memory consumption in software, as we can see in [11, 12, 13]. Fig. 1 shows how CS, PIT, and FIB are connected.

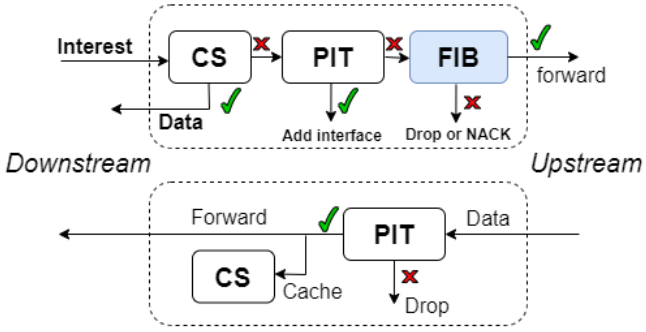


**Fig. 1** Operational flow in the NDN forwarding plane and its main data structures.

# 3 FIB Implementations in Hardware

The FIB implementations we present in this review are classified into two categories: 1) FIB designed for Fixed-Function Switches, and 2) FIB designed for Programmable Switches. In each category, we classify the papers according to their methods (HT, Trie, and BF) and present the publications in chronological order. Both fixed and programmable switches mean that the proposed mechanisms can be deployed either in traditional routers or in real programmable devices, not necessarily meaning they were implemented in such real devices. As we will see, some of them were implemented in software like BMv2 [14], others in FPGAs, and some of them the authors provide simulation analyses.

## *3.1 FIB for Fixed-Function Switches*

### 3.1.1 Hash Table

The first content router that supports name-based forwarding at high speed is Caesar [15]. Caesar distributes the FIB across multiple line cards and performs the LNPM accordingly. To optimize memory allocation, rather than duplicating the same forwarding table with $S$ entries at each line card, each line card stores a different subset of entries $S^{'}$ such that, summing up all of them, we get $S$. The downside of this approach is a possible increase in switching operations and complexity that may harm the performance. The authors suggest using CRC-64 as a hash function to store the prefixes into the FIB. A numerical evaluation of Caesar is performed by using the Xilinx Virtex-6 FPGA family as reference design. The main metric evaluated is the number of prefixes supported that depends on the number of line cards. A Caesar router with 600 line cards handles up to 600 million content prefixes.

The authors in [16] proposed a framework focusing on LNPM in NDN. Such a framework consists of two key components: a name prefix transformation and FIB instrumentation. When a router receives a name prefix announcement, it first applies the name reduction algorithm to transform the hierarchically structured name prefix into a compressed key. This transformation is achieved through hashing. For instance, if they have the NDN name /ufu/facom/mehar, they generate three keys by hashing /ufu, /ufu/facom, and /ufu/facom/mehar. As we can see, the redundant information (/ufu, /ufu/facom) is incorporated into the keys, which harms memory consumption. On the other hand, such an approach can easily be adapted to run in programmable switches. The evaluation was performed in both CPU and GPU and it used three name datasets. To measure the performance of the LNPM algorithm, the throughput was the only metric evaluated.

SACS [17] is another method that focuses on LNPM, like [16]. It consists of a shape and content search framework for TCAM and SRAM. In SACS, a TCAM-based shape search module is first used to determine a subset of possible matching name prefixes, and then an SRAM-based content search module is used on the sub-

set to find the longest matching prefix. The shape of a given prefix is a sequence of its lengths. For instance, the shape for the prefix /a/ab/abc is /1/2/3. These shapes are stored in TCAMs, pointing to hash tables located in SRAM. Compared to traditional hash-based methods, such an approach improves latency because many memory accesses are eliminated since the shape guides the search to a small subset of keys. However, SACS requires more memory than NFD [3], for instance, because it needs to maintain dual fingerprints and cells in a slot of hash tables. The experiments were conducted on a set of public name datasets and a public TCAM modeling tool [18] were used to test the performance of TCAM-based shape search. For the SRAM-based content search, the experiments were conducted on a server with Intel Xeon CPU E5-2640x2 and 94 GB of main memory. The main metrics evaluated were throughput and memory consumption.

### 3.1.2 Trie

Name Component Encoding (NCE) [19] is a mechanism to reduce memory consumption by assigning a unique code to a given name component in such a way that LNPM can be applied with the same semantics. For instance, the NDN names /ufu/facom/mehar and /facom/lsi can be converted to /1/2/3 and /2/4, respectively, by mapping a name component into an integer. The encoded name is then stored in the FIB implemented as a trie-based data structure. The LNMP is performed starting from the root to the full NDN name. The main limitation of NCE is the frequent access to a slow memory and the extra time to perform the encoding. Theoretical analysis and experiments conducted by using real dataset on a PC with an Intel Core 2 Duo CPU of 2.8 GHz and DDR2 SDRAM of 8 GB demonstrate that NCE can compress a FIB containing 3M entries to about 272 MB, 32.45% less than the baseline. Packet delay and average packet lookup time are also evaluated. For the same dataset, the former correspond to 1.9 us (7.7 faster than baseline) and the ladder 2,975.26 cycles. The benefits in terms of memory reduction and latency in NCE comes at the expense of a building time of 34s to encode the prefixes.

The second trie-based method proposed to fixed-function switches is [20]. Its main design goal is to have a compact data structure so that FIB with a few million entries can still fit in SRAM for fast lookup. Thus, the authors propose to use dual binary Patricia trie to minimize the redundant information stored. The binary representation provides more opportunities to compress shared parts between different prefixes. On the other hand, binary tries tend to increase the depth of the tree, impacting the lookup speed. Moreover, they introduce the idea of speculative forwarding, which uses the Longest-Prefix Classification (LPC) instead of the Longest Prefix Match (LPM). Unlike the LPM, LPC lookup guarantees that if there is a match, the packet will be forwarded to the same next hop that LPM would use but, if there is no match, the packet is still forwarded. However, for speculative forwarding, the imprecise forwarding of the LPC may cause forwarding loops. The evaluation is performed analytically and the metric used is the memory footprint. For a dataset

containing about 3.7M name prefixes, the memory footprint is around 31MiB, 50% less compared to tokenized Patricia trie.

#### 3.1.3 Bloom-filter

An Adaptive Prefix Bloom Filter (NLAPB) is presented in [21]. The key idea of NLAPB is to split NDN prefixes into two segments and conduct the lookup operation with a combination of CBF (Counting Bloom Filter) and trie. The first segment contains prefixes with *m* name components (B-prefix) stored in BF, whereas the second (T-suffix) contains variable-length prefixes stored in trie. NLAPB is implemented in a commodity router equipped with eight interfaces. The experiments illustrate that NLAPB achieves a fairly guaranteed scalability in terms of memory consumption when huge namespaces are considered. The limitation of NLAPB includes the possibility of false positives, which can impair the forwarding. Memory cost, lookup processing rate and false positive rate are the main metrics measured.

MaFIB [22] is another BF-based data structure for FIB. It uses a data structure called Mapping Bloom Filter (MBF), proposed in [23]. The MBF consists of a regular BF and a Mapping Array (MA), which are bit arrays in fast on-chip memory (SRAM). With the BF, the elements are verified whether they are in the MBF or not. The value in MA is utilized as the offset address to access the output face(s) stored in slow off-chip memory (DRAM). In comparison with [19] and other methods, MaFIB can provide a better compression ratio. However, its main limitations include frequent high access to DRAM to extract the output faces(s), and false positives may occur, impairing the accuracy of the forwarding. The performance of MaFIB is compared with the FIB based on NCE [19] in terms of the on-chip memory consumption, false positive probability, and building time.

B-MaFIB [24] is an improvement of MaFIB. It has the same features as MaFIB, but the authors changed MBF by using a bitmap-mapping bloom filter (B-MBF) index. The idea of B-MBF is to allow dynamic memory allocation to reduce memory consumption. However, B-MaFIB still needs to access slow DRAM to read the packet information, such as output port(s) and stale time. Besides the evaluated metrics in Ma-FIB [22], B-MaFIB also includes throughput.

### *3.2 FIB for Programmable Switches*

#### 3.2.1 Hash Table

NDN.p4 [25] is the first attempt to implement the NDN in the P4 language. They considered the original TLV name encoding to parse the NDN name content, although they concluded the P4 language restricts many operations on this kind of field. The FIB in NDN.p4 is implemented as a single P4 table with fixed-size entries. The number of table entries for a single FIB entry is proportional to the max-

imum number of components in processed packets. For instance, to store a prefix /ufu/facom/mehar, they calculate the hash of each part of the name, which is h(/ufu), h(/ufu/facom), and h(/ufu/facom/mehar), similarly to [16], and store all of them in the FIB. The LNPM is performed very quickly, in only one cycle, by using a ternary-matching operation. However, in a realistic scenario, this process can lead to inefficient memory use as the need to store $m$ entries in the FIB for only one name prefix, where $m$ is the maximum number of components. Also, the collision problem can increase memory consumption significantly.

To improve the NDN.p4, the authors in [26] designed an NDN router to address the scalability issues of the proposed FIB in NDN.p4 and to extend the NDN functionality, including the CS and multicast-capability. Similar to NDN.p4, the FIB in [26] is implemented using only one P4 table. To deal with the problem that names have variable length, they proposed a data structure called *hashtray*. The *hashtray* is constructed from a name of *max* components, and it is divided into *max* blocks, with each block $i$ containing the result of the hash of component $i$. In contrast to NDN.p4, such a technique requires a single table entry per FIB route, significantly improving memory usage. However, using a 16-bit hash function to construct the *hashtray* can cause lots of collisions. Furthermore, to store the *hashtrays* in the FIB properly, it is necessary padding it from the index $i$ up to the index *max*, where $i$ is the current number of name components in the name, and *max* is the maximum name components supported by the architecture.

A similar approach based on hashtrays is presented in [27]. However, they focus on the NDN routing mechanism primarily instead of forwarding. In addition to the limitations of a hashtray-based method [26], [27] stores the only single FIB table at the egress pipeline. As SRAM/TCAM are distributed equally across both ingress and egress pipelines, such an approach wastes half of the on-chip memory available in the switch.

The first attempt to implement the NDN router in a real programmable ASIC is [28]. The key idea is that Dpckts are forwarded by a switch ASIC alone, whereas an NDN engine forwards Ipckts at a server. The main limitation of such an approach is that FIB implemented in DRAMs can severely impact the latency. The data structure used to implement the FIB is HT. However, the authors do not provide any details about it. In terms of memory usage, the PIT is distributed across many different stages of pipeline to optimize TCAM and SRAM resources. However, there are a couple of open research issues such as re-designing the integrity maintenance method of PIT entries between pipelines, using the server as testers is not a good solution since the number of ports on a router is more than 100, and the degradation from the ideal throughput due to smaller Dpckt sizes.

### 3.2.2 Trie

On-Chip-FIB [29] is a FIB implemented as a Binary Search Tree (BST) in FPGA. Although FPGA is a P4-capable target [30], the OnChip-FIB was not designed to be P4 compatible. The content name is represented as a collection of strides of the

same size. For instance, the name /ufu/facom is represented as '/ufu', '/fac' and 'om', using stride size four. Those strides are inserted in the BST, and the LNPM is performed by traversing the BST starting from the root node down to a leaf node. In terms of memory consumption, the complexity of OnChip-FIB is $O(n)$, where $n$ is the number of names in the dataset. However, in the worst case, OnChip-FIB is $O(nk)$, where $k$ is the number of strides. When it comes to lookup speed, the time complexity of OnChip-FIB is $O(n)$, where $n$ is the name size.

A second trie-based implementation of the FIB is presented in ENDN [31]. In summary, ENDN extends the previous NDN data plane with a new P4 target architecture. The authors claim this novel architecture allows multiple isolated P4 forwarding functions to be defined and executed. They also extend the P4 language with several extern functions to enable the processing of strings. The FIB in ENDN (EFIB) is implemented as a data structure called FCTree [32]. FCTree uses trie and can compress name prefixes storing common sub-prefixes only one time. However, as FCTree is proposed to run in software, the authors do not explain how FCTtree is implemented in P4. ENDN uses a combination of trie and hashing because the authors add a function that takes a name component position in parameter and returns the hash of the component, which is used to query match tables and thus perform a specific P4 action based on the value of a content name component. The time complexity of LNPM, insertion, and deletion is $O(n)$, which is the main limitation of FCTree. Also, when it comes to implementing FCTree in P4, which the authors did not address, is how to use a self-balancing binary search tree taking into account the constraints that usually P4 targets impose.

### 3.2.3 Bloom-Filter

Finally, in [33] is proposed an FPGA-based FIB implementation for programmable switches. Just like [29], the mechanism in [33] is not P4-compliant. However, in contrast to [29], in which all FIB entries are stored in the same data structure (BST), in [33] the name prefixes in the FIB are divided into two distinct groups: 1) Group 1: Name prefixes with up to 4 components and 2) Group 2: Name prefixes with five or more components. The Group 1 name prefixes are stored in an external lookup table, i.e., in software, and the Group 2 name prefixes are stored in on-chip memory. Such division makes sense considering the limited amount of on-chip memory available in today's FPGA boards. The main data structure used is BF [34] combined with a hash function called $H_3$ to store the prefixes into the FIB on an FPGA. The main limitation of this approach is the high latency caused by the external name lookup. A proof of concept system is implemented on a virtex-7 FPGA where both memory consumption and packet processing rate are the metrics measured.

**Table 1** High-level comparison among FIB implementations for high-speed switches.

| Cat. | M | Ref. | Year | Key Features | Limitations | Target |
|---|---|---|---|---|---|---|
| F | HT | [15] | 2014 | •Distributes FIB across multiple line cards;<br>•FIB entries are stored by using CRC-64 hash; | •High #of switching operations, which impacts the latency; | FPGA |
| | | [16] | 2014 | •Focus on LNPM;<br>•Transform the name into a compressed key; | •Redudant information increases memory consumption; | GPU |
| | | [17] | 2018 | •Stores shape of FIB entry;<br>•Content search framework;<br>•Low #of memory access; | •Need to maintain dual fingerprint which increases memory usage; | TCAM |
| | Trie | [19] | 2012 | •Compress name prefixes;<br>•Component trie;<br>•Use State Transition Arrays; | •Frequent access to slow memory;<br>•Encoding process increases the latency; | CPU |
| | | [20] | 2015 | •Speculative forwarding;<br>•Longest-Prefix Classification; | •Forwarding loop may occur;<br>•Higher depth of the tree(>latency); | SRAM |
| | BF | [21] | 2014 | •Split NDN prefixes into two segments;<br>•Lookup based on popularity; | •False positives may occur;<br>•Dynamic memory allocation is hard to achieve in hardware; | Router |
| | | [22] | 2017 | •Low on-chip memory cost;<br>•Support update operations;<br>•Stores large name datasets;<br>•Low building time; | •Frequent access to DRAM;<br>•High memory usage due to CBFs in off-chip memory;<br>•Lots of hash functions executed in the dataplane; | SRAM DRAM |
| | | [24] | 2018 | •Idem [22];<br>•Leverage only one hash function in the data plane;<br>•Dynamic memory allocation;<br>•Use of Bitmap as BF; | •Frequent access to DRAM to access output faces increase latency; | SRAM DRAM |
| P | HT | [25] | 2016 | •FIB implemented as a single P4 table;<br>•#table entries are proportional to #name components;<br>•LNPM executed at line rate; | •Waste of memory because many tables entries are inserted for each NDN name prefix;<br>•Not scale to millions of prefixes; | BMv2 |
| | | [26] | 2018 | •FIB implemented as a single P4 table;<br>•FIB can scale better;<br>•Use of hashtrays in TCAMs; | •Use of 16-bit hash functions can cause lots of collisions, requiring more scarce TCAMs | BMv2 |
| | | [27] | 2021 | •Idem [26];<br>•Focus on routing; | •Idem [26];<br>•A large P4 table is used at egress wasting TCAM/SRAM at ingress; | BMv2 |
| | | [28] | 2021 | •FIB is stored in DRAM;<br>•Optimize memory usage by splitting the PIT into stages; | •LNPM in DRAM adds to latency; | ASIC |
| | Trie | [29] | 2019 | •Content name is represented as strides;<br>•Exploits the massive parallel processing of FPGAs; | •Worst-case memory complexity is $O(nk)$;<br>•Transverse the trie is time consuming; | FPGA |
| | | [31] | 2020 | •Extern to process strings;<br>•Support of wildcards searches;<br>•Compress prefixes whose share common sub-prefixes; | •NLPM, insertion and deletion is $O(n)$ impacting the latency;<br>•Supporting self-balance trees is costly in P4 targets; | BMv2 |
| | BF | [33] | 2019 | •Prefixes are splitted into 2 groups; | •External lookup increases latency; | FPGA |

(Cat=Category; F=Fixed-Function Switches; P=Programmable Switches; M=Method; HT=Hash Table; BF=Bloom-Filter)

## 4 Discussion and Future Directions

Table 1 shows a preference for implementing the FIB by using HT weather in fixed-function or programmable switches. The main reason is probably the simplicity of calculating hashing in such devices since there are several dedicated chips to do so at line rate. Also, hashing optimizes the latency, which fits well with high-speed switches. However, when it comes to FIB in NDN, hashing increases the memory footprint due to the redundancies in the name structure and the space necessary to deal with collisions [35].

On the other hand, BF is also a compelling data structure but is less used. False positives and large memory to deal with it could explain that. Considering that the most memory-efficient methods reported in this review are MaFIB [22] and B-MaFIB [24], both using BF as its primary data structure, such techniques should be better explored in future research.

Trie-based methods are widely adopted in software implementations of the FIB [6] but, as Table 1 shows, when it comes to FIB implemented in hardware, there are just a few in literature. All of them were implemented in FPGAs and software switches like BMv2, which are more flexible but less restrictive than ASICs. The main reason for that resides in the limitations in hardware such as no-loops, no pointers, and no dynamic memory allocation that all trie-based methods rely upon.

Since HT, tries, and BF are the methods used in all the papers we reviewed and also since such methods are widely used in software as well [6], one possible research direction is to come up with solutions to implement such data structures in hardware overcoming its limitations and constraints. Given the fact that some programmable ASICs like Tofino have been open-source recently [36] and the open P4 specification allows the definition of new target architectures, researchers can even develop new hardware design in the future to better support the implementation of the data structures aforementioned.

Finally, from all papers we reviewed, we can also observe that [28] was the only one that provided a prototype of an NDN router in an actual programmable switch (Tofino ASIC). However, only the PIT data structure is fully implemented in the hardware. Both CS and FIB are stored externally in DRAMs, which may impact the performance negatively. Given the crucial role that FIB plays in NDN, this gap needs to be filled. Therefore, efficient implementations of FIB and CS in the ASIC besides HT, tries, and BF is a worthy topic of research with excellent opportunities to innovate, mainly because P4 language is a trend now, and NDN has been gaining momentum over the last few years due to its standardization process at NIST [37]. Since on-chip memory in today's programmable ASICs is minimal but, at the same time, the scalability of NDN depends on processing packets at Tbps, the study of new compression techniques that takes into account both the flexibility of current programmable pipelines and the hardware constraints is a good direction towards filling this gap. Such an approach goes beyond the scope of NDN and can benefit all fields in computing that use cache-based systems implemented in hardware that makes use of string processing. Examples of such use-cases may include Direc-

tory Caches in Operating Systems (DCache), Content Delivery Networks (CDN), Domain Name System (DNS), and many others.

## 5 Conclusion

Forwarding Information Base (FIB) is one crucial data structure in Named-Data Networking (NDN) that aims to provide connectivity between consumers and producers. This work reviewed some recent FIB implementations for both fixed and programmable switches. Our observation is that implementing FIB data structure in hardware is quite challenging due to matching NDN rules to switch constraints and limitations. A total of 15 reviewed papers have shown that Hash Table is the most used method to implement the FIB in hardware, followed by Trie and Bloom-Filter. Such data structures are compelling, but, at the same time, they were not designed for the specific purpose of FIB.

Therefore, we argue that the research community needs to fill this gap and come up with new data structures for FIB that are aware of the architecture constraints while providing a good balance between memory footprint and lookup speed.

## References

1. B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, 2012.
2. L. Zhang, D. Estrin, and J. Burke, "Named data networking (ndn) project," University of California, Los Angeles, CA, NDN Project Technical Report NDN-0001, October 2010.
3. A. Afanasyev and J. Shi, "NFD Overview — Named Data Networking Forwarding Daemon (NFD) 0.6.6-24-g1402fa1 documentation," 2019. [Online]. Available: http://named-data.net/doc/NFD/current/overview.html
4. J. Shi, D. Pesavento, and L. Benmohamed, "Ndn-dpdk: Ndn forwarding at 100 gbps on commodity hardware," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, ser. ICN '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 30–40. [Online]. Available: https://doi.org/10.1145/3405656.3418715
5. E. Newberry, X. Ma, and L. Zhang, *YaNFD: Yet Another Named Data Networking Forwarding Daemon*. New York, NY, USA: Association for Computing Machinery, 2021, p. 30–41. [Online]. Available: https://doi.org/10.1145/3460417.3482969
6. Z. Li, Y. Xu, B. Zhang, L. Yan, and K. Liu, "Packet forwarding in named data networking requirements and survey of solutions," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1950–1987, 2019.
7. A. Tariq, R. A. Rehman, and B.-S. Kim, "Forwarding strategies in ndn-based wireless networks: A survey," *IEEE Communications Surveys Tutorials*, vol. 22, no. 1, pp. 68–95, 2020.
8. A.-q. Majed, X. Wang, and B. Yi, "Name lookup in named data networking: A review," *Information*, vol. 10, no. 3, 2019. [Online]. Available: https://www.mdpi.com/2078-2489/10/3/85
9. R. Shubbar and M. Ahmadi, "Efficient name matching based on a fast two-dimensional filter in named data networking," *International Journal of Parallel, Emergent and Distributed Systems*, vol. 34, no. 2, pp. 203–221, 2019.

10. C. Muñoz, L. Wang, E. Solana, and J. Crowcroft, "I(fib)f: Iterated bloom filters for routing in named data networks," in *2017 International Conference on Networked Systems (NetSys)*, 2017, pp. 1–8.
11. D. Saxena and V. Raychoudhury, "N-fib: Scalable, memory efficient name-based forwarding," *Journal of Network and Computer Applications*, vol. 76, pp. 101–109, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804516302132
12. C. Ghasemi, H. Yousefi, K. G. Shin, and B. Zhang, "A fast and memory-efficient trie structure for name-based packet forwarding," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018, pp. 302–312.
13. J. Seo and H. Lim, "Bitmap-based priority-npt for packet forwarding at named data network," *Computer Communications*, vol. 130, pp. 101–112, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366417303298
14. A. Bas, A. Fingerhut, and A. Sivaraman, "The behavioral model." May 2021, original-date: 2015-01-26T21:43:23Z. [Online]. Available: https://github.com/p4lang/behavioral-model
15. D. Perino, M. Varvello, L. Linguaglossa, R. Laufer, and R. Boislaigue, "Caesar: A content router for high-speed forwarding on content names," in *2014 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, 2014, pp. 137–147.
16. F. Li, F. Chen, J. Wu, and H. Xie, "Longest prefix lookup in named data networking: How fast can it be?" in *2014 9th IEEE International Conference on Networking, Architecture, and Storage*, 2014, pp. 186–190.
17. K. Huang and Z. Wang, "A hybrid approach to scalable name prefix lookup," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*, 2018, pp. 1–10.
18. B. Agrawal and T. Sherwood, "Ternary cam power and delay model: Extensions and uses," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 5, pp. 554–564, 2008.
19. Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen, "Scalable name lookup in ndn using effective name component encoding," in *2012 IEEE 32nd International Conference on Distributed Computing Systems*, 2012, pp. 688–697.
20. T. Song, H. Yuan, P. Crowley, and B. Zhang, "Scalable name-based packet forwarding: From millions to billions," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*, ser. ACM-ICN '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 19–28. [Online]. Available: https://doi.org/10.1145/2810156.2810166
21. W. Quan, C. Xu, J. Guan, H. Zhang, and L. A. Grieco, "Scalable name lookup with adaptive prefix bloom filter for named data networking," *IEEE Communications Letters*, vol. 18, no. 1, pp. 102–105, 2014.
22. Z. Li, K. Liu, D. Liu, H. Shi, and Y. Chen, "Hybrid wireless networks with FIB-based Named Data Networking," *EURASIP Journal on Wireless Communications and Networking*, vol. 2017, no. 1, p. 54, Mar. 2017. [Online]. Available: https://doi.org/10.1186/s13638-017-0836-0
23. Z. Li, K. Liu, Y. Zhao, and Y. Ma, "Mapit: An enhanced pending interest table for ndn with mapping bloom filter," *IEEE Communications Letters*, vol. 18, no. 11, pp. 1915–1918, 2014.
24. Z. Li, Y. Xu, K. Liu, X. Wang, and D. Liu, "5g with b-mafib based named data networking," *IEEE Access*, vol. 6, pp. 30 501–30 507, 2018.
25. S. Signorello, R. State, J. François, and O. Festor, "Ndn.p4: Programming information-centric data-planes," in *2016 IEEE NetSoft Conference and Workshops (NetSoft)*, 2016, pp. 384–389.
26. R. Miguel, S. Signorello, and F. M. V. Ramos, "Named data networking with programmable switches," in *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, 2018, pp. 400–405.
27. X. Guo, N. Liu, X. Hou, S. Gao, and H. Zhou, "An efficient ndn routing mechanism design in p4 environment," in *2021 2nd Information Communication Technologies Conference (ICTC)*, 2021, pp. 28–33.
28. J. Takemasa, Y. Koizumi, and T. Hasegawa, *Vision: Toward 10 Tbps NDN Forwarding with Billion Prefixes by Programmable Switches*. New York, NY, USA: Association for Computing Machinery, 2021, p. 13–19. [Online]. Available: https://doi.org/10.1145/3460417.3482973

29. D. Saxena, S. Mahar, V. Raychoudhury, and J. Cao, "Scalable, high-speed on-chip-based ndn name forwarding using fpga," in *Proceedings of the 20th International Conference on Distributed Computing and Networking*, ser. ICDCN '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 81–89. [Online]. Available: https://doi-org.ez34.periodicos.capes.gov.br/10.1145/3288599.3288613
30. H. Wang, R. Soulé, H. T. Dang, K. S. Lee, V. Shrivastav, N. Foster, and H. Weatherspoon, "P4FPGA: A Rapid Prototyping Framework for P4," in *Proceedings of the Symposium on SDN Research*. Santa Clara CA USA: ACM, Apr. 2017, pp. 122–135. [Online]. Available: https://dl.acm.org/doi/10.1145/3050220.3050234
31. O. Karrakchou, N. Samaan, and A. Karmouch, "Endn: An enhanced ndn architecture with a p4-programmabie data plane," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, ser. ICN '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–11. [Online]. Available: https://doi.org/10.1145/3405656.3418720
32. ——, "Fctrees: A front-coded family of compressed tree-based fib structures for ndn routers," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 1167–1180, 2020.
33. W. Yu and D. Pao, "Hardware accelerator for FIB lookup in named data networking," *Microprocessors and Microsystems*, vol. 71, Nov. 2019, publisher: Elsevier BV.
34. L. Luo, D. Guo, R. T. B. Ma, O. Rottenstreich, and X. Luo, "Optimizing bloom filter: Challenges, solutions, and comparisons," *IEEE Communications Surveys Tutorials*, vol. 21, no. 2, pp. 1912–1949, 2019.
35. E. C. Rosa and F. d. O. Silva, "A hash-free method for fib and lnpm in icn programmable data planes," in *2022 International Conference on Information Networking (ICOIN)*, 2022, pp. 186–191.
36. "Open Tofino," Dec. 2021, original-date: 2020-10-14T22:45:36Z. [Online]. Available: https://github.com/barefootnetworks/Open-Tofino
37. "NDN Community Meeting," Jun. 2020, last Modified: 2020-09-23T10:36-04:00. [Online]. Available: https://www.nist.gov/news-events/events/2020/09/ndn-community-meeting