PAPER

High-speed Design of Conflict-less Name Lookup and Efficient Selective Cache on CCN Router

Atsushi OOKA[†a)], ***Nonmember***, **Shingo ATA**[††b)], **Kazunari INOUE**[†*c)], ***Members***, ***and*** **Masayuki MURATA**[†d)], ***Fellow***

SUMMARY Content-centric networking (CCN) is an innovative network architecture that is being considered as a successor to the Internet. In recent years, CCN has received increasing attention from all over the world because its novel technologies (e.g., caching, multicast, aggregating requests) and communication based on names that act as addresses for content have the potential to resolve various problems facing the Internet. To implement these technologies, however, requires routers with performance far superior to that offered by today's Internet routers. Although many researchers have proposed various router components, such as caching and name lookup mechanisms, there are few router-level designs incorporating all the necessary components. The design and evaluation of a complete router is the primary contribution of this paper. We provide a concrete hardware design for a router model that uses three basic tables —forwarding information base (FIB), pending interest table (PIT), and content store (CS)—and incorporates two entities that we propose. One of these entities is the name lookup entity, which looks up a name address within a few cycles from content-addressable memory by use of a Bloom filter; the other is the interest count entity, which counts interest packets that require certain content and selects content worth caching. Our contributions are (1) presenting a proper algorithm for looking up and matching name addresses in CCN communication, (2) proposing a method to process CCN packets in a way that achieves high throughput and very low latency, and (3) demonstrating feasible performance and cost on the basis of a concrete hardware design using distributed content-addressable memory.

key words: *Future Networks, Content-centric Networking, Architecture, Router Hardware, Content-addressable Memory, Bloom Filter*

1. Introduction

1.1 Background

The Internet, which is now a global network of networks, is used in a form and on a scale considerably different than the network envisaged at the time that the original design principles and assumptions were decided, and this has given rise to many problems. The initial implementation of the Internet was developed to provide the ability to communicate within pairs of hosts. At present, however, the Internet is used for the purpose of distributing and retrieving various types of content, with this content going to and coming from global networks. This is quite different from communicating with a specific host. Nevertheless, Internet protocol (IP) datagrams require that an IP address be assigned, which specifies the network interface. In addition, there is no guarantee that the location of the requested data will be constant, because the data may be moved or deleted or the server providing that data may become temporarily inaccessible.

Information-centric networking (ICN), also known as content-centric networking (CCN) [1], has been proposed as a measure for overcoming the limitations of the current Internet architecture. The most significant feature in ICN/CCN is that a "name" address, which is variable-length and human-readable in a way similar to uniform resource locators (URLs), is assigned to each piece of content. Using the name address, content distribution applications (e.g., YouTube and Twitter), which have been becoming more and more popular, can be supported efficiently and securely by adhering to the end-to-end principle. The fundamental motivation for introducing ICN/CCN is to realize a content delivery channel that requires only directly specifying the name of the desired content at the network layer. An additional benefit of such a system would be that in-network caching could relax both the spatial and temporal constraints on communications that are present in the current Internet, which would significantly improve the flexibility of the placement of contents, efficiency of network resource usage, and the end-users quality of experience (e.g., content availability and response time). Against this background, ICN/CCN has proved increasingly attractive in recent years, and many researchers are involved in study and development of this area (see [2] and references therein).

Obviously, many challenges must be resolved to realize CCN, which would be a "clean-slate" network (i.e., a replacement rather than an incremental improvement). First, we need new name resolution and routing mechanisms that are based on the name addresses used in CCN. Second, the "bread crumb" forwarding technique, which uses Interest and Data packets and which naturally incorporates multicast and Interest aggregation into the network, requires lookup tables that can be updated much more quickly than IP address tables. Most research focuses on in-network caching mechanisms because they can cache content more efficiently and thus require fewer resources [3], [4]. In addition, there are a number of problems that have been analyzed and eval-

Manuscript received July 1, 2014.
Manuscript revised December 2, 2014.

[†]The authors are with the Graduate School of Information Science and Technology, Osaka University, Suita-shi, Osaka 565–0871, Japan
[††]The author is with the Graduate School of Engineering, Osaka City University, Sumiyoshi-ku, Osaka-shi, Osaka 558–8585, Japan
[*]Presently, the author is with the Nara National College of Technology, Yamatokoriyama-shi, Nara 639–1058, Japan
a) E-mail: a-ooka@ist.osaka-u.ac.jp
b) E-mail: ata@info.eng.osaka-cu.ac.jp
c) E-mail: inoue.kazunari@ist.osaka-u.ac.jp
d) E-mail: murata@ist.osaka-u.ac.jp

DOI: 10.1587/transcom.E0.B.1

uated: security, mobility, and CCN deployment, among others [5]–[7].

1.2 Related Work

A number of research projects have studied ICN/CCN approaches, such as CCNx [8], NDN [5], PURSUIT [9], and SAIL [10]. These all share the common concept of addressing the content that is to be exchanged in the communication by a "name", which is mnemonic and unique to each chunk of data. They also try to natively implement functions such as in-network caching, multicasting, and built-in security for data. In this paper, we focus on CCN/NDN, which is characterized by a hierarchical structure and variable-length names.

Most previous studies have focused on isolated components or techniques at the router level, such as caching and name lookup mechanisms. For example, Caesar [11] aims to implement a scalable high-speed forwarding table. DiPIT [12] and NameFilter [13] focus on pending interest tables (PITs) and propose a very fast inexpensive architecture consisting of two-level Bloom filters, but the probabilistic nature of that model means that false positives can never be completely eliminated. NCE [14], ENPT [15] and ATA(MATA) [16] approach highly memory-efficient name lookup mechanisms by using a trie-like structure. MATA achieves line speed (i.e., near-real-time performance) by means of a highly parallelized architecture using graphics processing units, although it is difficult to reduce the latency.

Among the existing complete router designs [4], [17], content-addressable memory (CAM), which has the potential to become a major lookup technology, has not been sufficiently researched because of its cost. Nevertheless, the estimations conclude that it would be impracticably difficult to support an Internet-scale CCN deployment using CAM, although the analysis indicates that at the content-delivery network or Internet service provider scale, it could be easily afforded and would make a significant contribution to investigating the feasibility of ICN/CCN. In addition, the designs and simulations of CCN routers shown in the existing studies are not based on hardware designs or implementations. We plan to eventually demonstrate a router design and hardware architecture with the same level of concreteness as that in [18]. In [18], the implementation of reconfigurable match tables for software-defined networking (SDN) is proposed and a detailed design and evaluation are described for a hardware implementation. The proposed techniques for quickly matching a number of entries in SDN could help to implement feasible matching mechanisms for ICN/CCN, but the techniques that would be useful in SDN cannot be directly applied to the router for use in CCN because of the variable-length name addresses allowed by CCN.

1.3 Objectives

We address one of the biggest challenges to implementing a CCN router to demonstrate the feasibility and specific performance of a CCN router. Of course, the hardware must be sufficiently powerful to realize CCN communications. The realistic performance of a hardware router is required to estimate performance at the network level and evaluate whether various proposals for CCN are reasonable. However, there are few studies offering a comprehensive design for a CCN router; instead, most previous studies have focused on isolated components or techniques at the router level, such as caching and name lookup mechanisms. We investigated a design for a CCN router in [19] and evaluated its performance; however, the discussion about the design did not completely elucidate the available algorithms for lookup. In this paper, we described such algorithms in detail. Furthermore, we evaluate the scalability of a distributed-and-load-balancing Bloom filter (DLB-BF) [20], which could increase the utility of memory space when implemented in our previous work [19].

In this paper, we propose a complete CCN router design that can be implemented on existing hardware and show the feasibility and performance of the router. In Section 2, we describe an accurate communication model for CCN that properly handles all packets. In Section 3, customizing the router architecture by using the name lookup entity (NLE) and the interest count entity (ICE) constructs is proposed, and the hardware design using CAM and a Bloom filter is demonstrated in Section 4. We design a feasible hardware architecture by means of dividing CAM into smaller parts. In Section 5, we comprehensively evaluate the throughput, size of memory, and cost of the CCN router. Finally, we give a conclusion and discuss areas for future research.

2. CCN

2.1 Principles of CCN

CCN is a novel network architecture that was designed with a focus on the content of data, rather than on the location of that data. This approach has the following advantages.

- Content-centric, rather than host-centric, communication
- *Names* that provide each chunk of data with unique, human-readable, and hierarchical addresses
- Mechanisms to natively support multicasting, in-network caching, and built-in security for data

The content-centric design was inspired by recent developments in the utilization of the Internet. A main use of current networks is the distribution and retrieval of vast amounts of data, such as HTML documents, images, and high-definition video. Specifying the locations of providers and consumers is not necessary for this purpose. However, the protocol for Internet addressing, which is the dominant tool for networked communication, imposes these kinds of unnecessary processes. In addition, data sharing systems that are unaware of network structure and packet content, such as CDNs and peer-to-peer (P2P) applications are costly

and inefficient. CCN is a solution for dealing with these incompatibilities and security concerns by shifting the routing behavior from focusing on "where" to focusing on "what".

The concept of *name*, with each chunk of data assigned a name, plays a major role in CCN as a replacement for the IP addresses that are presently assigned to device interfaces. In CCN, it is not necessary to know the location of the device that we want to communicate with; instead, we identify the name of the data chunk that we want. Names enable us to do this by providing each chunk of data with a unique, human-readable, and hierarchical address. They allow those who use networks and develop network applications to eliminate the complexity of identifying hosts and to directly specify the identifier of the content. For example, a picture of an apple produced by XYZ could be named "`/XYZ/pictures/apple.jpg`." The name could also contain the version and segment number of data to permit versioning and segmentation. For example "`/XYZ/pictures/apple.jpg/v1/s2`" could indicate the second chunk of version 1 of the image.

2.1.1 Communication Model

CCN's communication model is request-driven through the exchange of *Interest* packets and *Data* packets (abbreviated to Interest and Data below). To begin, a data consumer requests content by sending an Interest, which contain the name of the content. In response to the Interest, the content provider sends Data, which contain the actual data. Finally, the consumer receives all the Data and the request is satisfied.

The name written in an Interest request may be just a prefix of the requested content. For example, when a consumer requests a video named "`/video/a.mpg`", the producer may send the Data with the name "`/video/a.mpg/v1/s1`", so that the name contains the version and segment number of the data. This dynamic naming method is referred to as *active naming* in this paper. In addition, we must consider the case where a name and its prefix (e.g., "`/video/a.mpg`" and "`/video/a.mpg/v1/s1`") refer to different content. In this paper, we call pairs with this relationship *name siblings*. Name siblings complicate the handling of name addresses in a router, but there is no inherent reason to forbid the use of name siblings by applications running on CCN. It is not a foregone conclusion that name siblings will be accepted for CCN communications, but we include name siblings in our discussion here.

2.1.2 Router Behavior

Although a number of research projects approach ICN/CCN in a different way, we adopt the design principles of Named Data Networking (NDN) [5] in this paper. To implement forwarding functions in a CCN that includes multicasting, caching, and a loop-free architecture, the CCN router contains three data structures: a forwarding information base (FIB), a PIT, and a content store (CS). The FIB is a table used for determining the proper interface for forwarding Interest requests that have arrived at the router. The PIT remembers the interfaces from which these requests have arrived so that it can send back the matching Data that will be subsequently received by the router. Interest requests that specify duplicate names (i.e., that have already been recorded in the PIT) leave only the trace of the route and forwarding is skipped so as to aggregate requests and realize multicasting and a loop-free architecture. The CS serves as a cache for Data. Because identical Data are addressed by identical names, cached Data can be reused independently of the requester and time.

2.2 Name Lookup Algorithm

To demonstrate the utility and limitations of advanced functions in CCN, we consider all possibilities for name lookup algorithms that could be used on a CCN router, describing them in this subsection. An algorithm for implementing lookup tables in a CCN router is not trivial, and to the best of our knowledge has never been discussed. The tables contained by the router (i.e., FIB, PIT, and CS) are not simple hash tables with uniquely keyed entries; instead, a single retrieval key could match multiple entries in the table because of prefix matching and active naming. We need to consider how to match entries in the tables and select one of them so that packets are appropriately processed without conflict between the matching policies and implementations. The Interests and Data must be looked up in the FIB, PIT, and CS tables. There are five possible combinations (rather than six, because Data are not looked up in the FIB table).

2.2.1 Matching and Selecting Algorithms

A matching algorithm is an algorithm to decide whether the search key (denoted by K_S) matches the key stored in the table entry (denoted by K_E), and we must consider the case where a given key matches the prefix of another key K (denoted by $P(K)$). The following four matching algorithms are available:

- *Exact Match (EM)*, which matches when $K_S = K_E$;
- *Search-key Prefix Match (SPM)*, which matches when $P(K_S) = K_E$;
- *Entry-key Prefix Match (EPM)*, which matches when $K_S = P(K_E)$; and
- *Both-keys Prefix Match (BPM)*, which matches when $K_S = K_E$or when one is identical to the prefix of the other (*not* $P(K_S) = P(K_E)$, that is, when both keys have the same prefix).

EM is used for strict matching that disallows any ambiguities, such as for determining the existence of a prefix aggregation. SPM is a familiar matching algorithm that is employed in current IP routers as a longest-prefix-matching (LPM) algorithm, although LPM is not the only possibility for selecting from among distinct Data found by SPM. EPM and BPM have not been used in IP routers; however, these

matching algorithms should be taken into account because they potentially allow CCN routers to take advantage of active naming.

The non-exact matching algorithms might retrieve multiple entries, and so algorithms for choosing one of the retrieved entries should be described. Such algorithms are called selecting algorithms. When the matching algorithm is SPM, selecting the longest entry is suitable; this is just the LPM algorithm that is used in conventional IP routers. Selections from results provided by EPM and BPM are more complex. For example, when K_S is "`/video/A.mpg`", the K_S will match both "`/video/a.mpg/v1/s1`" and "`/video/a.mpg/v2/s4`". Since LPM cannot deterministically select only one of the entries when those entries are same the length, other criteria for use in selecting algorithms are needed. One strategy is to prioritize the time when the entries are registered or the number of requests. We can also adopt a simpler strategy when matching from FIB: select all matching entries. In that case, Interest requests are multicast from all ports corresponding to the matched entries, although this might generate excessive traffic. Richer strategies would enable a CCN router to use known preferences or the information on the publisher, scope, and other attributes (such as in the ChildSelector [1], [21] framework) can be used if the CCN router is powerful enough to employ those strategies.

2.2.2 Algorithms Suitable for Each Table

The combination of SPM and LPM is the most suitable for FIB. As with the strategy for current IP routers, a name address is hierarchically structured so that multiple name addresses having the same prefix can be aggregated into one entry. The other algorithms cannot aggregate entries.

When looking up Data in CS, EM should be used because the name assigned to the Data must not be an abbreviated active name and must, instead, be a complete name that identifies specific content. Additionally, the other algorithms do not support name siblings. For the process to look up Data in CS it is essential to avoid duplicate entries, but it is possible to skip this process when the Data is so unpopular that PIT does not have any matching entries.

When looking up an Interest in CS, either EM or EPM should be used because it would be undesirable for an Interest to match a Data or cache entry with a name shorter than the one specified in the Interest request. Thus, although SPM and BPM, which allow K_S to match a shorter K_E in CS, are unsuitable, EM and EPM cause no problems. We note that EPM requires that the priority rules select exactly one entry when a single K_S matches multiple K_E. If name siblings are allowed, then EM must be used in order to prevent a router from returning undesirable Data in response to the Interest request. Otherwise, EPM could improve the cache hit rate by exploiting an advantage of active naming. This advantage occurs in cases such as when there is an entry K_E^1 = "`/video/a.mpg/v1/s1`" in CS. When searching for an Interest named K_S^2 = "`/video/a.mpg`", K_E^1 can match K_S^2 by using EPM.

For looking up Data in PIT, SPM should be chosen: Active naming and name siblings cannot be supported by the other matching algorithms. For the selecting algorithm, we can select both the entry with the longest key and all entries that match the search key. While using LPM is a risk-free approach, it is more efficient to satisfy multiple Interests at once if name siblings are disallowed. As an example, suppose there are the entries K_E^1 = "`/video/a.mpg/v1/s1`", K_E^2 = "`/video/a.mpg/v2/s6`" and K_E^3 = "`/video/a.mpg`" in PIT. When the Data named K_S = "`/video/a.mpg/v1/s1`" arrived at the router, the Data were able to satisfy not only K_E^1 but also K_E^3 by selecting all matching entries. Needless to say, because an Interest request for content named "`/video/a.mpg`" will not be satisfied by content named "`/video/a.mpg/v1/s1`", the all-hits algorithm cannot work when name siblings are allowed.

When looking up an Interest in PIT, both EM and BPM are more suitable matching algorithms than the others. To see why, consider two cases: (1) PIT contains K_E^1 = "`/text/A.txt/v1/s1`" and K_E^2 = "`/text/A.txt/v2/s6`", and an Interest whose name is K_S^1 = "`/text/A.txt`" is looked up; and (2) PIT contains K_E^3 = "`/text/A.txt`", and an Interest whose name is K_S^2 = "`/text/A.txt/v1/s1`" is looked up.

First, an example of using EM is shown in Figure 1. In both *Case*(1) and *Case*(2), Interest matches none of the entries and is registered as a new entry. Thus, EM cannot aggregate Interests whose names are identical to the prefixes of entries in PIT, but EM is the only solution that can handle name siblings.

In contrast, BPM makes full use of active naming, as shown in Figure 2. Using BPM, Interests with names shorter than K_E are aggregated into the entries, as in *Case*(1). Of course, BPM also requires priority rules to select a single entry. In *Case*(2), K_E^3 is shorter than the Interest's K_S^2. However, K_S^2 cannot be aggregated to K_E^3 since the shorter key may match Data whose name is different from K_S^2 (e.g., "`/text/A.txt/v2/s6`") and the original Interest will be unsatisfied. Therefore, any matching entry with a shorter key is re-registered as a new entry with the longer key assigned to the Interest. It should be noted that this re-registering process takes advantage of active naming but disrupts the use of name siblings by eliminating the Interests with shorter names.

Finally, SPM and EPM are unsuitable for use in this context. Compared to BPM, these two matching algorithms cannot take full advantage of active naming; SPM cannot aggregate entries in *Case*(1), and EPM cannot aggregate entries in *Case*(2).

Table 1 summarizes the available algorithms for matching and selecting the entry in cases other than looking up an Interest in FIB or looking up Data in CS. The combination of SPM and LPM is used for FIB, and EM is used for looking up Data in CS. Only combination (I) is able to cope with name siblings, although all combinations support active naming.

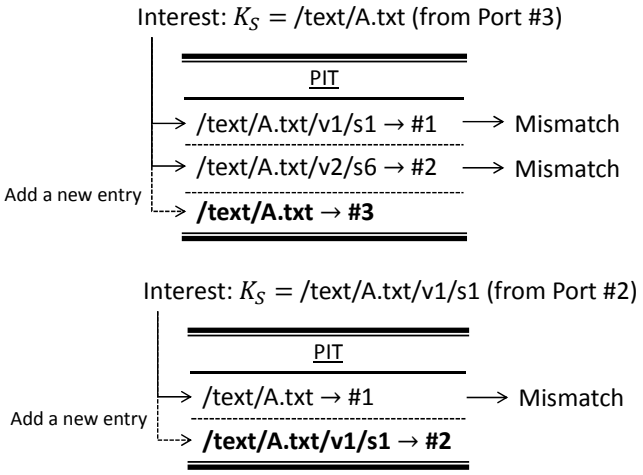

Fig. 1 Example of Using EM (upper: *Case*(1), lower: *Case*(2))

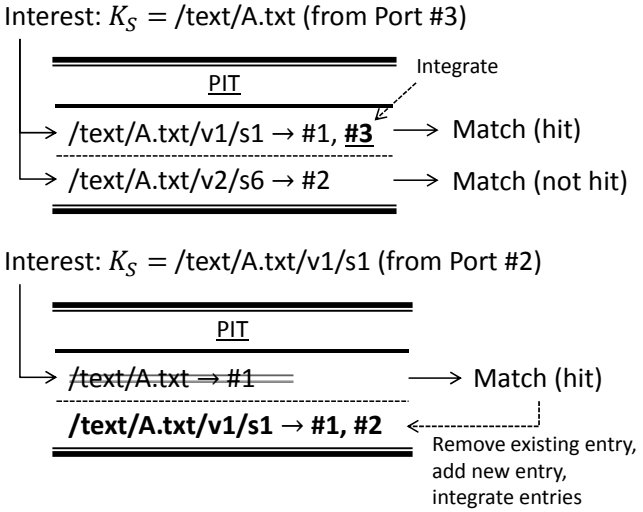

Fig. 2 Example of Using BPM (upper: *Case*(1), lower: *Case*(2))

2.2.3 How to Realize CCN Capabilities

We note first that the specification of CCN is still in progress and may be changed significantly in the future. One possible way to implement a CCN router at this stage is to enumerate its possible capabilities by considering use cases of CCN, and use this enumeration to figure out what kind of functionalities would be needed in a CCN router. Active naming and name siblings are capabilities that would make CCN more flexible and useful [1], [5], although the current CCNx v1.0 specification does not include them [21]. One of the aims of our paper is to represent the capabilities needed for CCN by a combination of algorithms and data structures implemented in CCN routers. We focus on active naming and name siblings in this paper because of their influence on efficiency, management cost, and constraints of the CCN architecture.

Active naming, which is supported by all combinations shown in Table 1, allows a consumer to request content by specifying incomplete names. Suppose there is a consumer who does not know the name of the latest version of the A.txt file; however, the name of the Data packets that the user wants to retrieve requires specifying a version of the file and the order of the segments (e.g. the *M*th segment of the *N*th version of A.txt can be named "`/text/A.txt/vN/sM`"). By using active naming, the consumer can send an Interest request for "`/text/A.txt`" and receive the Data named "`/text/A.txt/v2/s1`", whose complete name indicates that it is the first chunk of version 2 of the file. In addition to its usefulness for retrieving content from a partial name, active naming can be used for applications that dynamically generate content.

In terms of the CCN router behavior, the complexity and efficiency of the operations to handle active naming vary according to the combinations of matching and selecting algorithms. The combinations (II)–(IV) shown in Table 1 require complex selecting algorithms for selecting the optimal entry to increase the cache hit rate of active naming.

For example, consider what happens when an Interest request for "`/text/A.txt`" arrives at a router containing cached copies of "`/text/A.txt/v1/s1`" and "`/text/A.txt/v1/s2`". If the router employs the combination (III) or (IV), then the Interest can be satisfied by the router instead of the content producer although the Interest matches both entries, and so the router needs to know how to decide which cached Data to choose. In this case, "`/text/A.txt`" may be taken as a request for "`/text/A.txt/v1/s1`", which is the first chunk of A.txt. The decision may require all CCN routers to be familiar with the application-specific naming conventions and priorities, or require Interest requests that rely on active naming to contain additional information to identify the requested Data, such as by using a selector [1]. However, in the more complex case, where the Data for "`/text/A.txt/v1/s1`" are not cached in the router, the router will reply with undesired Data, such as "`/text/A.txt/v1/s2`", because the decision is based on the limited information available to the router. Thus, methods to avoid such errors are necessary (see Exclude [21] for an example).

There are similar problems with Interest aggregation in PIT. For instance, if a consumer uses active naming to request the latest version of A.txt and an Interest request for "`/text/A.txt`" arrives at a certain router simultaneously with another Interest request for "`/text/A.txt/v1/s1`", then the latest version of A.txt will be v2, but these two requests must be distinguished. However, both will be treated as the same request, for "`/text/A.txt/v1/s1`", by a router employing the combination (II) or (IV) unless information about versions is known in advance. To prevent problems resulting from not knowing the consumer's intention, a CCN architecture based on (II) or (IV) needs the additional mechanisms discussed above.

In contrast to the combinations (II), (III), and (IV), the combination (I) makes the selecting algorithms simple. In (I), it is unnecessary to avoid the conflicts between complete names and the ambiguous names produced by active

naming, although this comes at a cost: Interests using active naming cannot be aggregated nor satisfied by intervening routers. In the above examples, an Interest request for "`/text/A.txt`" will fail to match the cache entries in CS and the unsatisfied requests in PIT; after that, the producer will directly satisfy the Interest according to the latest state of the content or the application.

Another important point to consider is the management of name siblings. The use of name siblings will cause undesired aggregation of requests and cache hits in the combinations (II)–(IV). To illustrate the problem, consider content named "`/text/A.txt`" that is different from content whose name is a strict extension of the original name, such as "`/text/A.txt/v1/s1`". If an Interest request for "`/text/A.txt`" matches an entry whose key is "`/text/A.txt/v1/s1`" in PIT or CS, then the consumer who sent the Interest request will ultimately receive the Data for "`/text/A.txt/v1/s1`", which is not the requested content. Therefore, name siblings cannot be permitted in the CCN architecture if any combination other than (I) is adopted. Furthermore, to prevent name siblings from being created by application errors or malicious attacks, dynamic mechanisms to eliminate inappropriate name siblings must be incorporated into the functions of CCN. In contrast, the combination (I) is free from cost of such management and still leaves the ability to creatively apply name siblings; for example, a shorter name can be used for content that contains the information needed to know the longer name and request its Data, such as the latest version and the number of segments.

In summary, the combination (I) should be adopted when avoiding the use of complex selecting algorithms for active naming and the costly management of name siblings is desired. In contrast, any of the combinations (II)–(IV) should be selected when efficient handling of packets using active naming is desired.

3. Architecture

Here, we describe the design principles of our proposed router architecture. In comparison with fixed-length IP addresses, variable-length name addresses impose a very high load for lookup. To handle variable-length name addresses at line speed, we introduce CAM and a distributed-and-load-balancing Bloom filter (DLB-BF) [20] into the prefix table; an associated element is the name lookup entity (NLE). An NLE provides a map between name addresses and entries in each of the three tables so that only one lookup is required to deterministically retrieve the most specific entry from among the three tables . DLB-BF, which allows name lookups to be performed in parallel, reduces the workload on the CAM. We also present the interest count entity (ICE), which is a new mechanism for identifying content worth caching.

The most suitable combination of matching and selecting algorithms for the lookup mechanism using the CAM and Bloom Filter is combination (I) in Table 1. Table 2 shows the specific algorithms adopted in our implementation. If name siblings are disallowed (and therefore it is possible to use SPM as the matching algorithm), the combination (I) makes the lookup mechanism simple by choosing SPM for all matching algorithms except for the algorithm for the lookup of Interests in PIT, as shown in Table 2. For this reason, we assume that name siblings are not used. Additionally, Binary-CAM (BCAM) can be used instead of Ternary-CAM (TCAM); BCAM is more reasonable than TCAM with respect to cost and power.

Table 2 Name Lookup Algorithm Applied to Our Implementation

Packet	Storage	Matching Algorithm	Selecting Algorithm
Interest	FIB	SPM	Longest Match
	PIT	EM	-
	CS	EM [1]	- [2]
Data	PIT	SPM	Longest Match
	CS	EM [1]	- [2]

[1] This algorithm can be implemented as SPM.
[2] This algorithm can be implemented as LPM.

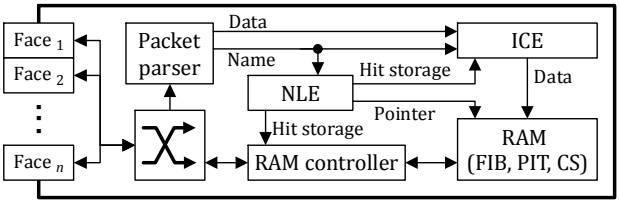

Fig. 3 CCN Router Architecture

Figure 3 illustrates the basic architecture of the proposed CCN router. First, an input packet is received by a Face element. After the packet is processed by the parser, its name and content are sent to an NLE and an ICE, respectively. NLE performs a lookup on the name and retrieves a pointer to a location in random access memory (RAM). ICE is used to avoid caching rarely requested data by counting how many Interest requests were made for the data. According to the results, an appropriate process, such as forwarding or caching, is determined. Finally, if the packet is to be forwarded, it is passed to an appropriate output face.

3.1 Name Lookup Entity

We propose NLE, which implements a fast lookup operation for name addresses. Almost all existing architectures that use a hash table sometimes yield a false positive, which results in a failure to forward packets. Preventing false positives in a hash table incurs a long delay to check that no component of the searched name is falsely matched. Our approach avoids this issue by using CAM instead of a hash table. CAM can search its entire memory in a single lookup, but the cost and power requirements have been assumed to be prohibitive. We therefore propose a solution that splits the CAM into many small parts; this is expected to be less expensive than a single large memory solution. In addition, DLB-BF can dramatically reduce the load on CAM without

Table 1 Summary of Matching and Selecting Algorithms

	CS-Interest	PIT-Data	PIT–Interest	Active naming	Name siblings
(I)	EM/-	SPM/LPM, FIFO[1], or all-hits	EM/-	Available	Available
(II)	EM/-	SPM/-	BPM/optimal	Available	Unavailable
(III)	EPM/optimal	SPM/LPM, FIFO, or all-hits	EM/-	Available	Unavailable
(IV)	EPM/optimal	SPM/-	BPM/optimal	Available	Unavailable

[1] FIFO: first in, first out

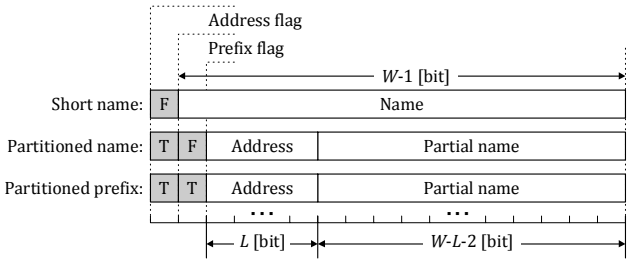

Fig. 4 Definition of CAM Entry

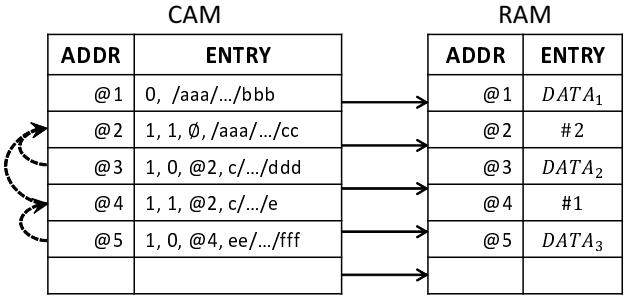

Fig. 5 Example of CAM and RAM Entries in NLE

sacrificing speed.

Because CAM stores fixed-length data words and name addresses are variable length, we must decide what to do when a name address is longer than the data word size. We divide such a name address into partial names and then simulate a hierarchical tree structure. We define three types of node: short name (SN), partitioned name (PN), and partitioned prefix (PP). An SN is used whenever the name is short enough to be store in a single data word; PN and PP are used otherwise. In terms of a tree structure, PN represents a leaf node and PP represents an internal node.

Figure 4 illustrates the definitions of fields of the node in the tree structure (i.e., the entry stored in CAM). W[bit] is the bitlength of CAM entries, and L[bit] is the bitlength of CAM addresses. "Address Flag" is set to 'TRUE(T)' in PN and PP, which use the "Address" field to store a link to the parent node. If "Prefix Flag" is true, this entry is PP, which is not a terminal node.

An example of several entries stored in CAM and RAM is shown in Figure 5. There are three names in NLE: N^A ="`/aaa/.../bbb`", N^B ="`/aaa/.../ccc/.../ddd`", and N^C ="`/aaa/.../ccc/.../eee/.../fff`". N^A is short enough to store in CAM as SN, while N^B and N^C are divided into two entries (N_1^B, N_2^B) and three entries (N_1^C, N_2^C, N_3^C), respectively. These divided entries are all of the same length: $W - L - 2$[bit], as shown in Figure 4. The values of an entry in CAM correspond to the definition in Figure 4.

N^A is stored in CAM and RAM as an SN (a single entry), and so we need only the name address to retrieve the data from RAM. The process to retrieve the data corresponding to N^B, which is too long to pack into SN, is as follows: a) divide N^B into N_1^B"`/aaa/.../cc`" and N_2^B ="`c/.../ddd`", where N_1^B and N_2^B = are used to search PP and PN, respectively; b) perform a lookup for N_1^B as PP with the Address field set to 0 (because the tree structure starts at this PP); c) create a search key as a PN from the name N_2^B and the address of the parent node N_1^B; and d) retrieve the data from RAM located at the address '@3', which was specified by searching the PN. Note that a lookup for a PN or PP entry requires the address of the parent PP. The lookup for N^C is performed in a similar way, although it requires one more PP lookup.

Since N^B and N^C share the prefix of name "`/aaa/.../cc`", the two entries for N_2^B and N_2^C, which are located at '@3' and '@4', respectively, assign the same value '@2' to the PP. The number of child nodes that have a references to this PP is held at the entry located at '@2' in RAM; we find the value '#2' there. This value is incremented whenever a new child node is registered and decremented whenever a child node is removed, allowing us to remove the PP entry when the count becomes zero.

It is worth noting that it is rare to perform multiple lookups for a long name like N^B and N^C, and doing so causes high latency. In accordance with the fact that 99% of domain names are no longer than 40 bytes [14], almost all name addresses can be stored in SN by setting $W = 320$. Entries of this length are supported in an existing CAM implementation.

3.2 Interest Count Entity

ICE is an entity used to avoid caching rarely requested data by counting Interest requests for the data. In general, the popularity of Internet traffic approximately follows Zipf's law. This means that a small amount of popular content accounts for the majority of requests. To exploit this characteristic, we propose ICE as a means of caching based on the number of requests for each piece of content. ICE counts the requests for each name, and only those Data whose frequency exceeds a certain threshold are cached. Thus, ICE prevents content that is requested once or just a few times

from occupying limited cache space. Algorithms 1 and 2 show pseudocode of these ICE processes.

Algorithm 1 Interest Process in ICE

```
1: procedure InterestProcessInICE(hitCS, name)
2:     if hitCS = True then
3:         return
4:     end if
5:     entry ← ICETable[H(name)]
6:     c ← entry.count
7:     n ← entry.name
8:     if n = name & c > 0 then
9:         entry.count++            ▷ The existing counter is incremented.
10:    else
11:        entry.name ← name        ▷ A new entry is created (or overwritten).
12:        entry.count ← 1
13:    end if
14: end procedure
```

Algorithm 2 Data Process in ICE

```
1: procedure DataProcessInICE(hitCS, name, data)
2:     if hitCS = True then
3:         return
4:     end if
5:     entry ← ICETable[H(name)]
6:     c ← entry.count
7:     n ← entry.name
8:     if c > THRESHOLD & n = name then
9:         AddCS(name, data)        ▷ The Data item is cached.
10:    end if
11: end procedure
```

4. Hardware Design

4.1 Name Lookup Entity

We now describe a detailed implementation of NLE. Figure 6 shows the hardware design of NLE. Roughly, NLE consists of four components: the unit to process *partial names* (upper left), the unit to process *partial prefixes* (lower left), DLB-BF (lower right), and CAM (upper right).

Name lookup is performed as follows. First, the input name is partitioned into fixed-length partial names if necessary. A partial name is a W_2-bits-wide segment of a name that is too long to store in a single entry (as SN in Figure 4). Since looking up a child node requires the address of its parent node, as discussed in 3.1, a buffer for partial names contains not only its string but also an address for the partial name. Second, a partial name and SN are further split into partial prefixes delimited by the character '/'. A buffer for partial prefixes both stores the prefixes and remembers the indexes of the partial names to which the partial prefixes correspond. Third, queries in DLB-BF for the partial prefixes are executed in parallel; the CAMs then search the partial prefixes for which a membership query to the DLB-BF yields true. Finally, a pointer is obtained from the resulting address and used to retrieve the data from RAM.

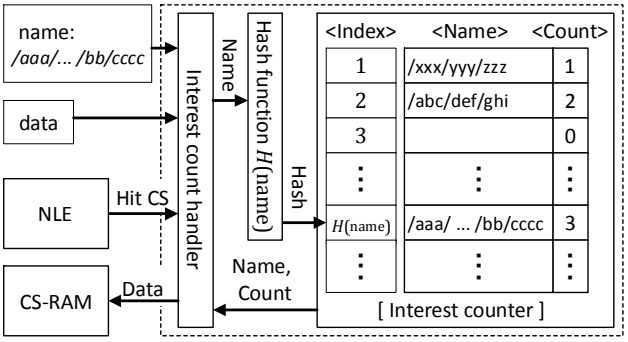

Fig. 7 Hardware Design of ICE

To reduce the cost and power requirements of the system, we split the monolithic CAM into D smaller CAMs. In general, the price of large memory is higher than the price of the same amount of memory in smaller pieces. The power required to search CAM is proportional to the size of the CAM. Thus, many small CAMs will have lower power requirements than a single large CAM. Additionally, the distributed CAMs make it easier to perform a lookup operations in parallel, which improves throughput significantly.

W is the length of a CAM entry, and W_1 and W_2 are determined according to W: W_1 is the maximum length of "Name" defined in Figure 4, and W_2 is the maximum length of "Partial Name" defined in the same place. Two conflicting characteristics are desirable for W. It should be large enough to avoid CAM lookups by PP and achieve a single CAM lookup; however, large values of W cause wasted space from storing short variable-length names into fixed-length CAM entries. We are going to investigate and optimize W in light of this tradeoff.

4.2 Interest Count Entity

The hardware design of ICE is illustrated in Figure 7. ICE is implemented as a simple hash table with name addresses as key is a name address and the counts of Interest as values. Because hash collisions may occur, ICE also holds the complete name address.

We now present the caching algorithm with ICE. When receiving an Interest, an entry containing a count of requests for content with the specified name is retrieved by using the name of the input Interest. If the count is zero or the name stored in the counter is different from the input name, the existing entry is overwritten with the new name, and the count value is reset to one. Otherwise, the count value is incremented. When Data that have not yet been cached arrive at the router, the data are cached in CS when the count value is larger than a certain threshold.

ICE makes it possible to cache only content for which caching will improve performance. Since most content is rarely requested, the limited resources of CS and CAM would be exhausted by simple caching. In contrast, the method of caching content that has a number of requests

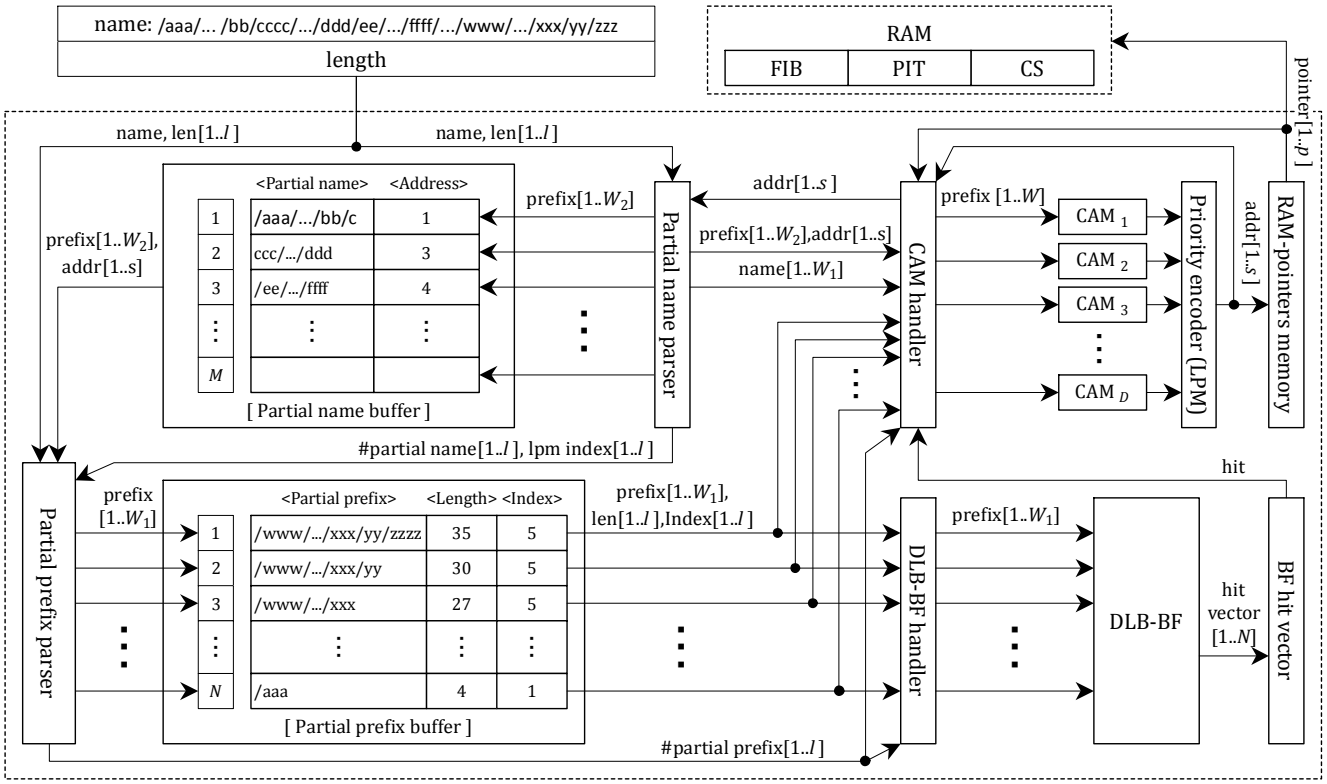

Fig. 6 Hardware Design of NLE

more than a certain threshold can be much more memory efficient. According to [22], ICE needs approximately one tenth the capacity of a universal cache.

Additionally, we can dynamically adjust the threshold according to network traffic volume. The number of requests for even unpopular content can be greater than a few if heavy traffic is handled. Furthermore, network traffic can vary hourly and daily. For these reasons, a fixed threshold is not ideal; however, a variable threshold can be used to maintain a desired cache hit ratio by adjusting the threshold in response to volume or characteristics of network traffic. The adjustment process is challenging because the first few times that content is requested, the returned data will not be cached but only counted. A method to determine suitable thresholds is left to future work.

4.3 Summary

Packet processes in our proposed CCN router processes are summarized in Figure 8. All packets received by the router are transmitted, integrated, cached, or satisfied with cached Data, with the action decided according to the flow chart.

NLE consisting of DLB-BF and CAM implements a lookup system based on LPM. The case where Interest matches PIT, however, requires checking whether the match is an EM since matching and selecting algorithms follow the strategy illustrated in Table 2. The check can be simply implemented in the obvious way: we need know only the indexes of both buffers (i.e., a buffer for partial names and a buffer for partial prefixes) to know where the matching prefix is stored. If the prefix's indexes in both buffers are 1, the matching partial prefix is essentially identical to its complete name, and the match is identified as an EM. Otherwise, the match is a non-exact match; therefore, the processes for partial prefixes should be repeated to continue.

When a Data item matches an entry in CS, the process does not stop but instead continues to run, as shown in Figure 8. Although the match appears to be proof that the Data item is cached in CS and that the Interest requesting the Data has been satisfied by the cache, active naming requires additional verification processes. For example, if CS contains a cache of Data named "`/video/a.mpg/v1/s1`", an Interest named "`/video/a.mpg`" will not match the cache entry because the Data's name is not identical to any prefixes of the Interest's name, and the Interest is expected to be satisfied with the returned Data. If the returned Data is named "`/video/a.mpg/v1/s1`", which is identical to the name of the cached content, the Data matches the cache entry. Without continuing the process in this case, the Interest named "`/video/a.mpg`" cannot be satisfied.

5. Evaluation

In this section, we analyze the performance of our CAM-based CCN router and discuss its feasibility and challenges to widespread adoption. We calculate the required memory size, cost of the memory, and throughput on the assumption of a table with 10 million entries, average packet size of 256

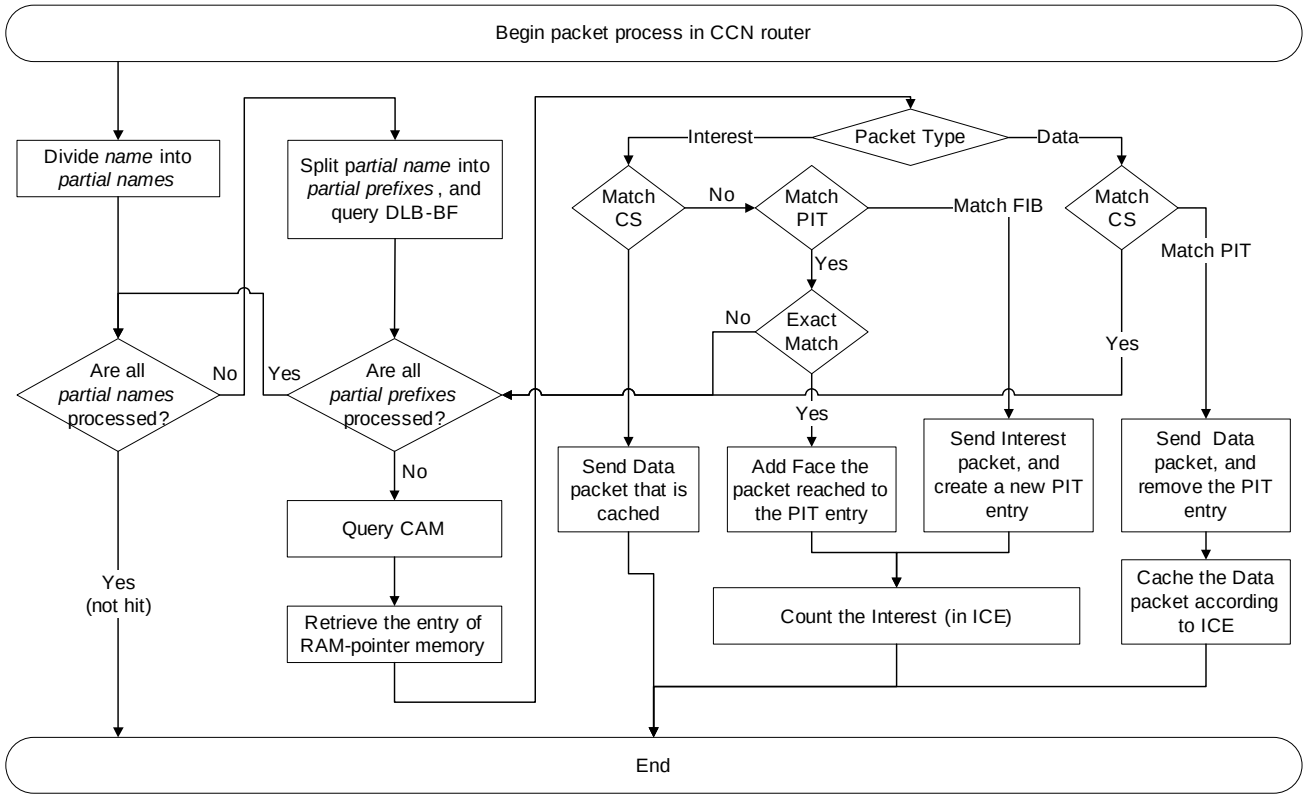

Fig. 8 Packet Processing in the Proposed CCN Router

bytes, Interest packets of 40 bytes, and Data packets of 1500 bytes; these values are the same as in [16], [17]. In addition, we assume that 99% of existing domain names are no longer than 40 bytes and have no more than six components [14]. We also experimentally evaluated the performance, testing the hardware design implemented in Quartus II using Verilog HDL.

Existing lookup mechanisms that exclude false positives and our proposed mechanism are compared on searches per second, throughput, and size of static random access memory (SRAM); the results are shown in Table 3. MATA-NW seems to be fast enough for our purposes, but its throughput is achieved by employing a pipeline to a GPU. Both the pipeline process and the GPU make it hard to reduce the latency of each packet process, although the pipelined process outputs the packets at high-speed. In this section, we demonstrate that NLE, which is our proposed lookup mechanism, can achieve higher throughput and much lower latency than the existing methods by using CAM and DLB-BF. The size of SRAM required for NLE is expected to be at most 576 MBytes of SRAM. Nevertheless, a large portion of the memory can be implemented with dynamic random access memory (DRAM) instead of SRAM. NLE is also expected to require 3.2 Gbits of CAM in addition to the SRAM. The approaches to cope with these challenges rely on splitting the CAM, and are discussed later.

Table 3 Performance of Lookup Mechanisms (Number of Entries: 10 Million.)

lookup mechanisms	Search (per s) [MSPS]	Throughput [Gbps]	SRAM size [MB]
Character Trie[13]	3.172	6.344	1,026.34
NCE [14]	4.017	8.034	718.44
ENPT [15]	20.67	41.34	116.02
NameFilter [13]	37.003	74.006	234.27
ATA (200μs latency) [16]	6.56	13.12	682.55
MATA (100μs latency) [16]	29.75	60.50	490.28
MATA-NW (100μs latency) [16]	63.52	127.04	490.28
NLE (proposed method)	81.6	167.116	576.0

5.1 Memory Size and Cost

Scalability is limited by the scalability of the CAM, and so the necessary amount of RAM is determined according to the anticipated number of CAM entries. We therefore discuss how much memory is required to implement NLE. For this determination, it is important that NLE consists of two buffers, DLB-BF, and CAM.

The required sizes of the partial name buffer S_N[bit] and the partial prefix buffer S_P[bit] can be calculated as follows:

$$S_N = M(W_2 + s), \tag{1}$$

$$S_P = N(W_1 + 2l), \tag{2}$$

where M is the number of entries in the partial name buffer; N is the number of entries in the partial prefix buffer; W_1 and W_2 are the bit-lengths of the buffers, as shown in Figure 6; s is the bit-length of the $< Address >$ field; and l is the bit-length of the $< Length >$ and $Index$ fields in Figure 6. According to the size of domain names mentioned above, we define $W = 40$[Bytes], as the size of a CAM entry and the upper size limit of an entry in the buffers shown in Figure 4. The buffer for partial names, whose capacity should be large enough to store complete name addresses, needs more than 37 entries to store a name whose length is the maximum transmission unit; therefore, we set $M = 32$ (cf. Figure 6). Since it is desirable to store all components into the buffer for partial prefixes, we set $N = 64$ according to the fact that the longest URL has roughly 70 components [17]. To achieve M and N, the partial name buffer needs 10 Kbits and the partial prefix buffer needs 22 Kbits; these buffer sizes are reasonable, although we must still investigate how parallel processing scales in terms of wiring cost.

The size of DLB-BF depends on the probability of a false positive. If m is the number of bits in the array, k is the number of hash functions, and n is the number of elements inserted into DLB-BF, the false positive probability α can be calculated as follows [20]:

$$\alpha = \left(1 - (1 - \frac{1}{m})^{kn}\right)^k \simeq \left(1 - e^{\frac{kn}{m}}\right)^k. \tag{3}$$

Since $k = \frac{m}{n} \log 2$ minimizes the probability α, the equation (3) results in the following expression:

$$\alpha = \left(\frac{1}{2}\right)^{\frac{m}{n} \log 2}, \tag{4}$$

which can be simplified to

$$\frac{m}{n} = -\frac{\log \alpha}{(\log(2))^2}. \tag{5}$$

By substituting $\alpha = 10^{-x}$ into Equation (5), we finally obtain the following equation:

$$\frac{m}{n} = \frac{\log 10}{(\log(2))^2} x \simeq 4.7925 \times x \tag{6}$$

This means that extending the length of each entry by about 4.8 bits decreases the probability of a false positive 10-fold.

The size of DLB-BF is shown in Figure 9. When $\alpha = 10^{-6}$ and $n = 10M$, the amount of memory required for BF is 288 Mbits, and this grows to 4.6 Gbits upon assigning 16 bits to each entry for implementing counting filters (denoted "CBF" in Figure 9), which allow deletion of entries. For practical purposes, at most 2.3 Gbits of SRAM (8×288 Mbits of SRAM) is currently available. DLB-BF can be simply scaled down by increasing the probability of a false positive from $\alpha = 10^{-6}$ to $\alpha = 10^{-3}$. We can also conserve SRAM by using DRAM or RLDRAM for implementing counting filters. Then, the required amount of SRAM

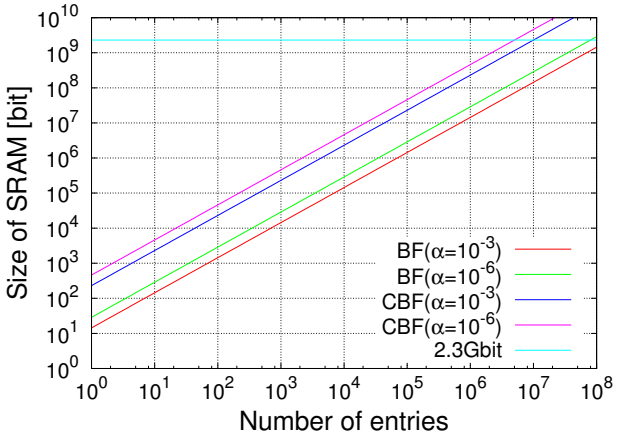

Fig. 9 The Size of SRAM Required for NLE

can be 288 Mbits, even when $\alpha = 10^{-6}$. If DLB-BF is implemented on SRAM, whose cost is, at present, approximately 1 USD/ MB [23], then 4.6 Gbits will cost 576 USD, and 288 Mbits will cost 36 USD. On the other hand, a 4.6 Gbits DLB-BF implemented on DRAM will cost only 4.50 USD (using a present value for DRAM of 1 USD/ 128 MB).

The memory required for CAM is the most serious problem because of the limited size available. When $W = 40$[Bytes], CAM requires 3.2 Gbits to hold 10 million entries. Although a single CAM with capacity on the order of gigabits does not exist, it is easier and more efficient to arrange many small CAMs. Since 1 Mbit of CAM currently costs about 1 USD, we can estimate the cost of the CAM to be 3200 USD. As a result, the total memory cost can be estimated at 3776 USD.

5.2 Throughput

The throughput of the CCN router strongly depends on the access time of NLE. The lookup operation in NLE requires accesses to two buffers, DLB-BF (implemented in SRAM), and CAM. Table 4 describes the minimum/average/maximum number of read/write access to the memory pools. In Table 4, α is the probability of a false positive from DLB-BF, $\bar{m}$ is the average number of partial names, $m_{\max}$ is the maximum number of partial names obtained from a name, and $S(m)$ and $T(m)$ are defined as follows:

$$S(m) = \sum_{i=1}^{m} n_i, \quad T(m) = \left\lceil \frac{S(m)}{N} \right\rceil,$$

where n_i is the number of partial prefixes obtained from the ith partial name and N is the number of entries in the partial prefix buffer. Table 4 covers three operations: the lookup process, the add process used when the added entry exists in CAM, and the add process used when the added entry is absent from CAM.

Although NLE is designed to handle names too long to store into a single entry, in practice, almost all names can

Table 4 Number of Read/Write Accesses in the Name Lookup Process, by Memory Type

		PNB[1]		PPB[2]		DLB-BF		CAM	
R(read)/W(write)		R	W	R	W	R	W	R	W
Lookup	min	0	0	1	1	1	0	1	0
	ave	$\bar{m}$	$\bar{m}$	$T(\bar{m})$	$S(\bar{m})$	$T(\bar{m})$	0	$\bar{m}+\alpha S(\bar{m})$	0
	max	$m_{\max}$	$m_{\max}$	$T(m_{\max})$	$S(m_{\max})$	$T(m_{\max})$	0	$m_{\max}+S(m_{\max})$	0
Add (in CAM)	min	0	0	0	0	0	0	1	0
	ave	$\bar{m}$	$\bar{m}$	0	0	0	0	$\bar{m}$	0
	max	$m_{\max}$	$m_{\max}$	0	0	0	0	$m_{\max}$	0
Add (not in CAM)	min	0	0	0	0	0	1	1	1
	ave	$\bar{m}$	$\bar{m}$	0	0	0	1	$\bar{m}$	1
	max	$m_{\max}$	$m_{\max}$	0	0	0	1	$m_{\max}$	1

[1] Partial Name Buffer
[2] Partial Prefix Buffer

be stored as a single entry and processed in a single buffer access (i.e., $\bar{m} \leq 1$) by setting $W = 40$. In addition, we can reduce the number of write accesses to the partial prefix buffer from $S(m)$ to m by parallelizing the process of writing partial prefixes. As a consequence, the access times required for the lookup and add operations are approximated as follows:

$$\begin{aligned} T_{\text{lookup}} &= (1.0+0.45)\times(2\bar{m}+2T(\bar{m})+S(\bar{m})) \\ &\quad +(1.0+4.0)\times(\bar{m}+\alpha S(\bar{m})) \\ &= 1.45\times(2+2+1)+5.0\times(1+0) \\ &= 12.25[\text{ns}] \\ T_{\text{add}} &= (1.0+0.45)\times(2\bar{m}+1) \\ &\quad +(1.0+4.0)\times(\bar{m}+1) \\ &= 1.45*(2+1)+5.0*(1+1) \\ &= 14.35[\text{ns}]. \end{aligned}$$

If the SRAM access time is 0.45 ns, then the CAM access time will be 4.0 ns, and the buffer access time will be 1.0 ns [17]. This access time results in a throughput for lookups of 81.6 million searches per second (MSPS) and a throughput for the add operation of 69.7 MSPS. With an average packet size of 256 bytes, these throughputs are roughly equivalent to 163 Gbits/s and 139 Gbits/s, respectively.

We also need to evaluate the environment in practical use, where there will be not only lookups but also updates. Unlike with IP packets, almost all CCN packets arriving at a router are looked up and result in updates to the tables in the router (i.e., an Interest will add a new PIT entry or update an existing PIT entry, and sending Data will remove the corresponding PIT entry). NLE can perform the lookup and update concurrently; therefore, the lookup and following update requires only an additional process to add/remove the matching entry to/from CAM and DLB-BF. The average access time for this operation is approximated as follows:

$$\begin{aligned} T_{\text{update}} &= T_{\text{lookup}}+(1.0+0.45)\times 1+(1.0+4.0)\times 1 \\ &= 12.25+6.45 \\ &= 18.70[\text{ns}]. \end{aligned}$$

The throughput is 53.5 MSPS (109 Gbps), assuming the same parameters as above. Thus, we can realize CCN router processing at line speed. These throughputs could be greatly improved by designing pipeline mechanisms for concurrent lookups.

In addition to the above analysis, we designed an experimental implementation of NLE in Quartus II using Verilog HDL. The implementation is simplified in terms of the bit-widths and memory accesses. Specifically, the bit-width of each name is set to be 1/16th of that in the design discussed above, and a simple equivalent circuit replaces the SRAM and CAM needed for DLB-BF and CAM. By using the timing analyzer in Quartus II, we evaluated the maximum clock speed in Cyclone-V GT (5CGTFD9E5F35C7) and Stratix V (5SGSMD3H1F35C1); these gave achieved 177.84 MHz and 517.6 MHz, respectively. In the future, we plan to improve the hardware design and evaluate the feasibility in terms of the required power and number of logic elements.

6. Conclusion and Future Work

This paper contributes evidence for the feasibility of CCN by designing concrete CCN router hardware and evaluating its performance. Needless to say, it is a requirement for implementation of CCN that CCN routers be feasible. In addition, accurate estimates of actual performance are essential to all sorts of network-level simulations. We addressed these problems by proposing CAM-based CCN router architecture. We proposed NLE, which consists of many small CAMs and DLB-BF and allows reasonable costs, and ICE, which assists in adaptive caching; thus, we have shown the entire design of a CCN router. We also performed a basic theoretical analysis of the expected throughput and cost of the CCN router.

As discussed in Section 4.1, we split the monolithic CAM into smaller parts to make NLE feasible. A significant challenge for our architecture is to scale the memory capacity and the number of entries. There are no existing TCAMs with more than 100 Mbits of memory capacity. In

addition, the power cost of a TCAM can be approximated as 1 kW/Mbit; the power requirements of our router, which needs at least 3.2 Gbits of CAM, can rise to more than 3 kW; however, even a 1 kW power requirement is beyond the capacity of any existing implementation by several orders of magnitude. These challenges can be addressed by using multiple small CAMs. A sufficient number of small CAMs can dramatically reduce the costs for capacity and power. Employing 32 units of small CAM instead of 1 large monolithic CAM, for example, allows NLE to be implemented on an existing 100-Mbit TCAM chip. As a rough estimate, the distributed CAMs require only $1/32$ of the power of a large single CAM. We can adjust the number of CAMs running in parallel to achieve the desired trade-off between the power and the lookup performance.

FIB is required to handle websites, the number of which is approaching 1 billion according to a survey in [24]. The line-speed (40 Gbps) traffic, whose average round-trip delay time (RTT) is RTT = 100ms, imposes 2 million entries per port on PIT. Even if the effect of ICE is maximized, the number of chunks that would be stored in CS for 10 million entries is equivalent to the amount of files accessed per day in terms of city-scale traffic [22].

In the future, it seems likely that such numbers increase rapidly along with the number of content items. Our architecture can easily scale with this increase in the number of names by installing additional distributed CAMs according to the number of entries required for NLE. A simple solution—ignoring lookup time—is to partially replace CAM with a hash table; however, the limits of the system can be relaxed without sacrificing speed because we can use not only 16 T / cell TCAMs but also 10 T / cell BCAMs, and we expect exponential growth in the capacity and availability of feasible memory. An architecture that combines CAM and a high-speed hash table to achieve further scalability and packet processing time will be studied in future work.

We also plan to evaluate the router performance in a hardware implementation of the router. The calculations in this paper show that it is essential to evaluate the practical throughput in terms of the likely cost of hardware implementation. Before physical implementation, an advanced mechanism to parse packets and control buffers in parallel must be developed. Additionally, FIB, PIT, and CS require policies for managing their entries and implementation details, including timeout, retransmission, and cache-replacement algorithms. We should take account of the advanced name lookup mechanisms that are beginning to be specified, such as ChildSelector, Min(Max)Suffix, and Exclude [21]. Several implementations of those mechanisms require the name lookup algorithm (III) or (IV) shown in Table 1, although our proposed architecture implements the name lookup algorithm (I). Ultimately, these developments will result in practical network-level evaluation and a realistic analysis of network bottlenecks.

Acknowledgment

This work was supported by the Strategic Information and Communications R&D Promotion Programme (SCOPE) of the Ministry of Internal Affairs and Communications, Japan.

References

[1] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard, "Networking named content," Proceedings of the ACM CoNEXT 2009, pp.1–12, December 2009.
[2] G. Xylomenos, C.N. Ververidis, V.A. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K.V. Katsaros, and G.C. Polyzos, "A survey of Information-Centric Networking research," IEEE Communications Surveys Tutorials, vol.16, no.2, pp.1024–1049, February 2014.
[3] C. Fricker, P. Robert, J. Roberts, and N. Sbihi, "Impact of traffic mix on caching performance in a content-centric network," Proceedings of the IEEE Conference on Computer Communications 2012, pp.310–315, March 2012.
[4] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," Proceedings of the ACM Re-Architecting the Internet Workshop, pp.1–6, November 2010.
[5] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J.D. Thornton, D.K. Smetters, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh, "Named data networking (NDN) project," October 2010.
[6] 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, July 2012.
[7] V. Jacobson, D.K. Smetters, N.H. Briggs, M.F. Plass, P. Stewart, J.D. Thornton, and R.L. Braynard, "VoCCN: Voice-over Content-Centric Networks," Proceedings of the 2009 workshop on Re-architecting the internet, pp.1–6, December 2009.
[8] "CCNx." http://www.ccnx.org/, 2014.
[9] N. Fotiou, P. Nikander, D. Trossen, and G.C. Polyzos, "Developing information networking further: From PSIRP to PURSUIT," Proceedings of the 7th International ICST Conference on Broadband Communications,Networks, and Systems, pp.1–13, October 2010.
[10] T. Levä, J. Gonçalves, R.J. Ferreira, *et al.*, "Description of project wide scenarios and use cases," February 2011.
[11] M. Varvello, D. Perino, and J. Esteban, "Caesar: a content router for high speed forwarding," Proceedings of the 2nd edition of the ICN workshop on Information-centric networking, pp.73–78, August 2012.
[12] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "DiPIT: A distributed bloom-filter based PIT table for CCN nodes," Proceedings of the 21st ICCCN 2012, pp.1–7, July 2012.
[13] Y. Wang, T. Pan, Z. Mi, H. Dai, X. Guo, T. Zhang, B. Liu, and Q. Dong, "NameFilter: Achieving fast name lookup with low memory cost via applying two-stage bloom filters," Proceedings of the IEEE INFOCOM 2013, pp.95–99, April 2013.
[14] 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," Proceedings of the IEEE 32nd International Conference on Distributed Computing Systems 2012, pp.688–697, June 2012.
[15] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in Named Data Networking," Proceedings of the ACM/IEEE 8th Symposium on Architectures for Networking and Communications Systems 2012, pp.211–222, October 2012.
[16] Y. Wang, Y. Zu, T. Zhang, K. Peng, Q. Dong, B. Liu, W. Meng, H. Dai, X. Tian, Z. Xu, H. Wu, and D. Yang, "Wire speed name lookup: a GPU-based approach," Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, pp.199–212, April 2013.

[17] D. Perino and M. Varvello, "A reality check for Content Centric Networking," Proceedings of the ACM SIGCOMM workshop on Information-centric networking, pp.44–49, August 2011.
[18] P. Bosshart, G. Gibb, H.S. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding metamorphosis: fast programmable match-action processing in hardware for SDN," Proceedings of the ACM SIGCOMM 2013, pp.99–110, August 2013.
[19] A. Ooka, S. Ata, K. Inoue, and M. Murata, "Design of a high-speed content-centric-networking router using content addressable memory," Proceedings of IEEE INFOCOM 2014 Workshop on Name-Oriented Mobility, Toronto, pp.1–6, April 2014.
[20] H. Song, F. Hao, M. Kodialam, and T.V. Lakshman, "IPv6 lookups using distributed and load balanced bloom filters for 100Gbps core router line cards," Proceedings of the IEEE INFOCOM 2009, pp.2518–2526, April 2009.
[21] "CCNx 1.0 protocol specifications roadmap." http://www.ietf.org/mail-archive/web/icnrg/current/pdfZyEQRE5tFS.pdf, November 2013.
[22] F. Guillemin, B. Kauffmann, S. Moteau, and A. Simonian, "Experimental analysis of caching efficiency for YouTube traffic in an ISP network," Proceedings of the 25th International Teletraffic Congress, pp.1–9, September 2013.
[23] S. Iyer, R. Kompella, and N. McKeown, "Designing packet buffers for router linecards," IEEE/ACM Transactions on Networking, vol.16, no.3, pp.705–717, June 2008.
[24] "netcraft." http://www.netcraft.com/, December 2013.

Atsushi Ooka received an M.E. degree from the Graduate School of Information Science and Technology, Osaka University, in 2014. Currently, he is a Ph.D. candidate. His research interests include the design and implementation of a router hardware for content-centric networking.

Shingo Ata received M.E. and D.E. degrees in Informatics and Mathematical Science from Osaka University in 1998 and 2000, respectively. Since 2013, he has been a professor at the Graduate School of Engineering at Osaka City University. His research work includes networking architecture, design of communication protocols, and performance modeling of communication networks. He is a member of IEICE, IEEE, and ACM.

Kazunari Inoue received his B.E. and D.E. degrees in Information Technology from Yokohama National University and the Research Institute for Nano-device and Systems at Hiroshima University, Japan, in 1984 and 2005, respectively. In 1984, he joined Mitsubishi Electric Corporation, and moved to Renesas Electronics Corporation in 2003, where he has been engaged in the development of LSI and solutions in network applications, research and design of switching, routing, and buffering. He is a member of IEICE and the Semiconductor Technology Academic Research Center (STARC).

Masayuki Murata received his M.E. and D.E. degrees from Osaka University, Japan, in 1984 and 1988, respectively. In 1984, he joined the Tokyo Research Laboratory, IBM Japan, as a researcher. He was an assistant professor from 1987 and an associate professor at Osaka University from 1992 to 1999. Since 1999, he has been a professor at Osaka University, and he is now with the Graduate School of Information Science and Technology, Osaka University. He has more than 500 papers in international and domestic journals and conferences. His research interests include computer communication networks, performance modeling, and evaluation. He is a member of IEICE, IEEE, and ACM.