information

MDPI

Review

Name Lookup in Named Data Networking: A Review

Al-qutwani Majed, Xingwei Wang * and Bo Yi

College of Software, Northeastern University, Shenyang 110819, China; gatwani@yahoo.com (A.-q.M.); yibobooscar@gmail.com (B.Y.)

* Correspondence: wangxw@mail.neu.edu.cn

Received: 22 November 2018; Accepted: 2 February 2019; Published: 26 February 2019

Abstract: Named data networking (NDN) is an alternative model to the current traditional IP-based Internet by improving content distribution in a network with content name. In NDN, content is sent/received based on its name rather than its address. Forwarding is still considered the bottleneck of NDN despite the effort in naming, routing, mobility, and security. NDN requires per-packet update that is preceded by a one-name lookup operation, which results in heavy time processing. The content names consume a large amount of memory, which causes the fast growth of NDN tables. To review the current solutions for NDN name lookup, this paper explores the existing NDN approaches and architectures by using a novel classification method. Furthermore, name lookup approaches were analyzed and compared in a novel manner. This survey highlighted the issues that affect name lookup performance in NDN, such as scalability, memory consumption, time efficiency, latency, and validity. We pointed out directions for future work and open issues to improve NDN name lookup in minimizing implementation cost and enhancing performance.

Keywords: named data networking; name lookup; NDN

1. Introduction

Named data networking (NDN) [1] is a promising networking paradigm that pays considerable attention to Internet networking. NDN concentrates on data rather than their sources. It is data-based, which radically differs from IP-based networking in many aspects [2]. NDN forwarding is a stateful network, whereas IP forwarding is stateless. NDN supports in-network caching, whereas IP does not. However, the major difference between them is that NDN forwarding is performed based on the content name rather than content address. The consumer requests the desired data, which are identified by a variable-length name, by sending an interest packet. The intermediate NDN router stores the interest name and its incoming interface. When the desired data arrive at the NDN router in the form of data packet, the router looks up its tables to find the corresponding interface or drop the data packet if a matched entry is not found [3]. An NDN router is composed of the following components: (1) content store (CS), which is a cache memory to store data in response to forthcoming interests from new requesters; (2) pending interest table (PIT), which is a table that is used to store interest packets temporarily until they are satisfied; and (3) forwarding interest base (FIB) to forward interest packets to the next routers by the longest prefix matching (LPM).

Name lookup is a bottleneck in NDN forwarding. Unlike IP address, the NDN name is unbounded with a variable length such that the router may search for up to tens or hundreds of characters before finding the required name. Millions or even billions of names may be stored in the NDN router, and the NDN name table can be larger by several orders compared with the IP table. Interest packet forwarding could perform three name lookup operations for CS, PIT, and FIB, whereas data forwarding could execute two name lookup operations for PIT and CS. In addition to name lookup, the NDN router performs other operations, such as updating, inserting, or deleting an element to interest or

data packet. Another challenge is that name lookup is conducted based on two methods, namely, exact matching for CS and PIT and LPM for FIB [2]. Thus, NDN name lookup consumes much time and memory is more difficult than IP-based Internet.

In this paper, we summarize and review papers that focus on the name lookup issue. To the best of our knowledge, ours is the first paper to survey various approaches for NDN name lookup. We cover three basic data structures that are used in NDN, namely, tries, hash tables (HT), and bloom filters (BF). The classification process is conducted based on these data structures. Certain approaches combine more than one data structure or are even implemented on additional hardware equipment. Specifically, we make the following contributions to the literature:

- We present an overview of NDN naming and name lookup challenges.
- We analyze and compare the currently proposed methods in NDN name lookup.
- To the best of our knowledge, this is the first paper that collects and compares the studies published in NDN name lookup.

The remainder of the paper is organized as follows: Section 2 presents an overview of NDN architecture and its naming. Section 3 introduces the existing solutions for the NDN name lookup problem. Section 4 presents an analysis of the current solutions and performance comparison. Section 5 recommends directions for future work in NDN name lookup. Section 6 concludes this survey after discussing the current solutions and their performance.

2. NDN Review

In this section, we describe the nature of NDN and its characteristics. We also present the components of NDN in addition to their functionalities. Furthermore, this section describes the challenges of NDN name lookup.

2.1. NDN Architecture

NDN is a novel networking paradigm that was introduced by [1] as an instance of information-centric networking (ICN) [4]. NDN is also known as a name-based network because packets are identified and forwarded according to the name of the data, whereas packets in an IP-based network are forwarded based on packet address or port number. This notion is the significant difference between NDN and IP-based networks [5]. Two types of packets are used in NDN, namely, interest and data. The content name is added as a string to both types of packets [6]. The interest packet is issued for the content requested by the user, whereas the data packet is the corresponding data content as requested [7]. When an interest packet arrives at a router that contains the requested content, its corresponding data packet is sent by retracing the reverse path of the interest packet [6]. The content producer adds certain contents into the network. When receiving these contents, routers extract the name prefixes of contents to construct their forwarding data bases FIBs [7]. Packets are forwarded based on hop-by-hop method.

Figure 1 [1] depicts the NDN router, which has the following data structures [6,7]:

- FIB. It stores information about the interfaces of interest packet and forward it upstream to the next hop by using LPM. It maintains name prefixes and corresponding interfaces to the producer or provider, which may have the requested content.
- PIT. It maintains interest packets until either they are satisfied or their lifetime is expired. It uses the exact matching (EM) method for searching PIT entries and can forward multiple interest packets. When a specific content has multiple interest packets, the first interest packet is forwarded, while all others are pending in PIT and waiting for the corresponding data packet.
- CS. Its basic functionality is to optimize content retrieval time, delivery latency and save bandwidth. It is used as a temporary cache for data packets because several users may request the same content consequently, e.g., several users may watch the same video. CS also uses the EM method for searching CS entries.

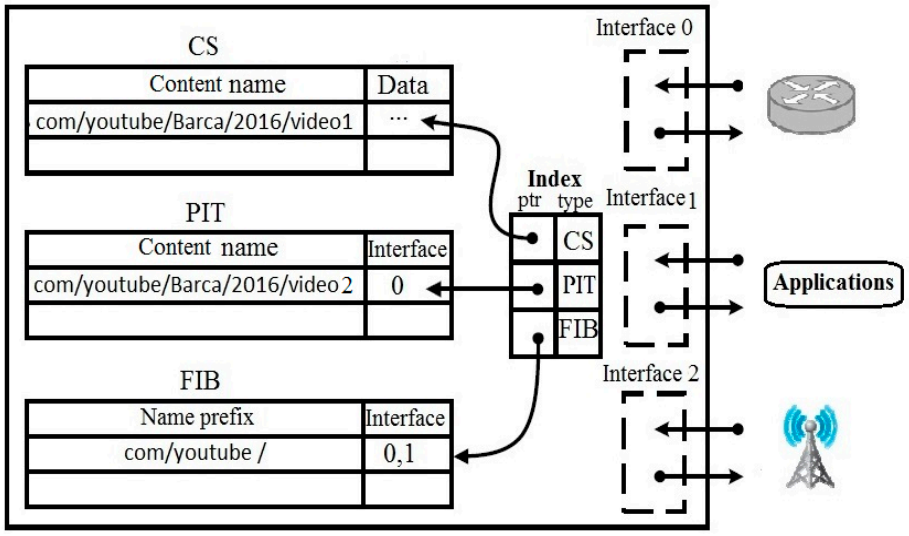

Figure 1. NDN router and its components [1].

Once an interest packet arrives at an interface of the NDN router, as shown in Figure 2 [8], the following procedure is executed:

(1) The router consults its CS if the desired data packet is already cached. If a matching entry is available, then the data packet is sent following the reverse path of the interest packet.
(2) If the CS has no available entry for the desired data packet, then the router searches the PIT to find an entry for corresponding interest packet. If a corresponding entry is available, then the given entry is aggregated in the list of incoming interfaces in the PIT.
(3) If no entry is matching in the PIT, then the router inserts a new entry for the interest packet to the PIT and searches against the FIB based on LPM to select the next hop for the given interest packet and forwards the interest upstream.

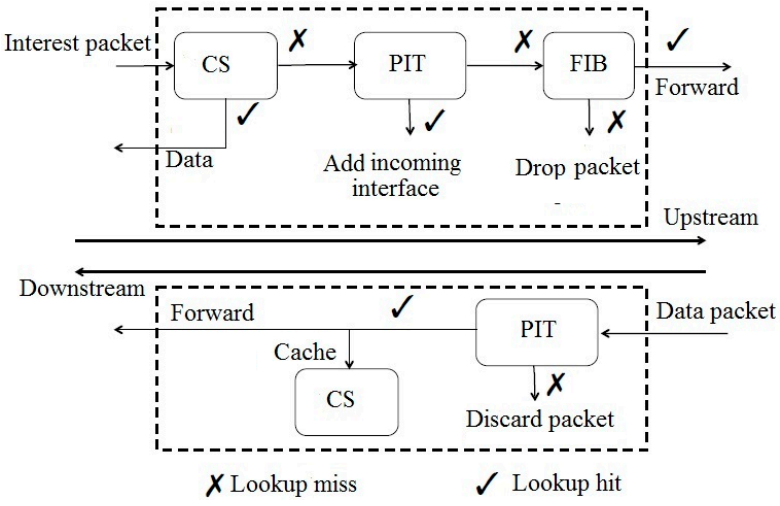

Figure 2. Forwarding process at NDN router [8].

When a data packet arrives at the NDN router, the router executes the following procedure:

(1) The router searches all PIT entries. If a matching entry exists, then the router forwards the data packet to the interfaces that are linked with the PIT and deletes its corresponding entry in the PIT. The data packet can be cached according to the router's caching policy, which may lead to the update of the CS.

(2) If no matching entry exists in the PIT (because of lifetime or other reasons), then the data packet is dropped [9].

2.2. NDN Name Lookup

2.2.1. Naming in NDN

Each packet in NDN is identified by a globally unique name. The packet name is opaque to the network and dependent on application. The application can implement a naming method based on its requirement because it is independent of the network [6]. An NDN name can use a hierarchical structure that is composed of components separated by a slash ('/') or dot ('.'). The hierarchical structure enables a name to be aggregated and positively affects the name lookup speed and NDN scalability. An example of a NDN name composed of five components is com/youtube/Barca/2016/video1, where com/youtube/ is the domain name, and Barcelona/2016/video1 is the content's directory path [10].

2.2.2. Packet Lookup

As previously stated, lookup in CS and PIT is based on EM, whereas that in the FIB is based on LPM. Two EMs and one LPM in NDN complicate name lookup, which creates a concern for packet forwarding. To employ high-speed packet forwarding, NDN name lookup should take the following issues into account.

(1) High-speed lookup. A new technology has been presented with wire speed lookup of 160 GBPS (gigabits per second). To execute NDN forwarding in a large-scale router, name lookup has to maintain the wire speed. At the same time, the NDN router has to reduce the packet forwarding latency to less than 100 μs (microseconds) [11]. Thus, sending packets with low latency while maintaining high-speed lookup is challenging [12].
(2) Memory storage. An NDN router may contain tens of millions or even billions of names because each name may consist of tens or hundreds of characters and name length is variable. Thus, an NDN router may need tens of gigabytes of memory to perform name lookup in a large-scale router.
(3) Fast update. An NDN router should process at least one new type of update when the content is inserted to or deleted from the NDN router. The corresponding name entry should be added to or deleted from the router, which renders the update operation recurrently. Therefore, the update operations in NDN router make the fast update issue more critical to address as well as affect the overall performance.
(4) Scalability. As the number of contents increases, the table size of the NDN router grows fast. As a result, scalability is a considerable challenge.

Interest packet forwarding may include many operations [4], that is, a lookup in CS is required when an interest packet arrives. If it fails to be hit, then a lookup and addition are processed in the PIT. Then, a lookup is needed in FIB using LPM until a match is found. When a data packet arrives at the NDN router, it requires a CS lookup, PIT lookup and deletion, and CS deletion and addition.

3. NDN Name Lookup and Current Solutions

In the proposed NDN, the interest and data packets are looked up in the router data structures for forwarding. At least there is a need to perform one name lookup operation for each packet arrives at the router. These critical challenges and the nature of the variable and unbounded names motivated the researchers to introduce many approaches for solving this issue. In this section, we introduce an extensive survey of the NDN name lookup approaches that have been proposed in recent years.

3.1. Name Lookup Classification

The proposed solutions are classified in this paper according to the components of the NDN router and type of data structures used. These solutions are classified into FIB name lookup, PIT name lookup, and general solutions for the entire router. The entire solutions are those that are performed in parallel for FIB, PIT, and CS. The proposed methods are classified again based on the data structure used, as shown in Table 1. To the best of our knowledge, no paper studied the CS name lookup because the CS principle is the same as PIT. Therefore, we selected these solutions because they are considered the main approaches for the name lookup issue. The classification method considers the field of approach and its performance results. We also covered papers in relation to content centric networking CCN, where NDN is another name for CCN. For facilitation, Table 2 presents the terminologies abbreviations we have used in this review. The period of the surveyed papers spans from 2010 to 2017.

Table 1. Classification of NDN name lookup approaches.

FIB	PIT	Entire NDN Router
Trie-based	Trie-based	HT-based
HT-based	HT-based	BF-based
BF-based	BF-based	Hybrid approaches
Hybrid approaches	Hybrid approaches	

Table 2. Abbreviations of the review's terminologies.

Terminology	Abbreviation	Terminology	Abbreviation
Named data networking	NDN	Hash table	HT
Information-centric networking	ICN	Bloom filter	BF
Content centric networking	CCN	Name component trie	NCT
Longest prefix matching	LPM	Encoded name component trie	ENCT
Exact matching	EM	Leaf prefix count table	LPCT
Content store	CS	Name lookup with adaptive prefix bloom filter	NLAPB
Pending interest table	PIT	Split name-character trie	SNT
Forwarding information base	FIB	Exact string differentiation	ESD
Parallel name lookup	PNL	Fingerprint-based Patricia trie	FPT
Million lookup per second	MLPS	Component radix trie	CRT
Name prefix trie	NPT	Name radix trie	NRT
Named component encoding	NCE	Linear chaining hash-table	LHT
Encoded name prefix trie	ENPT	D-left hash table	DHT
Graphics processing unit	GPU	Distributed PIT	DiPIT
State transition array	STA	United bloom filter	UBF
Field-programmable gate arrays	FPGA	Name prefix hash table	NPHT
Aligned transition arrays	ATA	Cyclic redundancy check	CRC-32
Multiple aligned transition array	MATA	Deterministic finite automata	DFA
Hierarchical aligned transition array	HATA	Steerable name lookup based on bloom filter	SNLBF
Longest prefix classification	LPC		

3.2. Current Solutions

3.2.1. FIB Schemes

An FIB entry consists of a name prefix and its corresponding ports. The NDN name lookup is performed using LPM. For instance, if one packet with the content name/com/google/news/china matches the second and third entries in FIB, then the packet is forwarded to all ports that contain the third entry based on LPM [11]. This section covers various approaches that have been introduced for name lookup in FIB.

In the proposed NDN, Zhang et al. [1] suggested the base solution for name lookup. The proposed approach is based on ternary content addressable memory (TCAM). In this approach, when packets arrive at the NDN router, names are loaded to TCAM directly. Although TCAM enables fast lookup, it results in considerable waste of memory and causes excessive power consumption. In addition,

this approach is applied only for small FIB entries with short names. Therefore, exploiting TCAM is an inappropriate solution for storing NDN names in large-scale tables due to its high power consumption, high price, and high update cost.

- **Trie-Based Approach**

Wang et al. [12] presented parallel name lookup (PNL) as the first solution in NDN name lookup for FIB forwarding. The names were represented by a trie with component granularity, namely, name prefix trie (NPT). Figure 3 shows that many NPT nodes can be collected and assigned to parallel memories. When FIB receives a packet, it checks whether the first component matches one of the root edges. If so, then the lookup node transits from the root to the child node, and the subsequent lookup process is performed repetitively. In certain memories, the content name is matched such that when many input names arrive concomitantly; they visit all memories and work concurrently, thus achieving high-speed lookup. However, several nodes maybe duplicated during the allocation process and cause explosion in memory because the memory is dependent on the access probability of each node.

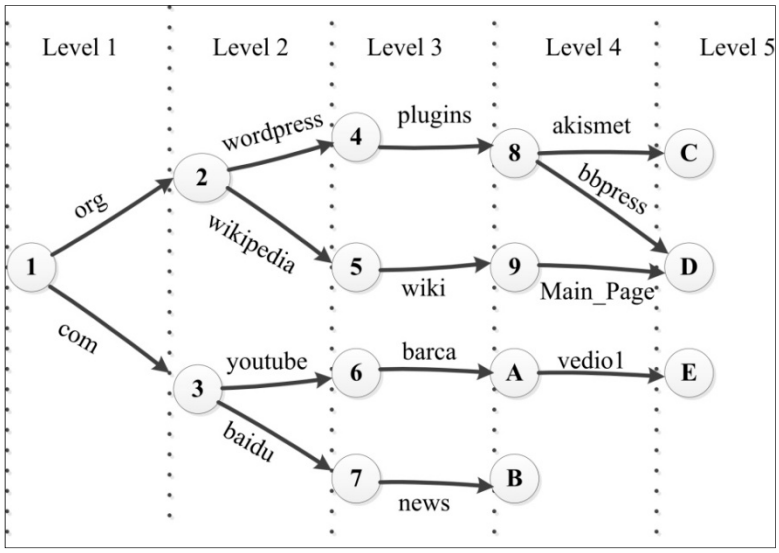

Figure 3. An Example of name prefix trie (NPT).

Yi Wang et al. [5] introduced named component encoding (NCE), using a 32-bit code (an integer) to encode name components. To implement this method, they proposed two techniques, namely, a code allocation method to reduce memory size for name components and enhanced state transition arrays (STA) to speed up the longest prefix lookup. After encoding the name components, the produced codes were assigned to the corresponding edges in the name prefix trie (NPT), which produced a new trie called encoded name prefix trie (ENPT). Compared with NPT, which is considered NCT, NCE needs only 18% of the memory size of NCT. In the worst case, it can process 1.3 MLPS (million lookups per second) and more than 20,000 name updates per second.

Yi Wang et al. [11] exploited the massive processing power of a graphics processing unit (GPU) to present the first wire speed engine based on a hardware accelerator. The lookup latency that resulted from the massive power of GPU is solved by utilizing a multi-stream mechanism of the GPU architecture. The specific GPU-based scheme works as follows: First, the name table is arranged in a character trie. Afterwards, the character trie is transformed into a multiple aligned transition array (MATA). MATA is transferred to the GPU-based lookup engine. The GPU accepts the input name and performs a name lookup using MATA. Once a name lookup is processed, the result of the lookup in GPU is transferred to the CPU to acquire the information of the next-hop interface. Experimental evaluations show that this approach handles 63.52 MLPS and achieves more than 30 K and 600 K per second for insertions and deletions, respectively.

Instead of GPU, Li et al. [13] used field-programmable gate arrays (FPGA) for the first time as a solution for NDN name lookup. Processing name lookup using FPGA is challenging. The potential collisions between transitions could happen in one level of a trie. To address the nature of names in NDN, a novel hierarchical aligned transition array (HATA) is proposed to represent the name trie. With the use of the multi-candidates and parallel validation method, the transitions stored in the aligned transition arrays (ATAs) are selected, where at most one transition is suitable. The validation stage can be processed in parallel because the ATAs are stored in different RAMs. Therefore, an additional ATA-to-RAM mapping occurs. The experimental results demonstrate that the proposed FGPA-based approach uses only 90% of the memory storage of the GPU-based approach. In addition, the lookup performance is faster by about 2.4 times, whereas the latency is reduced by up to 3 times. Ina practical device, HATA achieves 125 MLPS on average.

Song et al. [14] designed a Patricia trie-based approach to solve the name lookup problem with billions of prefixes. In the Patricia trie, the nodes that have one child are merged with their parents to reduce the memory size and time cost. It is considered a suitable choice for the following reasons: First, the binary trie method can compress the shared parts in prefixes. Second, names can be processed as bit strings in any name system. Third, the name can be used directly in lookup and forwarding without the need for parsing. To enhance the name lookup, speculative forwarding is presented, which is a policy that sends packets using the longest prefix classification (LPC) rather than LPM. In LPC lookup, if a match exists, then LPC forwarding behaves similarly to LPM and sends the packet to the next hop. If no match exists, then it is still forwarded to the next hop instead of dropping the packet. The dual binary Patricia is used to support speculative forwarding with LPC. Instead of storing the complete name component, the Patricia trie stores only the differences among name components in the first level. Therefore, the memory storage is obviously reduced. Experimental results show that compared with the SRAM-based scheme, which achieves 142 MLPS with approximately 16 million names, the DRAM-based one is better for one billion (1 B) names and achieves 20 MLPS.

The name lookup speed can be increased by reducing the number of comparisons using the port information in trie nodes. Li et al. [15] employed this idea to introduce the P-trie scheme. The nodes in P-trie are divided into the following types: accepted (the node in which the last character of the prefix is stored), advanced (the node in which all prefixes of its children have the same outgoing port, whereas none of its parents have a different outgoing port), and intermediate (all the other nodes). When a corresponding node appears at any advanced node, its port element is returned immediately. If the corresponding node occurs at the intermediate or accepted node, then the search proceeds and the subsequent characters are compared. Conversely, if mismatch occurs at any node, then the port element of the node is returned as the search result. P-trie is implemented in MATA to improve memory proficiency and compared with ternary search trie, ATA, and MATA. Experimental results demonstrate that P-trie is faster than MATA, ATA, and ternary search tree by 3, 7.5, and 15 times, respectively.

Divya et al. [16] presented a name forwarding scheme called N-FIB based on Patricia trie. N-FIB consists of two components, namely, FIB aggregation and FIB lookup. The former component collects multiple entries in one entry. Furthermore, it supports the insertion and deletion operations of NDN forwarding. The latter component searches the aggregated FIB for the LPM of a name. The time complexity of N-FIB with aggregation is $O\left(\frac{N}{K}\right)$, where N is the number of names and K is the depth of trie. N-FIB is compared with BF to demonstrate its memory cost and time proficiency. With a 29-M name table, N-FIB consumes only 264.79 MB and needs 403.841 µs for lookup.

- **Hash Table-Based Approaches**

The LPM using hash table (HT) constrains name lookup. To deal with these limitations, Wang et al. [17] proposed the greedy name lookup method called Greedy-SPHT, which is combined with the string-oriented perfect hash table (PHT). The greedy strategy is exploited to optimize the search path of the prefix name table. Furthermore, the HT-based limitation is treated using a string-oriented HT to

accelerate name lookup speed and reduce memory space occupation. The hash table stores only the signature of the name in the table entry rather than the name itself. Inspired by the distributions of name components, the authors improved the name lookup performance by rearranging the search order of name prefixes based on their component distribution. The greedy-SPHT is compared with the implementation of content-centric networking (CCNx), CCNx-PHT, and greedy-PHT to evaluate its throughput. Experimental results under the tables of the 3-M and 10-M entries show that greedy-SPHT outperforms CCNx, CCNx-PHT, greedy-PHT, and NameFilter by achieving up to 57.14 MLPS while occupying only 72.95 MB memory for 3 M and 247.20 MB for 10 M.

- **Bloom Filter-Based Approaches**

Applying the bloom filter (BF) for FIB can support high-speed name lookup. Unfortunately, it cannot be applied directly. Wang et al. [18] designed a two-stage BF-based scheme called NameFilter. In the first stage, the name prefix is mapped to one memory access BF, and the name length is determined. In the second stage, the name is looked up in a simplified group of the merged BFs depending on the first stage. Experimental results with 10 M entries table demonstrate that it can achieve a throughput of 37 MLPS with a low memory cost of 234.27 MB, which translates to 12 times more speedup than the character trie.

- **Hybrid Approaches**

Several approaches that have been implemented for FIB could combine more than one data structure or exploit additional hardware solutions. The name component encoding (NCE) approach was proposed by Wang et al. [19]. The NCE is used by exploiting the massive parallel processing power of GPU. With a local code allocation algorithm and a complicated transition array, NCE uses one single memory access during the transition process between nodes. In addition, pipeline and CUDA multi-stream (CUDA® is a parallel computing platform and programming model developed by NVIDIA for general computing on GPUs (https://developer.nvidia.com/cuda-zone)) techniques are implemented in GPU to enhance lookup performance and reduce lookup latency at the same time. The name table is arranged as a name component trie (NCT), and the components in NCT are encoded into integer codes to organize the NCT as an encoded name component trie (ENCT). Then, ENCT is restructured as a one-dimensional transition array, in which a transition between nodes is accrued by one single memory access. A local-code allocation mechanism is designed to reduce the memory occupation of ENCT. However, the coded trie method requires encoding content names concurrently. Hence, the component hash table (CHT) is introduced to attain fast component code mapping. Experimental results show that, with 100 µs latency, CHT achieves 51.78 MLPS on a name table with 10 M entries. Compared with the character trie, NCE can save 59.57% on memory storage and support 900 K and 1.2 million prefix per second for insertions and deletions, respectively.

The interest packet, which matches the non-aggregatable prefix, is forwarded with the same length to the next hop, which leads to latency in the off-chip memory. Motivated by this observation, Fukushima et al. [20] proposed a novel scheme for name lookup in FIB. The length of the prefix can be added into the packet that is forwarded to the next hop to avoid seeking the same length again. When a name arrives at the FIB, it is processed through the following steps: (1) The prefix is filtered by a table called leaf prefix count table (LPCT), which holds the number of every prefix and its length. (2) If there is no such prefix, then the BF is called to avoid the redundant hash table probing. (3) The HT is utilized via one single access of the off-chip memory to search the prefix entry. If the prefix entry is found and its child field is 0, then it is the LPM. Otherwise, if the name is taken as an input and the LPM is looked up, then the length and the next hop are returned. This scheme is implemented on hardware and software-based FIB, respectively. The proposed hardware-based scheme can reach approximately 66 MLPS for FIB with less than 50 million names, whereas the throughput of the software-based scheme is 16.6 MLPS.

Yuan et al. [21] proposed the longest name prefix matching (LNPM), which combines binary search and HT with one billion entries. The HTs are constructed for name prefixes with up to K components according to the number of name components. For each node of K HTs, the binary search is performed. If a matching prefix exists in the corresponding HT, then it transits to the right sub-tree to lookup whether another longer prefix is possible. If no matching exists, then it transits to the left sub-tree to lookup the longest prefix. It ends only when it reaches the bottom of the binary search or reaches a leaf FIB entry. The fingerprint-based HT is used to reduce the number of required memory accesses, where the hash values of the keys in the name tables are the fingerprints. Experimental results showed that 6 MLPS can be reached with one billion entries in a seven component-name table.

Perino et al. [22] designed a content router called Caesar with a novel prefix bloom filter (PBF) combined with a HT that can support packet forwarding with a GPU to achieve a positive wire speed. PBF identifies the length of the LPM using one single memory access in the first stage. Afterwards, it uses the HT lookup to extract the next hop information or exclude false positives in the second stage. Experimental results demonstrated that the line card of Caesar maintains about 95 MLPS with a packet size of 188 bytes and a table size of 10 million.

Li et al. [7] presented a design for the fast LPM using a hybrid data structure of hash, fat trie, and array. It is composed of two name lookup stages, namely, slow path and fast path. For the former, when a name prefix arrives at an NDN node, a name reduction algorithm is applied to convert the name prefix into a non-human-readable compressed key. Then, the transformed key is inserted into or removed from FIB using the dual-hash function. For the latter, when a router receives an interest packet, the name reduction cannot be applied. Instead, a key set generation algorithm is used with the name prefix to produce a set of unstructured keys. Experimental results demonstrate that the 37 and 70 MLPS name lookup throughputs can be achieved on CPU and GPU platforms, respectively.

Quan et al. [23] proposed one a name lookup engine with an adaptive prefix bloom filter (NLAPB). In NLAPB, each prefix is divided into B- and T-suffixes. The B-prefix (B refers to Bloom) is a limited length, which is processed using counting bloom filters (CBF). AHT is used with the B-prefix to verify the location of the corresponding T-suffix tree (T refers to tree). The T-suffix, whose length is variable but shorter than its full name, is processed using the tree. Through the tree, the prefix matching process proceeds until the longest prefix is matched. The lengths of the B- and T-suffixes are controlled by their popularity to speed up lookup. For T-suffixes, the matching process leads to many repetitive memory accesses. Hence, the length of the B-prefix should be variable according to the popularity of a prefix. If the popularity is stronger, then the length of the B-prefix should be higher and vice versa. Compared with components-trie (CT) [6], bloom-hash (BH) [24] and HT [25], NLAPB achieves up to 800 K for lookup processing, where the average of name components is 6, whereas CT, BH, and HT achieve 700, 550, and 680 K, respectively. With regard to the memory cost, NLAPB shows an average consumption of 400 MB, whereas BH achieves the best performance with about 300 MB. CT and HT consume about 600 MB and 350 MB, respectively.

Yun et al. [26] used the MATA data structure, which was introduced in another study [11], as a data structure for their split name-character trie (SNT). First, a city hash function is used to represent the NDN name as a 32-bit integer. Second, SNT uses these integers as state transitions to create a character trie. Finally, the created character trie is stored as a new data structure called MATA-CH, which differs from the MATA in two aspects: (1) MATA-CH is more flexible because it is no longer limited by stride and (2) it reduces the number of state transitions because each component is stored as one state transition. Inspired by the split routing lookup model (SRLM) [27], SNT is used to enhance the throughput of name lookup. The trie is divided into two smaller sub-tries. Once a new name arrives, it is split into two parts to perform lookup separately in the two sub-tries. SNT can reduce memory cost by about 30–38% using component hash and by 30–33% using splitting trie. In spite of the large memory using the HT, SNT can improve memory cost by 23–49%.

Yuan et al. [28] enhanced the name lookup throughput by using fingerprint-based methods with a hybrid data structure that is composed of the Patricia trie and HT. On the basis of speculative

forwarding, an exact string differentiation (ESD) problem is formulated. In ESD, no corresponding prefixes exist in the router table. The required process is prefix differentiation rather than prefix matching. Following this scheme, the fingerprint-based method is proposed. By decreasing the height of the trie, the fingerprint-based Patricia trie (FPT) reduces the lookup latency introduced by Patricia trie in [14]. Concerning memory storage, a fingerprint-based hash table (FHT) stores only the fingerprint of the name. The implementation of proposed FPT and FHT schemes shows that the proposed fingerprint-based Patricia trie can reduce the lookup latency to 0.29 µs while using only 3.2 GB memory to store one billion names, each of which has only one name component with an average length of 30 bytes.

3.2.2. PIT Schemes

The PIT is as important as the CS and FIB table due to its frequent update operations. Each PIT entry contains the following fields: content name, nonce, incoming interface, outgoing interface, and expiration. Content name and nonce are used to identify the interest packet and stops the duplicate forwarding. The incoming and outgoing interfaces represent the source and destination of the interest packet in the PIT entry. When the NDN router forwards an interest packet, the expiration time of the interest packet is associated with its corresponding entry. If the data packet fails to return upon the expiration time, then the corresponding PIT entry is deleted [6].

The following three operations are performed on the PIT: update, insert, and delete. When a new interest arrives at the PIT, the content name of the interest is used to check if the content's entry already exists in the PIT. If the content entry is not extant in the PIT, then the insert operation is performed by creating a new entry. If the entry exists and is not expired, then an update operation is performed to prevent duplicate packet forwarding. The delete operation is implemented when the corresponding data are received or the entry is expired. Such an operation consists of one lookup operation and one low-level delete depending on the data structure [3].

- **Trie-Based Approach**

The first detailed study of the PIT memory size was performed by Dai et al. [3]. To make PIT more scalable, the encoding idea in [5] is used to reduce PIT size and speed up the access operation. When an NDN packet arrives at the PIT, each of its name components is obtained and assigned a code generated through NCE. Then, the code strings are added to the ENPT, which, in turn, is used to lookup, insert, and delete the PIT entries. With the use of NCE, PIT memory size could be reduced by 87.44%; furthermore, the resulting PIT exhibits good scalability. With respect to PIT access operations, evaluation results demonstrate that the PIT lookup, insertion, and deletion speeds are 1.4, 0.9, and 0.9 MLPS, respectively.

Divya et al. [6] proposed an efficient name encoding approach called Radient, in which one name component is replaced by a unique code (32-bit integer), such that identical components share the same code. The Radient approach consists of two main parts: a component radix trie (CRT) and an encoded name radix trie (NRT). CRT consists of two parts: the available token stack, which keeps track of available tokens that are either newly created or deleted, and a token frequency map, which keeps track of every component added to the PIT. As shown in Figure 4, when a new packet request is inserted to the PIT, its name content is decomposed into components, each of which is searched in the CRT. If a component exists in the CRT, then its allocated code is returned. If no matching occurs in the CRT, then the component is added and a new code is generated in the available token stack. Afterwards, the frequency of the incoming component is increased by 1 using the token frequency map. Then, the encoded name is forwarded to the NRT for the PIT update. For the deletion operation, the lazy deletion mechanism is adopted. The memory required for a 29 M dataset to implement the Radient approach is 140.10 MB. Bycontrast, other approaches have the following requirements: ENPT (217.05 MB), NPT (301.00 MB), CCT (654.08 MB), and HashMap (329.75 MB).

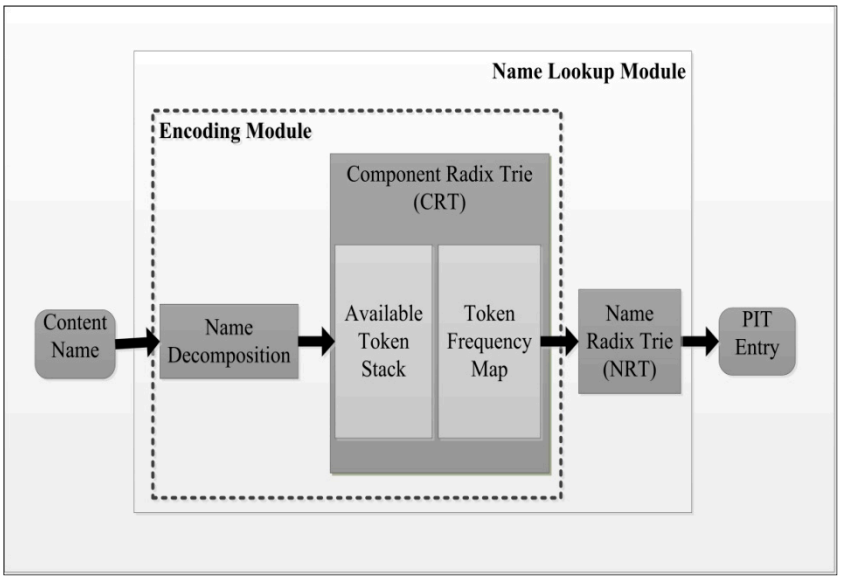

Figure 4. Framework of Radient design.

- **Hash Table-Based Approaches**

Varvello et al. [29] focused on designing and implementing a PIT that can sustain wire speed. The placement and data structure of PIT affects its performance. The linear chaining hash-table (LHT) and open-addressed d-left hash table (DHT) were proposed as data structures for PIT name lookup. According to [3], the PIT can be placed on input only, output only, input and output, and third-party modes. Experimental results show that the third-party mode is the most promising because it needs only one probe to realize all PIT operations. By using the DHT, the PIT can process traffic of up to 10 Gbps and store up to 1 million string entries.

Yuan et al. [30] proposed a design for the PIT with a DHT based on the idea of the core and edge routers of IP-based networks. In this design, instead of name strings, fixed-length 16-bit fingerprints are stored for the core routers, while the edge routers remain as usual. However, fingerprint collision and the disability of distinguishing such collision arise from duplicate interests. The fingerprint collision problem is mitigated by relaxing the interest aggregation feature of the core routers. First, the interest packet, which arrives at the core router, is forwarded. Then, to implement fingerprint duplication, most interest aggregations occur at the edge routers, and CS is also used to avoid duplicated interest requests. Experimental results show that using this approach, PIT memory size is reduced for core routers, and either SRAM or RL-DRAM can be used to support name updates at 100 GBPS with only 37 MB to 245 MB of memory.

Giovanna et al. [31] achieved platform implementation using an enhanced open-addressed HT for the PIT. They provided a PIT model to evaluate the average and maximum PIT sizes at the steady state. The authors concluded that PIT size is small under normal network settings even without caching and with optimal network bandwidth.

- **Bloom Filter-Based Approaches**

You et al. [32] proposed a data structure design called a distributed PIT (DiPIT) by deploying a small-size memory for every interface based on a CBF. The PIT table is divided into sub-tables called PITi. A PITi is implemented so that using a BF stores only the list of PIT packets in the associated face. Whether hierarchical or flat, DiPIT does not rely on the naming method. Compared with the HT method, DiPIT can reduce memory cost by 63% and achieve high packet name lookup.

Li et al. [33] presented the united BF (UBF) to compress the PIT effectively. With the use of UBF, the counter leak and false negative of the CBF is avoided. The UBF contains the current and noncurrent BFs. Each PIT name time is divided into epochs. At the beginning of each epoch, the noncurrent BF

is set to 0. If a new content name is inserted into the PIT, then the insertion is implemented on both current and noncurrent BFs, whereas the lookup is implemented only on the current BF. At the end of the epoch, the two BFs swap their values. The current BF represents the interest packets received in the last and current epochs. After each name is inserted into one BF, the name will be removed at the end of the next epoch. Experiments show that with CBF, reductions occur in storage (10%) and error probability (0.1%). However, with UBF, storage can be reduced by up to 40% and error probability can be reduced by around 0.027%.

Li et al. [34] proposed a mapping bloom filter (MaPIT) design which is composed of an index table and packet store. The index table consists of two structures of bit arrays: a regular BF and 30-bit mapping array (MA). The BF is used to check whether the entries exist in the MBF or not. The value of the MA is exploited to access the packet store. The BF comprises n equivalent parts, each of which matches one bit of n bits MA. For each new incoming entry, the initial state of its corresponding bit in MA is 0. When an entry is added to the MBF, a few bits of MA maybe changed to 1 s according to their corresponding hash values in the BF. MaPIT utilizes an on-chip memory for the index table and an off-chip memory for the CBF and packet store. In MaPIT, the packet names are encoded using MD5 and SHA1 functions. MaPIT performance is evaluated and compared with the NCE and HT schemes. Moreover, MaPIT has the lowest memory consumption and the lowest false positive probability relative to NCE and the HT.

Tsilopoulos et al. [35] suggested a semi-stateless forwarding method for the PIT. The request is traced at specific hops instead of traced in every hop. Then, the request collects the reverse path information at intermediate hops to be used in delivering responses among routers by using the BF. Through this scheme, forwarding cost is efficiently reduced by 54% to 70% for unicast delivery and 34% to 55% for multicast delivery.

- **Hybrid Approaches**

Yu et al. [9] implemented a combination of BF and linear-chained HT by co-design of hardware and software. The PIT operations are integrated into one command. The packet name is divided into two parts: content name and segment number (sn of 32-bit). Using hash functions, the variable-length content name is encoded to a 64-bit name ID (nid). The name ID table (nidT) is included to store the distinct name IDs in the PIT. If the content name is already inserted in the nidT, then the router does not search against the FIB for forwarding the interest packet. A limited on-chip block RAM is used to apply a BF for the lookup table, while the full lookup table is stored in an external SRAM. The key (nid, sn) is mapped to a bucket using a hash function. This approach was implemented on a FPGA, and its overall packet processing rate was up to 60 MLPS.

3.3. Entire NDN Router Schemes

The original NDN proposal introduced by [1] suggests the exploitation of the massive power of TCAM to improve name lookup performance for all NDN components. Unfortunately, TCAM is impractical due to its low memory, high price, and excessive power consumption [19]. However, a few studies presented some designs to accelerate name lookup in the entire NDN router without focusing on a specific part of an NDN component by using the HT and BF methods. The trie-based method is not applied given its different nature of name lookup in the FIB (which uses the LPM) and the PIT/CS (which uses EM). This section introduces these works, which we call the general schemes.

- **Hash Table-Based Approaches**

Yuan et al. [36] presented a thorough study of the present CCNx implementation. They confirmed that CCNx needs extensive efforts to achieve scalable performance and provided a proper definition of the main scalability problem in NDN forwarding. The packet forwarding operations in NDN and CCNx were studied in detail. Three key issues in NDN name lookup were investigated: exact string matching, longest prefix matching, and large-scale flow-maintenance. CCNx implements a name prefix

hash table (NPHT) to merge the FIB and PIT in one HT. Each NPHT entry has a pointer indicating the PIT and FIB. If an interest's name component matches the entry added by previous interests for the same name, then the pointer provides faster prefix lookup by decreasing the number of the required comparisons for the LPM. For steady name lookup, they suggested mapping a longer name to a shorter name at every NDN node. They recommended exploiting the features of the URL. One feature they suggested is that the EPM and LPM should take the name component as a matching unit rather than just considering the name as characters. Another feature is that the distribution of name components in the URL can be used to attain outstanding performance.

After studying various designs, So et al. [37] proposed a forwarding engine for an NDN router based on the chained HT. Using a 64-bit SipHash, the generated value is stored at every HT entry to minimize string comparison. For FIB name lookup, a two-stage LPM algorithm is used. In the first stage, the lookup starts from the most occupied component level denoted as M and moves to the next shorter prefix until prefix matching occurs. If prefix matching does not exist, then the interest has no LPM in the FIB. The second stage starts if a match is found and shows a potential LPM at the maximum depth of the prefix, denoted as MD (MD > M). Lookup begins from the MD, and the search continues until a match is made. Given that the PIT and CS use the full matching method for lookups, they are combined in one HT lookup. The proposed forwarding design could achieve an overall packet processing of up to 8.82 MLPS.

Hsu et al. [38] encoded the length variable prefix into 32-bit fixed length by implementing a cyclic redundancy check (CRC-32) hash function to minimize the name lookup in the NDN router. All encoded components, excluding the slash symbol, are chained to form a 32-bit encoded name prefix as shown in Figure 5. If a new data packet is received by an NDN router, then its name prefix is encoded and then the data packet is stored in the CS. When a new interest packet with a variable length prefix name arrives at the NDN router, its name prefix is encoded before it is forwarded to the CS, PIT, and FIB. With the use of the CRC-32 encoding method, the storage space is reduced to 26.29% and the lookup speed is increased by 21.63% compared with the original NDN method.

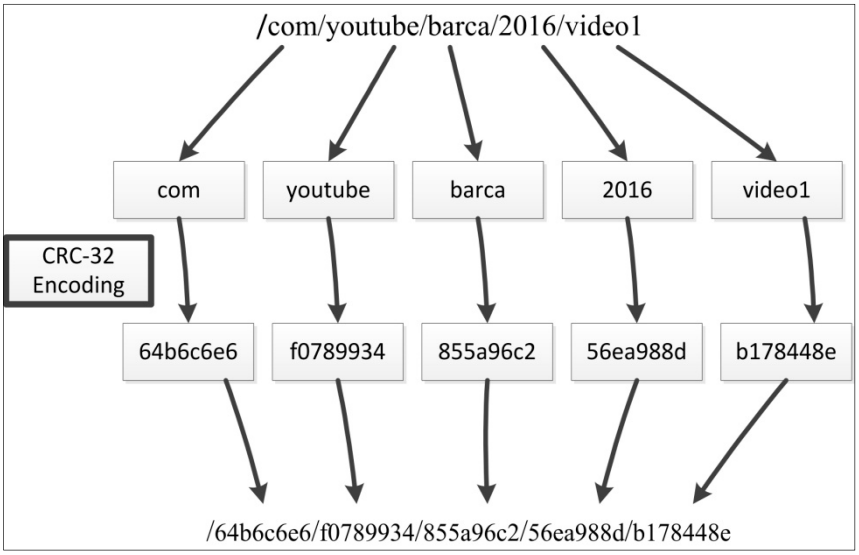

Figure 5. Encoding name prefix based on CRC-32.

- **Hybrid Approaches**

After exploring the deterministic finite automata (DFA) and HT, So et al. [25] found that HT is easier and better than DFA due to the variable length names in the NDN. To avoid additional comparison and prevent collision, a chained HT with the linked list is used. The HT is also combined

with the BF and data prefetching to improve the performance of name lookup in the CS and PIT, hereby achieving 1.5 MLPS for the interest packet and 2.14 MLPS for the data packet.

A unified index needs only one index lookup for forwarding, and Dai et al. [10] designed a unified index data structure for the CS, PIT, and FIB called the 'bloom filter aided hash table'(BFAST). BFAST exploits the counting BF to distribute the load among HT buckets. To obtain a good lookup performance, an auxiliary counting BF is used to check which hash function determines the index of the smallest counter during item update. When implementing BFAST in the CS, PIT, and FIB, the signature of the name or prefix, which is calculated by the hash function, is stored in the index, while the name or prefix is stored in a field of the table entry. Evaluation results showed that BFAST can achieve 36.41 MLPS for LMP lookup and can reach a lookup speed of 81.32 MLPS with an artificial request trace for the whole NDN architecture. BFAST outperforms the NameFilter, NCE, and CCNx methods, but its memory consumption is higher than that of others. Figure 6 depicts BFAST and its data structures.

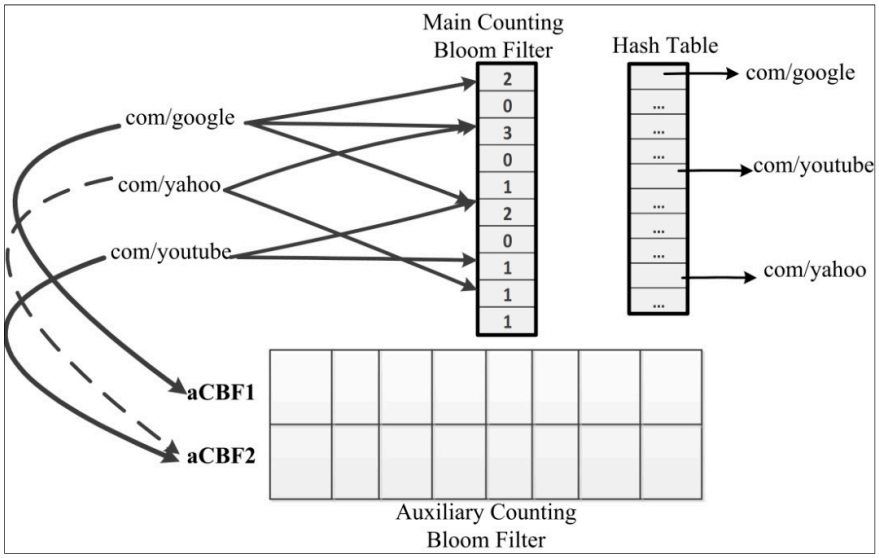

Figure 6. BFAST data structures: CBF, auxiliary CBFs, and the HT.

Huang et al. [39] presented the steerable name lookup based on the Bloom Filter (SNLBF) for the NDN. The prefixes in the SNLBF are classified according to the component number and the prefix length for mapping them into different BFs. These mapped prefixes are stored in different HTs. Each entry in the HT is allocated memory of a particular size. As a result, the number of memory access is decreased to one. A memory-efficient component steerable structure, called the number of component steerable structure (NCS), is designed to optimize the number of components when performing the LPM. Evaluation results demonstrate that with 2.5 M table name, the SNLBF achieves better lookup performance than CCNx and BH, but BH and CCNx require slightly more memory than the SNLBF.

4. Comparative Analysis

This section will compare the packet name lookup approaches that were previously presented in NDN. Although approaches could include other issues such as scalability and latency, we take memory consumption and lookup performance as the main benchmark. Comparing results obtained with different input datasets is difficult. Therefore, for memory consumption, we evaluate the memory consumption with respect to the original size of the dataset. For lookup performance, we measure the lookup speed for packets in seconds. However, some aspects of the performance of approaches are missed.

4.1. Current Solution Analysis

4.1.1. Trie-Based Schemes

The current trie-based schemes need efforts to fit NDN name lookup [40]. The time complexity of the trie-based solution is dependent on the number of name components traversing through nodes according to the name length. Lookup performance degrades when the tree depth increases. The transition between nodes must match the component in the whole character trie, and thus, the name lookup process spends extra time to search the longest prefix from the root node to the leaf node. Mapping the component and code, which is stored in a character trie, slows the name lookup [17]. The component trie requires a particular encoding, which creates additional processing and creates a bottleneck for the entire lookup process. However, the character trie does not have the encoding scheme; therefore, the lookup performance may decrease as the length of an input name increases since the lookup is processed character by character [37]. Implementing the trie data structure combined with a hardware scheme achieves high-speed lookup but requires extra chips and storage. Using the trie bitmap to mitigate this problem needs the lookup of numerous off-chip memory accesses. Moreover, increasing the stride size of the trie bitmap rapidly expands the bitmap size and off-chip access. When a new child is inserted, other child nodes must be moved to other locations and all child nodes should be shifted in the worst case [6]. NCE provides poor support of EM lookup in the CS and PIT. Furthermore, as more prefixes are added, the trie nodes gain many outgoing edges; thereby slowing down the lookup process [10]. Trie-based FIB lookup does not completely consider the variable and unbounded length name [20]. The tree depth can be very long because of the length of the NDN name.

4.1.2. HT-based Schemes

HT-based solutions perform numerous hash computations, which notably reduce lookup performance [11]. Hash-based approaches require $O(1)$ comparisons for EM and $O(L)$ for LPM, where L is the length of name [20]. These hash-based methods all have false positives caused by hash collisions. Given that hash collision leads to packet forwarding to the wrong destination, it causes more memory accesses and thus increases time complexity [11]. Without further remedy, false positives weaken the reliability of an NDN router [19]. When storing name content as a fingerprint, name string comparison is easily avoided via checking the fingerprint only; however, in any case, one insert operation is needed to store the content name on DRAM memory [30].The overflow table of the D-left HT, which stores the item that cannot be matched in the HT, may facilitate the resolution of the bucket overflow issue, but it slows down the lookup process. In addition, the possibility of duplication in fingerprint matching within the same hash bucket should be considered [9]. Multiple hash functions outperform the single hash function, but the former requires considerable memory for traversing all the candidate buckets to find the least loaded bucket [10].

4.1.3. BF-Based Schemes

Many aspects readily obstruct the BF application in NDN name lookup [40]. The BF presents only a binary "yes" or "no" answer and cannot determine the identity of the matched entry. Moreover, the BF may create a false positive depending on the entries added into it. Given the possible large number of NDN names, the false positive rate is not easily controlled in long names. Furthermore, hash computing for the possible prefixes of NDN names is needed in BFs, which will degrade the lookup efficiency, particularly for long names. To enhance the BF to support update operations, the CBF is used by replacing the bit-vector of the BF with an array of counters. However, the CBF solution has two potential shortcomings. First, multiple hash functions are required to reduce the false positive rate. With multiple hash functions, an update operation needs two memory accesses: one to acquire the existing count values and another to update those values. Second, the size of the CBF counter is constrained by the implemented hardware. In addition, when counter overflow arises, the CBF cannot

correctly perform update operations. When the counters use more bits, the CBF may not be stored on-chip [9]. In NDN, the BF must be stored in DRAM because the expected number of rules is large. Conversely, the NameFilter scheme [18] has two essential restrictions. First, it cannot decrease the false positive rate, and thus, some packets may be sent to the wrong destinations. Second, it cannot support dynamic forwarding and multi-path routing [22]. In BFAST, the packet forwarding rate is restricted by the update rate due to the per-packet update in the PIT and CS [9].

4.1.4. Hybrid Schemes

Most hybrid solutions exploit the HT either with a BF or trie. Combining the BF and HT creates a few drawbacks. Aggregation is still required and the memory for the HT is large for a fast memory chip. The fast BF concentrates on reducing the times of memory accesses on the slow HT. Nevertheless, this method is currently not a good solution for the aggregated table in the FIB because the BF could damage the hierarchical structure of the name prefix. However, such an approach can be used for the PIT and CS tables because they perform exact matching [32]. A hybrid trie and BF method achieves only 1 MLPS for the FIB packet lookup [9].

Implementing a solution with specific hardware like the GPU or FPGA is suitable for NDN routers and could reach 60 MLPS. However, such technique suffers from several limitations. Given the per-packet update constraint, low memory density, large power consumption, and high cost, such hardware solutions are unlikely to be implemented in an NDN router [17]. Nowadays, the throughput of CPUs has increased because of developments such as CPU caching, high memory bus and bandwidth, and short execution pipelines. The increased demand of virtualized systems requires an elastic infrastructure with less hardware. In addition, some software approaches achieve high performance with less per-packet delay [6].

4.2. Performance Comparison

The performance of schemes introduced in this survey varies in many aspects due to the nature of NDN components and the variable length names. While some approaches reveal high-speed NDN operations, their performance in memory consumption, latency, power consumption, or scalability may not be satisfied, and vice versa.

4.2.1. Lookup Performance

Choosing a suitable data structure affects time efficiency when applying various methods in NDN name lookup. As we noticed during the exploration of various methods, BF-based solutions achieve high lookup speed relative to other schemes. However, for insert and delete operations, the BF has low performance due to its nature of providing only "yes" or "no" replies. For example, the NameFilter scheme is fast, but it does not completely support incremental updates [10]. The trie-based solutions are simple in design and efficient in terms of time, but they have high lookup cost because their name lookup time depends on the name length. The cost of a trie-based update is less than that of the HT and BF. The HT performs only $O(1)$ comparisons for EM, and for LPM with a name length B, it requires $O(B)$ comparisons in the worst case. The HT update also requires refreshing the whole table whenever inserting or deleting an element. Finally, implementing a software scheme with additional hardware demonstrates high-speed lookup, but the performance of deletion and insertion is not improved as in [3,19]. Thus, hybrid solutions may not achieve noticeable enhancement compared with pure methods as shown in [10,17].

4.2.2. Memory Consumption

Memory consumption is another concerning issue because the NDN router contains millions or billions of names. The BF method consumes a low amount of memory (Table 3), but it still suffers from false positives. The trie scheme (Table 4) also consumes less memory compared with the HT. Conversely, the HT-based method (Table 5) naturally consumes less memory, but it tends to consume

a large amount of memory due to the requirement of decreasing the probability of false positives; high memory corresponds to low incidences of false positives, and vice versa. Combining a software solution with a hardware one creates a trade-off between throughput and memory. Enhancing lookup performance using a hybrid solution (Table 6) negatively affects the memory consumption, as shown with the BF and linear-chained HT [9], MATA [11], NCE-GPU [19].

Table 3. Comparison of lookup throughput based on BF schemes.

Method	Data Structure	Lookup Performance	Memory Consumption Ratio	Dataset	Dataset Size (MB)
NameFilter [18]	FIB	37 MLPS	30.1% (299 MB)	3 M and 10 M	N/A
DiPIT [32]	PIT	N/A	63%	200 M packets per second	N/A
UBF [33]	PIT	N/A	45% (110 MB)	4 M	200 MB
MaPIT [34]	PIT	N/A	99.34% (2.097 MB) on on-chip memory	2 M	320 MB

Table 4. Comparison of lookup throughput based on Trie-based schemes.

Method	Data Structure	Lookup Performance	Memory Consumption Ratio	Dataset	Dataset Size (MB)
NCE [3]	PIT	1.4 MLPS	71.8% (167.85 MB)	18 M	595.04 MB
NCE [5]	FIB	1.3 MLPS	46.64%	6 M	957.33 MB
Radiant [6]	PIT	0.051 MLPS	87.63% (140.1 MB)	29 M	1132.65 MB
MATA [11]	FIB	63.52 MLPS	70.5% 149.92 MB 490.28 MB	3 M and 10 M	Nearly 470 MB and 1700 MB
PNL [12]	FIB	N/A	N/A	2M	N/A
HATA [13]	FIB	125 MLPS	63.45%, a reduction of 90% compared with [11]	1 M 10 M	N/A
Patricia trie [14]	FIB	142 MLPS for SRAM 20 MLPS for DRAM	80.53% (7.32 GB)	1B	37.6 GB
P-trie [15]	FIB	13 MLPS	N/A	100 K	N/A
N-FIB [16]	FIB	404 microseconds for 29 M	96.94% (264.79 MB)	29 M	8617.595 MB
NCE-GPU [19]	FIB	51.78 MLPS	61.3% (650.49 MB) 120.86 MB 529.63 MB	3 M and 10 M	1680.89 MB
SNT [26]	FIB	N/A	35.3% (110 MB)	1 M	Nearly 170 MB

Table 5. Comparison of lookup throughput based on HT schemes.

Method	Data Structure	Lookup Performance	Memory Consumption Ratio	Dataset	Dataset Size (MB)
NPHT [36]	FIB, PIT, CS	N/A	N/A	N/A	N/A
Greedy-SPHT [17]	FIB	57.14 MLPS	45.11% (320.15 MB)	3 M and 10 M	583.27 MB
LNPM [21]	FIB	6 MLPS for 1 billion names of 7 components	34.23% (111.8 GB)	1 B	170 GB
DHT [29]	PIT	3.4 MLPS	N/A	N/A	N/A
D-left hash table [30]	PIT	0.83 MLPS	64.49 MB	1M	N/A
Semi-stateless forwarding method [35]	PIT	N/A	45–66% of CCN	N/A	N/A
SipHash [37]	FIB, PIT, CS	8.8 MLPS at 20 Gbps	35.82 MB	1 M	N/A
CRC-32 hash function [38]	FIB, PIT, CS	Improving by 21.63% from the NDN proposal	31.93% (618 KB)	1 M	908 KB

Table 6. Comparison of lookup throughput based on hybrid schemes.

Method	Data Structure	Lookup Performance	Memory Consumption Ratio	Dataset	Dataset Size (MB)	Notes
Hybrid data structure of hash, fat trie, and array [7]	FIB	37 MLPS (CPU) 70 MLPS (GPU)	N/A	12 M	1088 MB	Focuses on measuring the number of packet lookups per second
BF and linear-chained HT [9]	PIT	56–60 MLPS	N/A	2 M	N/A	Focuses on the packet processing rate
BFAST [10]	FIB, PIT, CS	36.4 MLPS	50.3% (419.32 MB) 48.5% (1517.6 MB)	3 M and 10 M		High speed and low memory
Non-aggregatable prefix [20]	FIB	66 MLPS	N/A	620 M	1 GB	Focuses on measuring the number of packet lookups per second
Caesar [22]	FIB	95 MLPS	N/A	10 M	42 MB	Focuses on measuring the number of packet lookups per second
NLAPB [23]	FIB	800 KLPS	500 MB	10 M	N/A	
A chained HT with a linked list [25]	FIB, PIT, CS	1.5 MLPS	N/A	Names with an average length of 33.9 bytes	N/A	
FHT [28]	FIB	N/A	57% (3.2 GB)	1 B	N/A	
SNLBF [39]	FIB, PIT, CS	3.5 MLPS	50 MB	3 M	N/A	

4.2.3. Scalability

Designing a scalable NDN router is primarily affected by variable-length name and per-packet update. The hierarchical name is also critical to scalability because it can be aggregated. Therefore, the hierarchical instead of the flat name should be used. The trie-based solution degrades when more names are inserted into the NDN router because the scalability depends on the trie depth. Some BF approaches can solve the scalability problem by combining the second stage of the BF [18]. For example, the NameFilter scheme can save 77% for 3 M name entries and 65% for 10 M name entries. The HT-based solution has low scalability (Table 5) for insertion and deletion, and combining the HT and BF achieves scalable performance. Implementing the data structure with additional hardware causes considerably slower memory expansion compared with other methods. Thus, such an approach maintains reasonable scalability when the memory space occupied by names is increased.

4.2.4. Latency and Validity

The BF and HT suffer from false positives. A false positive affects the validity of the NDN network because packets are sent to wrong destinations. In the HT, latency depends on the selection of the type of hash function. For example, with the SipE and CityA function, latency could be reduced by 25% and 33% respectively, at the cost of increased hash computation [18]. As stated, some solutions are enhanced with additional hardware and achieve high-speed lookup; however, the latency increases because the packet must undergo additional hardware initially before it returns to the NDN router. With the trie-based solution, latency depends on the depth of the trie. Moreover, the long names lead to high packet latency.

5. Name Lookup Related Projects

Name lookup is a critical issue for the content name-based networks' projects. Several projects were funded for designing a content-based network for future internet [41], such as Named Data Network (NDN), Combined Broadcast and Content Based (CBCB) [42], Data Oriented Network Architecture (DONA) [43], Network of Information (NetInf) [44], and Publish Subscribe Internet

Technology (PURSUIT) [45]. Among these projects, NDN is a promising design to deal with application generated content-based and location-independent names to forward contents to the requesters, irrespective of their address. NDN name lookup remains an open issue that needs more research and improvements.

6. Future Work

The NDN router update operations induce many challenges for the name lookup. Although many schemes have been proposed, more practical and flexible schemes must be investigated to provide low processing cost and high-speed lookup in the NDN router. This section explains the future directions in NDN name lookup as shown generally in (Table 7).

Table 7. Future directions for NDN name lookup.

Approach	Future Directions
Trie-based approaches	• Reduce the high update rate. • Reduce the scalability. • Reduce the number of memory accesses required for each entry. • Study some related techniques such as load balancing hash with parallel.
BF-based approaches	• Make real studies with a large set of names. • Study caching optimization and names distribution. • Implement the BF with the aided of the GPU solutions.
HT-based approaches	• Full implementation and further optimization. • Large-scale studies. • Study some HT technique, such as summary BF, bucket prefetching or ad hoc solutions for multi-type data structures.
Hybrid approaches	• Achieve better performance in memory consumption.
General suggestions	• Make update methods meet the practical requirements in real NDN. • Schemes should be studied for more than one billion names and deployed on real platforms. • Study whether a flat name and a self-certifying flat can support high performance or not. • Study forwarding strategy and NDN component placement and their impacts on name lookup. • Differentiate between the NDN components to facilitate the selection of the best choice for each component.

6.1. Trie-Based Approaches

NDN sends/receives packets using variable and unbounded length names, thereby entailing high memory consumption. Consequently, a high NDN update rate generates a scalability challenge for name resolution, and reducing update time plays a significant role in name lookup performance. Such reduction is directly related to the minimization of the number of memory accesses required for each entry. In trie based schemes, the time required for insertion, deletion and update should be reduced. For future study we can study whether reducing the number of memory accesses for name lookup will reduce lookup time. We also suggest studying some techniques such as load balancing hash [46] with parallel access to reduce both lookup time and trie's depth.

6.2. BF-Based Approaches

Implementing the BF based schemes on NDN need to make real studies with a large set of names to enhance the NDN performance. Studies with large sets of names will improve the evaluations of the lookup time, the table size, the network load and the optimal false positive. In future BF work, caching optimization and names distribution should be taken into the account. We suggest implementing the BF with the aided of the GPU solution to improve the NDN name lookups.

6.3. HT-Based Approaches

In HT schemes, more efforts are needed for NDN router such as full implementation, further optimization, and large-scale studies to enhance the name lookup performance. Therefore, we suggest as future work studying further HT optimization technique, such as summary BF [47], bucket prefetching or ad hoc solutions for multi-type data structures in NDN.

6.4. Hybrid Approaches

Adopting hardware approaches—such as through the GPU and FPGA—demonstrates high performance, but their high cost in terms of power, memory, and latency required increased attention. Memory is not a major concern due to the development in memory storage technologies. Therefore, the hybrid schemes with hardware approaches achieved high performance and are promising solutions to the name lookup issue. We noticed that the hybrid approaches have been attracted more consideration as Figure 7 depicts. Therefore, we suggest focusing on these approaches to achieve better performance in name lookup.

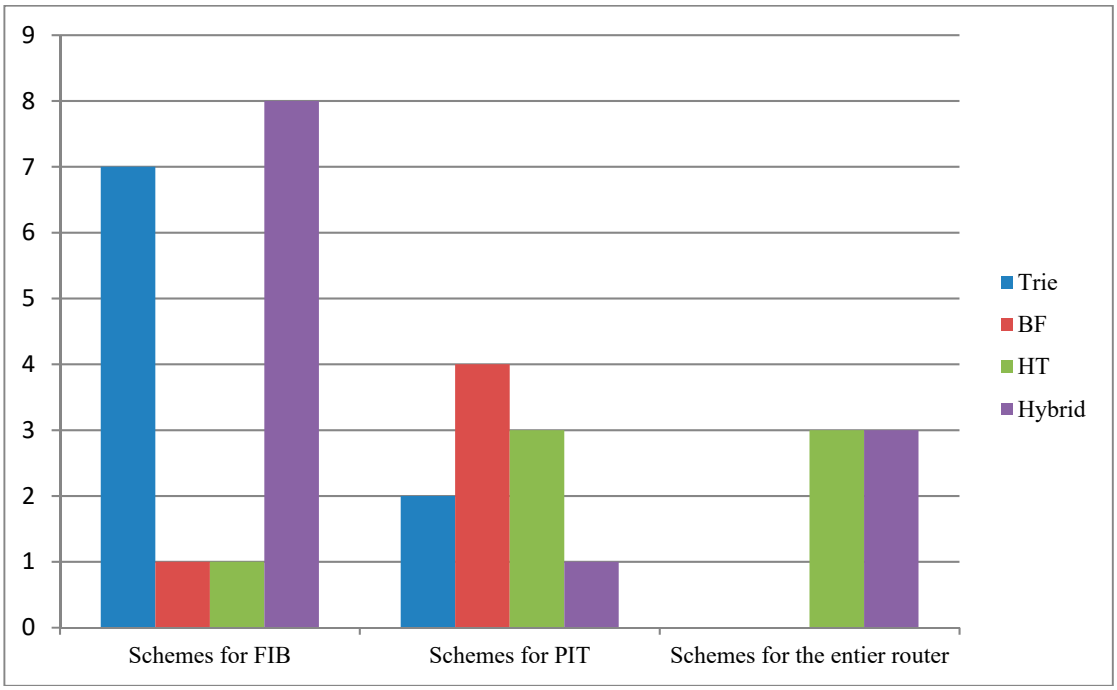

Figure 7. Distribution of the paper schemes.

6.5. General Suggestions

Making NDN update methods meet the practical requirements in real NDN needs more efforts for further study in future work. NDN topology modifications and content name insertion or deletion cause NDN router updates. Given that no NDN is deployed today, NDN update cannot be precisely measured. Besides, most introduced schemes treated the name lookup issue only under the scale of a

few million names. Therefore, schemes should be scalable for more than one billion names and their deployment on commodity devices or real platforms.

The NDN naming mechanism affects name lookup performance and remains a considerable challenge for further study. Given name aggregation in the hierarchical names, any change in the content hierarchy by a producer or provider changes the content name. Conversely, a flat name does not require a high cost for finding LPM. To the best of our knowledge, no research explains whether a flat name can support high performance or not. A self-certifying flat name could be suitable for practice. Nevertheless, no general agreement exists in the research community regarding which naming mechanism is suitable for NDN [2]. Other factors also affect name lookup performance, such as forwarding strategy and NDN component placement. These issues are out of our scope, but require more attention in future work.

As we know, an NDN router is composed of three components (the FIB, PIT, and CS) with different principles. While the FIB name lookup is based on the LPM method, the CS and PIT counterparts are based on the EM method. This disparity creates a major challenge for finding a suitable data structure that can solve this issue with two variable types of lookup. We suggest that any approach should differentiate between the NDN components to facilitate the selection of the best choice for each component. High-speed lookup and low latency with scalability are aspects that require more attention when introducing new approaches for NDN name lookup.

7. Conclusions

The vast number of names and their variable length present considerable challenges to NDN. NDN requires per-packet update preceded by name lookup operation. Aside from its lengthy processing time, NDN consumes a large amount of memory, which means NDN tables grow rapidly. In this paper, we surveyed the most popular NDN name lookup approaches in a novel classification. This survey also highlighted the issues that affect name lookup performance in NDN, such as scalability, memory consumption, time efficiency, latency, and validity. The trie-based, BF-based and HT-based data structures were used to perform the name lookup process. For the FIB, the trie-based data structure is the most frequently used, while its HT-based counterpart is the most frequently used for the entire router. While every solution has its pros and cons, the most promising solution is the one that exploits the pros of the data structure while avoiding its cons. Thus, we indicated some directions for future work and open issues to improve NDN name lookup in terms of minimizing implementation cost and enhancing performance.

Funding: This work was supported by the Program for Liaoning Innovative Research Term in University under grant no. LT2016007.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. NDN Project Team. Named Data Networking (NDN) Project, Technical Report NDN-0001. 2010. Available online: http://named-data.net/publications/techreports/ (accessed on 5 February 2019).
2. Saxena, D.; Raychoudhury, V.; Suri, N.; Becker, C.; Cao, J. Named data networking: A survey. *Comput. Sci. Rev.* **2016**, *19*, 15–55. [CrossRef]
3. Dai, H.; Liu, B.; Chen, Y.; Wang, Y. On pending interest table in named data networking. In Proceedings of the Eighth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Austin, TX, USA, 29–30 October 2012; pp. 211–222.
4. Jacobson, V.; Smetters, D.K.; Thornton, J.D.; Plass, M.; Brigs, N.; Braynard, R. Networking named content. *Commun. ACM* **2012**, *55*, 117–124. [CrossRef]
5. Wang, Y.; He, K.; Dai, H.; Meng, W.; Jiang, J.; Liu, B.; Chen, Y. Scalable name lookup in NDN using effective name component encoding. In Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems (ICDCS), Macau, China, 18–21 June 2012; pp. 688–697.

6. Saxena, D.; Raychoudhury, V. Radient: Scalable, memory efficient name lookup algorithm for named data networking. *J. Netw. Comput. Appl.* **2016**, *63*, 1–13. [CrossRef]
7. Li, F.; Chen, F.; Wu, J.; Xie, H. Longest prefix lookup in named data networking: How fast can it be? In Proceedings of the 2014 9th IEEE International Conference on Networking, Architecture, and Storage (NAS), Tianjin, China, 6–8 August 2014; pp. 186–190.
8. Zhang, L.; Afanasyev, A.; Burke, J.; Jacobson, V.; Crowley, P.; Papadopoulos, C.; Wang, L.; Zhang, B. Named data networking. *ACM SIGCOMM Comput. Commun. Rev.* **2014**, *44*, 66–73. [CrossRef]
9. Yu, W.; Pao, D. Hardware accelerator to speed up packet processing in NDN router. *Comput. Commun.* **2016**, *91*, 109–119. [CrossRef]
10. Dai, H.; Lu, J.; Wang, Y.; Liu, B. BFAST: Unified and scalable index for NDN forwarding architecture. In Proceedings of the 2015 IEEE Conference on Computer Communications (INFOCOM), San Francisco, CA, USA, 10–15 April 2015; pp. 2290–2298.
11. Wang, Y.; Zu, Y.; Zhang, T.; Peng, K.; Dong, Q.; Liu, B.; Meng, W.; Dai, H.; Tian, X.; Wu, H.; et al. Wire Speed Name Lookup: A GPU-based Approach. In Proceedings of the NSDI, Lombard, IL, USA, 2–5 April 2013; pp. 199–212.
12. Wang, Y.; Dai, H.; Jiang, J.; He, K.; Meng, W.; Liu, B. Parallel name lookup for named data networking. In Proceedings of the 2011 IEEE Global Telecommunications Conference (GLOBECOM 2011), Houston, TX, USA, 5–9 December 2011; pp. 1–5.
13. Li, Y.; Zhang, D.; Yu, X.; Liang, W.; Long, J.; Qiao, H. Accelerate NDN name lookup using FPGA: Challenges and a scalable approach. In Proceedings of the 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, Germany, 2–4 September 2014; pp. 1–4.
14. Song, T.; Yuan, H.; Crowley, P.; Zhang, B. Scalable name-based packet forwarding: From millions to billions. In Proceedings of the 2nd International Conference on Information-centric Networking, San Francisco, CA, USA, 30 September–2 October 2015; pp. 19–28.
15. Li, D.; Li, J.; Du, Z. An improved trie-based name lookup scheme for Named Data Networking. In Proceedings of the 2016 IEEE Symposium on Computers and Communication (ISCC), Messina, Italy, 27–30 June 2016; pp. 1294–1296.
16. Saxena, D.; Raychoudhury, V. N-FIB: Scalable, memory efficient name-based forwarding. *J. Netw. Comput. Appl.* **2016**, *76*, 101–109. [CrossRef]
17. Wang, Y.; Xu, B.; Tai, D.; Lu, J.; Zhang, T.; Dai, H.; Zhang, B.; Liu, B. Fast name lookup for named data networking. In Proceedings of the 2014 IEEE 22nd International Symposium of Quality of Service (IWQoS), Hong Kong, China, 26–27 May 2014; pp. 198–207.
18. Wang, Y.; Pan, T.; Mi, Z.; Dai, H.; Guo, X.; Zhang, T.; Liu, B.; Dong, Q. Namefilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters. In Proceedings of the INFOCOM, Turin, Italy, 14–19 April 2013; pp. 95–99.
19. Wang, Y.; Dai, H.; Zhang, T.; Meng, W.; Fan, J.; Liu, B. GPU-accelerated name lookup with component encoding. *Comput. Netw.* **2013**, *57*, 3165–3177. [CrossRef]
20. Fukushima, M.; Tagami, A.; Hasegawa, T. Efficiently looking up non-aggregatable name prefixes by reducing prefix seeking. In Proceedings of the 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Turin, Italy, 14–19 April 2013; pp. 340–344.
21. Yuan, H.; Crowley, P. Reliably scalable name prefix lookup. In Proceedings of the 2015 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Oakland, CA, USA, 7–8 May 2015; pp. 111–121.
22. Perino, D.; Varvello, M.; Linguaglossa, L.; Laufer, R.; Boislaigue, R. Caesar: A content router for high-speed forwarding on content names. In Proceedings of the Tenth ACM/IEEE Symposium on Architectures for Networking and Communications Systems, Los Angeles, CA, USA, 20–21 October 2014; pp. 137–148.
23. Quan, W.; Xu, C.; Guan, J.; Zhang, H.; Grieco, L.A. Scalable name lookup with adaptive prefix bloom filter for named data networking. *IEEE Commun. Lett.* **2014**, *18*, 102–105. [CrossRef]
24. Dharmapurikar, S.; Krishnamurthy, P.; Taylor, D.E. Longest prefix matching using Bloom filters. *IEEE/ACM Trans. Netw.* **2006**, *14*, 397–409. [CrossRef]
25. So, W.; Narayanan, A.; Oran, D.; Wang, Y. Toward fast NDN software forwarding lookup engine based on hash tables. In Proceedings of the 2012 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS), Austin, TX, USA, 29–30 October 2012; pp. 85–86.

26. Tan, Y.; Zhu, S. Efficient Name Lookup Scheme Based on Hash and Character Trie in Named Data Networking. In Proceedings of the 2015 12th Web Information System and Application Conference (WISA), Jinan, China, 11–13 September 2015; pp. 130–135.
27. Yanbiao, L.; Zhang, D.; Huang, K.; He, D.; Long, W. A memory-efficient parallel routing lookup model with fast updates. *Comput. Commun.* **2014**, *38*, 60–71.
28. Yuan, H.; Crowley, P.; Song, T. Enhancing Scalable Name-Based Forwarding. In Proceedings of the Symposium on Architectures for Networking and Communications Systems, Beijing, China, 18–19 May 2017; pp. 60–69.
29. Varvello, M.; Perino, D.; Linguaglossa, L. On the design and implementation of a wire-speed pending interest table. In Proceedings of the 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), Turin, Italy, 14–19 April 2013; pp. 369–374.
30. Yuan, H.; Crowley, P. Scalable pending interest table design: From principles to practice. In Proceedings of the 2014 IEEE INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2049–2057.
31. Carofiglio, G.; Gallo, M.; Muscariello, L.; Perino, D. Pending interest table sizing in named data networking. In Proceedings of the 2nd International Conference on Information-Centric Networking, San Francisco, CA, USA, 30 September–2 October 2015; pp. 49–58.
32. You, W.; Mathieu, B.; Truong, P.; Peltier, J.F.; Simon, G. Dipit: A distributed bloom-filter based pit table for ccn nodes. In Proceedings of the 2012 21st International Conference on Computer Communications and Networks (ICCCN), Munich, Germany, 30 July–2 August 2012; pp. 1–7.
33. Li, Z.; Bi, J.; Wang, S.; Jiang, X. The compression of pending interest table with bloom filter in content centric network. In Proceedings of the 7th International Conference on Future Internet Technologies, Seoul, Korea, 11–12 September 2012; p. 46.
34. Li, Z.; Liu, K.; Zhao, Y.; Ma, Y. MaPIT: An enhanced pending interest table for NDN with mapping bloom filter. *IEEE Commun. Lett.* **2014**, *18*, 1915–1918. [CrossRef]
35. Tsilopoulos, C.; Xylomenos, G.; Thomas, Y. Reducing forwarding state in content-centric networks with semi-stateless forwarding. In Proceedings of the 2014 IEEE INFOCOM, Toronto, ON, Canada, 27 April–2 May 2014; pp. 2067–2075.
36. Yuan, H.; Song, T.; Crowley, P. Scalable NDN Forwarding: Concepts, Issues and Principles. In Proceedings of the International Conference on Computer Communications and Networks, Munich, Germany, 30 July–2 August 2012; pp. 1–9.
37. So, W.; Narayanan, A.; Oran, D. Named data networking on a router: Forwarding at 20gbps and beyond. In Proceedings of the ACM SIGCOMM Computer Communication Review, Hong Kong, China, 12–16 August 2013; Volume 43, pp. 495–496.
38. Hsu, J.M.; Chang, J.Y. A CRC-32 Name Prefix Encoding in Named Data Networking. *Int. J. Adv. Inf. Technol. (IJAIT)* **2014**, *8*, 125–130.
39. Huang, S.; Xu, J.; Yang, X.; Wu, Z.; Niu, C. Steerable Name Lookup based on Classified Prefixes and Scalable One Memory Access Bloom Filter for Named Data Networking. *Int. J. Future Gen. Commun. Netw.* **2016**, *9*, 87–100. [CrossRef]
40. Quan, W.; Xu, C.; Vasilakos, A.V.; Guan, J.; Zhang, H.; Grieco, L.A. TB2F: Tree-bitmap and bloom-filter for a scalable and efficient name lookup in content-centric networking. In Proceedings of the 2014 IFIP Networking Conference, Trondheim, Norway, 2–4 June 2014; pp. 1–9.
41. Bari, M.F.; Chowdhury, S.R.; Ahmed, R.; Boutaba, R.; Mathieu, B. A Survey of Naming and Routing in Information-Centric Networks. *IEEE Commun. Mag.* **2012**, *50*, 44–53. [CrossRef]
42. Carzaniga, A.; Rutherford, M.; Wolf, A. A Routing Scheme for Content-Based Networking. In Proceedings of the 3rd Annual Joint Conference IEEE Computer and Communication Societies (INFOCOM 2004), Hong Kong, China, 7–11 March 2004; pp. 918–928.
43. Koponen, T.; Chawla, M.; Chun, B.G.; Ermolinskiy, A.; Kim, K.H.; Shenker, S.; Stoica, I. A Data-Oriented (and Beyond) Network Architecture. In Proceedings of the 2007 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, Kyoto, Japan, 27–31 August 2007; pp. 181–192.
44. Dannewitz, C.; Golic, J.; Ohlman, B.; Ahlgren, B. Secure Naming for a Network of Information. In Proceedings of the 2010 INFOCOM IEEE Conference on Computer Communications Workshops, San Diego, CA, USA, 15–19 March 2010; pp. 1–6.

45. Lagutin, D.; Visala, K.; Tarkoma, S. Publish/Subscribe for Internet: PSIRP Perspective. In *Towards the Future Internet Emerging Trends from European Research*; IOS Press: Amsterdam, The Netherlands, 2010; pp. 75–84.
46. Fagin, R.; Nievergelt, J.; Pippenger, N.; Strong, H.R. Extendible hashing—A fast access method for dynamic files. *ACM Trans. Database Syst.* **1979**, *4*, 315–344. [CrossRef]
47. Kirsch, A.; Mitzenmacher, M.; Varghese, G. Hash-based techniques for high-speed packet processing. In *Algorithms for Next Generation Networks*; Springer: London, UK, 2010; pp. 181–218.