Design of a High-speed Content-centric-networking Router Using Content Addressable Memory

Atsushi Ooka*, Shingo Ata†, Kazunari Inoue*‡ and Masayuki Murata*
* Graduate School of Information Science and Technology, Osaka University, Osaka, Japan,
Email:a-ooka@ist.osaka-u.ac.jp, murata@ist.osaka-u.ac.jp
† Graduate School of Engineering, Osaka City University, Osaka, Japan, Email: ata@info.eng.osaka-cu.ac.jp
‡ Nara National College of Technology, Nara, Japan, Email: inoue.kazunari@ist.osaka-u.ac.jp

***Abstract*—Content-centric networking (CCN) is an innovative network architecture that is being considered as a successor to the Internet. To implement the novel 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 incorporating two entities that we propose. One of these entities is the name lookup entity (NLE), which looks up a name address within a few cycles from content addressable memory (CAM) by use of a Bloom filter; the other is the interest count entity (ICE), which supports to select 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 performance and cost on the basis of a concrete hardware design.**

I. Introduction

Information-centric networking (ICN) or content-centric networking (CCN) [1] has been proposed as a measure for overcoming the limitations of current Internet architecture, and a number of research projects, such as CCNx [2], NDN [3], PURSUIT [4], and SAIL [5], have explored these candidates. Obviously, many challenges must be resolved to realize ICN/CCN, which is a clean-slate network. First, we need new name resolution and routing mechanisms that are based on the name addresses used in CCN [1]. Second, the "bread crumb" forwarding technique, which naturally incorporates multicast and request aggregation into the network, requires lookup tables that can update much more quickly than IP tables. Most research focuses on in-network caching mechanisms because they can cache content more efficiently and thus require fewer resources [6], [7]. In addition, there are a number of problems that have been analyzed and evaluated: security, mobility, and CCN deployment, among others [3], [8], [9].

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, hardware must be feasible to realize CCN communication. 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 of the router.For example, Caesar [10] aims to implement a scalable high-speed forwarding table. DiPIT [11] and NameFilter [12] focus on PIT and propose very fast inexpensive architecture consisting of two-level Bloom filters, but the probabilistic model means that false positives can never be completely eliminated. NCE [13], ENPT [14] and ATA(MATA) [15] approach memory-efficient name lookup mechanisms by using a trie-like structure. MATA achieves wire speed by means of a highly parallelized architecture using GPU, although it is difficult to reduce the latency. In addition, among the existing complete router designs [16], CAM, which has the potential to become a major lookup technology, has not been sufficiently researched because of its cost.

In this paper, we propose a complete CCN router design that can be implemented with existing hardware and show the feasibility and performance of the router. In Section II, we describe an accurate communication model for CCN that properly handles all packets. In Section III, customizing the router architecture by using the name lookup entity (NLE) and the interest count entity (ICE) is proposed, and the hardware design using CAM and a Bloom filter is demonstrated in Section IV. In Section V, we comprehensively evaluate the throughput and cost of the CCN router. Finally, we give a conclusion and discuss areas for future research.

II. Background

CCN was designed with a focus on the content, rather than the location, of data. To this end, each chunk of data has a *name* that acts as a unique, human-readable, and hierarchically structured address; therefore, it is not necessary to specify the locations of providers and consumers. 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 Interest, which contain the name of the content. In response to 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 may be just a prefix of the requested content. For example, when a consumer requests

[1]In this paper, we especially focus on CCN hereafter.

978-1-4799-3088-3/14/$31.00 ©2014 IEEE

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 as *name siblings*. Since there is no inherent reason to forbid use of name siblings by applications running on CCN, we also include name siblings in our discussion although it is not obvious that name siblings will be accepted for CCN communications.

A. Router Behavior

We adopt the design principles of Named Data Networking (NDN) [3] 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: forwarding information base (FIB), pending interest table (PIT), and content store (CS). The FIB is a table used for determining the proper interface for forwarding Interest that have arrived at the router. The PIT remembers the interfaces from which Interest have arrived so that it can send back the matching Data that will be subsequently received by the router. Interest that have 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.

B. Name Lookup Algorithms

An algorithm for lookup tables in a CCN router is not trivial, and to our best 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; 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 Interest and Data must be looked up in the FIB, PIT, and CS tables. There are five possible combinations because Data are not looked up in the FIB table.

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: a) *Exact Match (EM)*, which matches when $K_S = K_E$, b) *Search-key Prefix Match (SPM)*, which matches when $P(K_S) = K_E$, c) *Entry-key Prefix Match (EPM)*, which matches when $K_S = P(K_E)$, and d) *Both-keys Prefix Match (BPM)* which matches when $K_S = K_E$and when one is identical to the prefix of the other (*not* $P(K_S) = P(K_E)$).

The non-exact matching algorithms might retrieve multiple entries, therefore, 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 longest-prefix-matching (LPM) algorithm used in conventional IP routers. Selections from 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 that are same length, other criteria for 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 packets are multicast from all ports corresponding to the matched entries although generating excessive traffic.

2) Algorithms Suitable for Each Table: The combination of SPM and LPM is the most suitable for FIB, which accords with the strategy for current IP routers. In fact, 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 active name and must be a complete name that identifies specific content. The other algorithms do not support name siblings. The process to look up Data in CS is essential for avoiding 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 in Interest. 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.

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. Although using LPM is a risk-free approach, it is more efficient to satisfy multiple Interests at once if name siblings are disallowed.

When looking up an Interest in PIT, both EM and BPM are more suitable matching algorithms than the others, although we omit the details here because of complexity and space limitations. In brief, EM can aggregate only Interests whose names are identical, but EM is also the only solution that handles name siblings. In contrast, BPM makes full use of active naming although re-registering an entry and priority rules for selecting is required.

Table I summarizes the available algorithms for matching and selecting the entry in cases other than looking up Interest in FIB or CS. Although all combinations support active naming, only combination (I) is able to cope with name siblings.

TABLE I
SUMMARY OF MATCHING AND SELECTING ALGORITHMS

	CS-Interest	PIT-Data	PIT–Interest
(I)	EM/-	SPM/LPM, FIFO[1]or all hit	EM/-
(II)	EM/-	SPM/-	BPM/optimal
(III)	EPM/optimal	SPM/LPM, FIFO or all hit	EM/-
(IV)	EPM/optimal	SPM/-	BPM/optimal

[1] FIFO: first in, first out

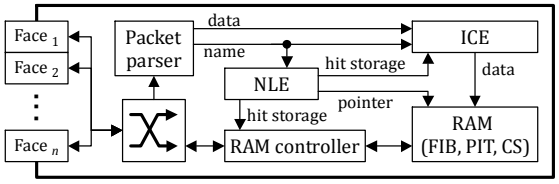

Fig. 1. CCN Router Architecture

III. ARCHITECTURE

To handle the variable-length name address at wire speed, we introduce CAM and a distributed-and-load-balancing Bloom filter (DLB-BF) [17] into the prefix table; an associated element is an NLE. NLE maps between a name address and entries in each of the three tables so that only one lookup is required to retrieve the most specific entry from among the three tables without a false positive. DLB-BF, which allows membership queries for all of the prefixes of a name address to be performed in parallel, reduces the workload on the CAM. We also present ICE, which is a new mechanism for identifying content worth caching.

The most suitable matching and selecting algorithms for the lookup mechanism using the CAM and Bloom Filter is combination (I) in Table I. If name siblings are disallowed, the combination (I) makes the lookup mechanism simple by choosing SPM for all matching algorithms except the lookup of Interests in PIT. For this reason, we assume that there are not any name siblings. Additionally, Binary-CAM (BCAM) can be used instead of Ternary-CAM (TCAM); for our approach, BCAM is more suitable.

Figure 1 illustrates the basic architecture of the proposed CCN router. First, an input packet is received on a face. 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 Interests sought 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.

A. NLE

We propose NLE, which implements a fast lookup operation for a name address. 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. In addition, DLB-BF can dramatically reduce the load on CAM without sacrificing speed.

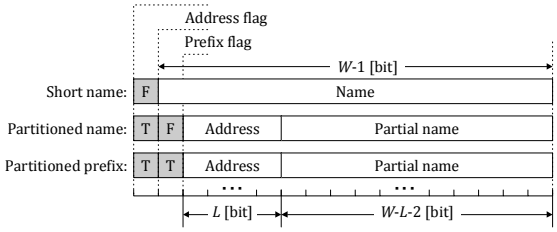

Fig. 2. Definition of CAM Entry

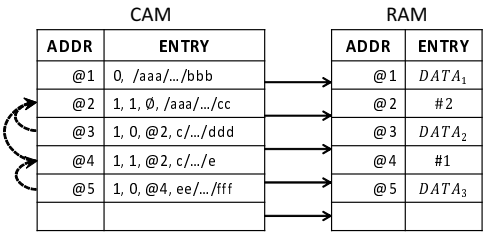

Fig. 3. Example of CAM and RAM Entries in NLE

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). SN is used whenever a 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 (or a root node). Figure 2 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 3. 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 $(W-L-2)$-bits-wide segments: two entries (N_1^B, N_2^B) and three entries (N_1^C, N_2^C, N_3^C), respectively. The values of an entry in CAM also correspond to the definition in Figure 2. N^A is stored in CAM and RAM as 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 is 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 PN from the name N_2^B and the address '@2', where the parent node N_1^B was retrieved, 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. In face, setting $W = 320$ makes it rare to perform multiple lookups for a long name like N^B and N^C according to the fact that 99% of domain names are no longer than 40 bytes [13].

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.

B. ICE

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 just a few times from occupying limited capacity.

IV. Hardware Design

A. Design of NLE

We now detail an implementation of NLE. Figure 4 shows the hardware design of NLE. Roughly, NLE consists of four components: the unit to process *partial names*, the unit to process *partial prefixes*, DLB-BF, and CAM.

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. Since looking up a child node requires the address of its parent node, a buffer for a partial name contains not only its string but also an address for the partial name. Secondly, 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.

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 2, 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.

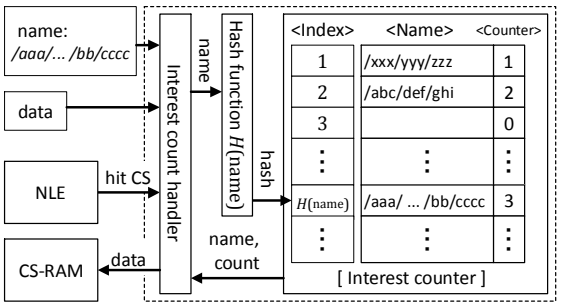

Fig. 5. Hardware Design of ICE

B. Design of ICE

The hardware design of ICE is illustrated in Figure 5. 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 that 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 a Data that has not yet been cached arrives at the router, the data is cached in CS if 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 more than a certain threshold can be much more memory efficient. According to [18], 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

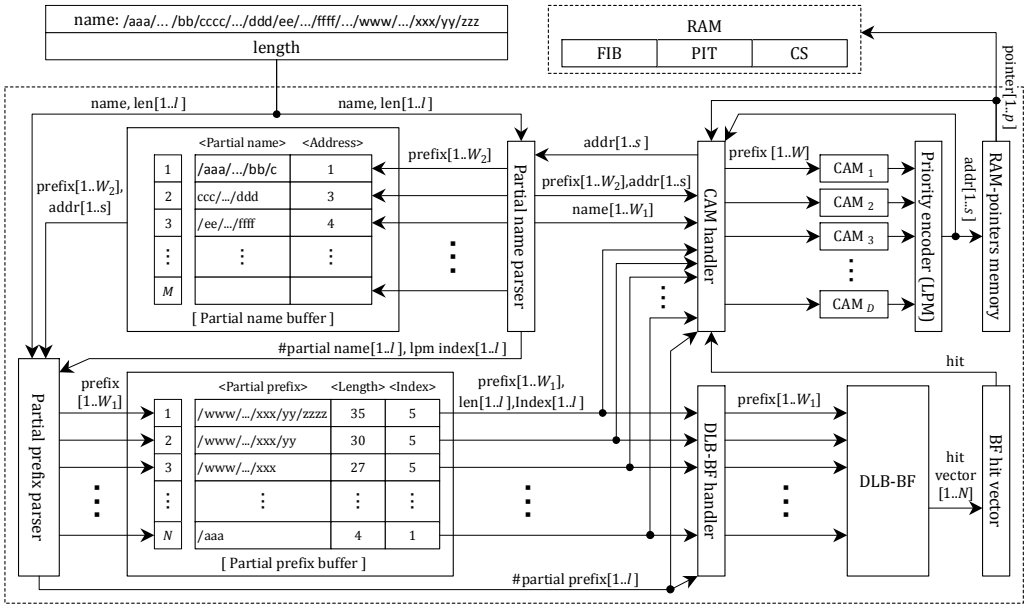

Fig. 4. Hardware Design of NLE

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.

V. 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 bytes, Interest packets of 40 bytes, and Data packets of 1500 bytes; these values are the same as in [16], [15]. In addition, we assume that 99% of existing domain names are no longer than 40 bytes and have no more than six components [13].

Firstly, we consider the memory size and cost. Scalability is limited by CAM, and so the necessary amount of RAM is determined according to the number of CAM entries. We therefore discuss how much memory is required to implement NLE, which consists of two buffers, DLB-BF and CAM.

According to the size of domain names mentioned above, we define $W = 320$, as the size of a CAM entry and the upper size limit of an entry in the buffers shown in Figure 2. 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 4). 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 [16]. 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 [17]: $\alpha = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \simeq \left(1 - e^{\frac{kn}{m}}\right)^k$. Since $k = \frac{m}{n} \log 2$ minimizes the probability α, this equation with $\alpha = 10^{-x}$ results in $\frac{m}{n} \simeq 4.8 \times x$. This means that extending the length of each entry by about 4.8 bits decreases the probability of a false positive 10-fold. When $\alpha = 10^{-6}$ and $n = 10M$, the amount of memory required for DLB-BF is 288 MB, and this grows to 4.6 GB upon assigning 16 bits to each entry for implementing counting filters, which allow deletion of entries. Implementing DLB-BF on SRAM, whose cost is approximately 1 USD / MB [19], the 4.6 GB for the DLB-BF will cost 4,600 USD.

The memory required for CAM is the most serious problem because of the limitation of the size. When $W = 320$, 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 3,200 USD. As a result, the total memory cost can be estimated at 7,800 USD.

Secondly, we consider the throughput. The throughput of CCN router strongly depends on the access time of NLE. The lookup operation in NLE requires accesses to two buffers, DLB-BF implemented with SRAM and CAM. Although NLE is designed to handle names too long to store into a single entry, in practice, almost all names can be stored as a single entry and processed in a single buffer access by setting $W = 320$. As a consequence, the access times required for lookup and add operations are approximately $T_L = 12.25[\mathrm{ns}]$ and $T_A = 14.35[\mathrm{ns}]$, respectively, if SRAM access time is

0.45 ns, CAM access time is 4.0 ns, and buffer access time is 1.0 ns [16]. This access time results in throughput for lookup (and deletion) of 81.6 million searches per second (MSPS) and throughput for the add operation of 69.7 MSPS. With an average packet size of 256 bytes, these throughputs are roughly equivalent to 163 Gbps and 139 Gbps, respectively. Thus, we can realize CCN router processing at wire speed. These throughputs could be greatly improved by designing mechanisms for concurrent lookups.

VI. Conclusion and Future Work

Our paper contributes evidence for the feasibility of CCN by designing concrete CCN router hardware and evaluating its performance. We addressed the challenges by proposing CAM-based CCN router architecture. We also 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.

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. Furthermore, our evaluated situation, which assumed 10 million entries, will not be practical in the future. FIB is required to handle websites, the number of which is approaching 1 billion according to a survey in [20]. The line-speed (40 Gbps) traffic, whose average round-trip delay time (RTT) is $\mathrm{RTT} = 100\mathrm{ms}$, imposes 2 million entries per port on PIT. Even if the effect of ICE is maximized, the number of chunks stored in CS for 10 million entries is equivalent to the amount of files accessed per day in terms of city-scale traffic [18]. By disregarding lookup time, we can easily scale our router by using a hash table instead of CAM; however, these limitations can be relaxed without sacrificing speed because we can use not only 16 T / cell TCAM but also 10 T / cell BCAM, and we expect exponential growth in feasible memory. An architecture that combines CAM and a high-speed hash table to balance scalability and packet processing time will be studied in future work.

We also plan to evaluate the router performance based on a hardware implementation of the router. The calculations in this paper show that it is essential to evaluate the practical throughput from the likely cost of hardware implementation. Before physical implementation, an advanced mechanism to parse packets and control buffers in parallel must be developed. Ultimately, this 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," in *Proceedings of the ACM CoNEXT 2009*, December 2009, pp. 1–12.

[2] "CCNx," PARC, 2014. [Online]. Available: http://www.ccnx.org/

[3] 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," pp. 1–24, October 2010. [Online]. Available: http://named-data.net/techreport/TR001ndn-proj.pdf

[4] N. Fotiou, P. Nikander, D. Trossen, and G. C. Polyzos, "Developing information networking further: From PSIRP to PURSUIT," in *Proceedings of the 7th International ICST Conference on Broadband Communications,Networks, and Systems*, October 2010, pp. 1–13.

[5] T. Levä, J. Gonçalves, R. J. Ferreira *et al.*, "Description of project wide scenarios and use cases," pp. 1–99, February 2011. [Online]. Available: http://www.sail-project.eu/wp-content/uploads/2011/02/SAIL_D21_Project_wide_Scenarios_and_Use_cases_Public_Final.pdf

[6] C. Fricker, P. Robert, J. Roberts, and N. Sbihi, "Impact of traffic mix on caching performance in a content-centric network," in *Proceedings of the IEEE Conference on Computer Communications 2012*, March 2012, pp. 310–315.

[7] S. Arianfar, P. Nikander, and J. Ott, "On content-centric router design and implications," in *Proceedings of the ACM Re-Architecting the Internet Workshop*, November 2010, pp. 1–6.

[8] 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.

[9] 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," in *Proceedings of the 2009 workshop on Re-architecting the internet*, December 2009, pp. 1–6.

[10] M. Varvello, D. Perino, and J. Esteban, "Caesar: a content router for high speed forwarding," in *Proceedings of the 2nd edition of the ICN workshop on Information-centric networking*, August 2012, pp. 73–78.

[11] W. You, B. Mathieu, P. Truong, J. Peltier, and G. Simon, "DiPIT: A distributed bloom-filter based PIT table for CCN nodes," in *Proceedings of the 21st ICCCN 2012*, July 2012, pp. 1–7.

[12] 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," in *Proceedings of the IEEE INFOCOM 2013*, April 2013, pp. 95–99.

[13] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen, "Scalable name lookup in NDN using effective name component encoding," in *Proceedings of the IEEE 32nd International Conference on Distributed Computing Systems 2012*, June 2012, pp. 688–697.

[14] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in Named Data Networking," in *Proceedings of the ACM/IEEE 8th Symposium on Architectures for Networking and Communications Systems 2012*, October 2012, pp. 211–222.

[15] 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," in *Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation*, April 2013, pp. 199–212.

[16] D. Perino and M. Varvello, "A reality check for Content Centric Networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, August 2011, pp. 44–49.

[17] 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," in *Proceedings of the IEEE INFOCOM 2009*, April 2009, pp. 2518–2526.

[18] F. Guillemin, B. Kauffmann, S. Moteau, and A. Simonian, "Experimental analysis of caching efficiency for YouTube traffic in an ISP network," in *Proceedings of the 25th International Teletraffic Congress*, September 2013, pp. 1–9.

[19] 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.

[20] "netcraft," December 2013. [Online]. Available: http://www.netcraft.com/