Master's Thesis

Title

Hardware Design and Evaluation of CAM-based High-speed CCN Router

Supervisor

Professor Masayuki Murata

Author

Atsushi Ooka

February 10th, 2014

Department of Information Networking

Graduate School of Information Science and Technology

Osaka University

Master's Thesis

Hardware Design and Evaluation of CAM-based High-speed CCN Router

Atsushi Ooka

Abstract

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 CCN router is the primary contribution of this thesis. We provide a concrete hardware design for a CCN router model that uses three basic tables —forwarding information base (FIB), pending interest table (PIT), and content store (CS)—and 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 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 CCN router performance and cost on the basis of a concrete hardware design.

Keywords

Communication Architecture

Future Networks

Content-centric Networking

Router Hardware

Content Addressable Memory

Bloom Filter

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

1.1 Background

The Internet, which is now a global network of networks, is used in a form and scale considerably different from the original design principles and assumptions, and this gives rise to many problems. The initial implementation of the Internet was developed for providing communications between pairs of hosts. At present, however, the Internet is used for the purpose of distributing and retrieving various types of content to and from global networks instead of communicating with a specific host. Nevertheless IP datagrams require that an IP address be assigned, which specifies the network interface. To map an IP address to the data in which we are interested, network applications need to employ complex middleware services such as DNS. In addition, there is no guarantee that the requested data always exist at the same location, because the data may be moved or deleted, or the server providing data may become temporarily inaccessible.

Information-centric networking (ICN) or content-centric networking (CCN) [1] has been proposed as a measure for overcoming the limitations of current Internet architecture. The most significant feature in ICN/CCN is that the "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.

Obviously, many challenges must be resolved to realize 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. 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 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 [2, 3]. In addition, there are a number of problems that have been analyzed and evaluated: security, mobility, and CCN deployment, among others [4, 5, 6].

1.2 Related Work

A number of research projects have studied ICN/CCN approaches such as CCNx [7], NDN [4], PURSUIT [8], and SAIL [9]. These all share the common concept of addressing the content that is exchanged in communication by “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 security that is built into data. In this thesis, 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 of the router, such as caching and name lookup mechanisms. For example, Caesar [10] aims to implement a scalable high-speed forwarding table. DiPIT [11] and NameFilter [12] focus on pending interest table (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 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 [16, 3], 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 is too difficult to support an Internet scale CCN deployment although the analysis that at least CDN or ISP scale can be easily afforded is a significant contribution to investigating the feasibility of ICN/CCN. In addition, the designs and simulations of CCN router shown in the existing studies are not based on hardware designs or implementations. We eventually plan to demonstrate the design and evaluation based on hardware with the same level of [17]. In [17], the implementation of reconfigurable match tables for software-defined network (SDN) is proposed and detailed design and evaluation based on hardware implementations are described. The proposed techniques for quick matching a number of entries in SDN could help to implement feasible matching mechanisms for ICN/CCN, but the techniques in SDN cannot be directly applied to the router in CCN because of the variable-length name addresses in 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, 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, such as caching and name lookup mechanisms.

In this thesis, 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 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) is proposed, and the hardware design using content addressable memory (CAM) and a Bloom filter is demonstrated in Section 4. In Section 5, we comprehensively evaluate the throughput 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 not on the location of data but on the content of 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 for native multicasting, in-network caching, and security that is built into the 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 Internet, which is the dominant tool for communication, imposes these kinds of redundant processes solely for deploying or retrieving the data that is contained within. In addition, duplicate packets carrying identical information as part of data sharing systems such as content delivery networks (CDN) and peer-to-peer (P2P) applications consume bandwidth and degrade network performance. CCN is a solution for dealing with these incompatibilities and security concerns by shifting the routing behavior based on from "where" to "what".

A *name*, which is assigned to each chunk of data, plays a major role in CCN instead of the IP address that is assigned to device interfaces. In CCN, it is not necessary to know where the device we want to communicate with is located, and instead we identify the name of data 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 segmentation of data. 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 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 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 thesis. 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 thesis, we call pairs with this relationship as *name siblings*. Name siblings complicate the handling of name addresses in a router, but there is no inherent reason to forbid use of name siblings by applications running on CCN. It is not obvious that name siblings will be accepted for CCN communications, but we also include name siblings in our discussion.

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) [4] in this thesis. 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), 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.

2.2 Name Lookup Algorithm

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.

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 $K_S = K_E$and when one is identical to the prefix of the other (*not* $P(K_S) = P(K_E)$, i.e., both keys contain the same prefix).

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-pattern-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 excessive traffic could be generated.

2.2.2 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 abbreviated active name and must be a complete name that identifies specific content. Additionally, 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. If name siblings are allowed, EM must be used in order to prevent a router from returning undesirable Data to Interest. Otherwise, EPM could improve the cache hit rate by exploiting the advantage of active naming. Suppose there is $K_E^1 =$ "`/video/a.mpg/v1/s1`" in CS. When searching Interest named $K_S^2 =$ "`/video/a.mpg`", K_E^1 can match the 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. Suppose there are $K_E^1 =$ "`/video/a.mpg/v1/s1`", $K_E^2 =$ "`/video/a.mpg/v2/s6`" and $K_E^3 =$ "`/video/a.mpg`" in PIT. When Data named K_S= "`/video/a.mpg/v1/s1`" arrived at the router, the Data can satisfy not only K_E^1 but also K_E^3 by selecting all matching entries. Needless to say, because the Interest requesting content named "`/video/a.mpg`" must not be satisfied with the content named "`/video/a.mpg/v1/s1`", the all hit 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 explain the reason, consider two cases; (1) a case where PIT contains $K_E^1 =$ "`/text/A.txt/v1/s1`" and $K_E^2 =$ "`/text/A.txt/v2/s6`" and Interest whose name is $K_S^1 =$ "`/text/A.txt`" is looked up, and (2) a case where PIT contains $K_E^3 =$ "`/text/A.txt`" and 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 $Case(2)$, Interests match none of entries and registered as a new entry. Thus EM cannot aggregate Interests whose names are identical to the prefix of entries in PIT, but EM is the only solution that handles name siblings. We must concern about the problem that Interest whose name is an abbreviated active name but name sibling cannot be satisfied due to consecutive arrivals of Interests whose names are complete. For instance, consider that PIT receives many Interests whose names contain the information of version or segment number of the data, such as "`/text/A.txt/v1/s1`", "`/text/A.txt/v1/s2`", $\cdots$, and "`/text/A.txt/v1/s`N" in $Case(2)$. Until all the sequential requests are satisfied, K_E^3 has been unsatisfied. In most cases, the request of K_E^3 may be expired and retransmitted. To eliminate the expirations and retransmissions, FIFO (first in, first out) may be necessary for looking up Data in PIT as the selecting algorithm.

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 $Case(2)$, the K_E^3 is shorter than the Interest's K_S^2. K_S^2 cannot be aggregated to K_E^since the shorter key matches Data whose name is different from K_S^2 (e.x., "`/text/A.txt/v2/s6`") and the origin Interest will be unsatisfied. Therefore, any matching entry with 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 name siblings by eliminating the Interests with shorter names.

Finally, SPM and EPM are available but they are unsuitable. 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. Although all combinations support active naming, only combination (I) is able to cope with name siblings.

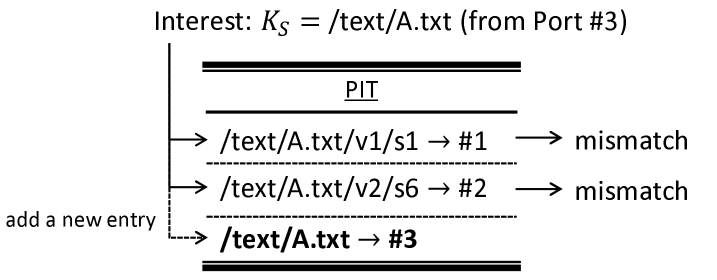
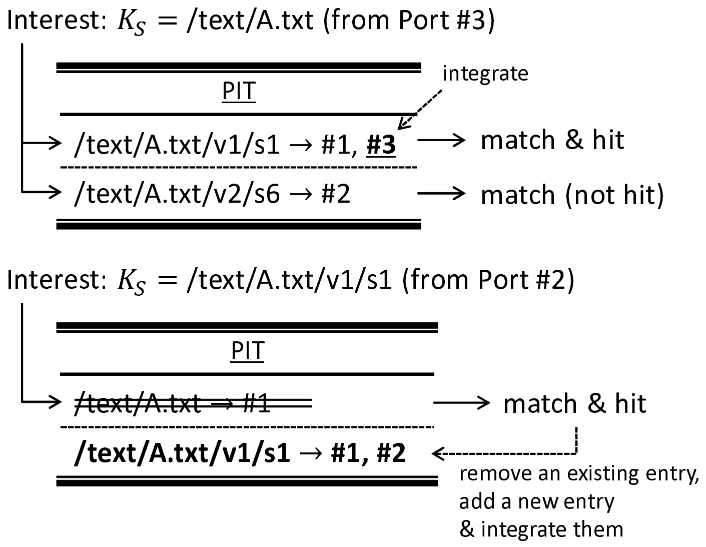


Figure 1: Example of Using EM (upper: $Case(1)$, lower: $Case(2)$)



Interest: K_S = /text/A.txt (from Port #3)
PIT
integrate
/text/A.txt/v1/s1 → #1, #3
match & hit
/text/A.txt/v2/s6 → #2
match (not hit)
Interest: K_S = /text/A.txt/v1/s1 (from Port #2)
PIT
~~/text/A.txt → #1~~
match & hit
/text/A.txt/v1/s1 → #1, #2
remove an existing entry,
add a new entry
& integrate them

Figure 2: Example of Using BPM (upper: $Case(1)$, lower: $Case(2)$)

Table 1: Summary of Matching/Selecting Algorithms

	FIB	CS		PIT	
		Data	Interest	Data	Interest
(I)	SPM/LPM	EM/-	EM/-	SPM/LPM or others[1]	EM/-
(II)	SPM/LPM	EM/-	EM/-	SPM/-	BPM/optimal
(III)	SPM/LPM	EM/-	EPM/optimal	SPM/LPM or others[1]	EM/-
(IV)	SPM/LPM	EM/-	EPM/optimal	SPM/-	BPM/optimal

[1] FIFO (first in, first out) or all hit are also available.

3 Architecture

Firstly, we describe design principles of our proposed router architecture. In comparison with a fixed-length IP address, a variable-length name address imposes a very high load for lookup. To handle the variable-length name address at line speed, we introduce CAM and a distributed-and-load-balancing Bloom filter (DLB-BF) [18] 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 name lookups 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 1. More specifically, Table 2 shows the 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 the lookup of Interests in PIT as shown in Table 2. For this reason, we assume that there are not any name siblings. Additionally, Binary-CAM (BCAM) can be used instead of Ternary-CAM (TCAM); BCAM is more reasonable than TCAM in 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 (implemented as SPM)	- (implemented as LPM)
Data	PIT	SPM	Longest Match
	CS	EM (implemented as SPM)	- (implemented as LPM)

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 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.

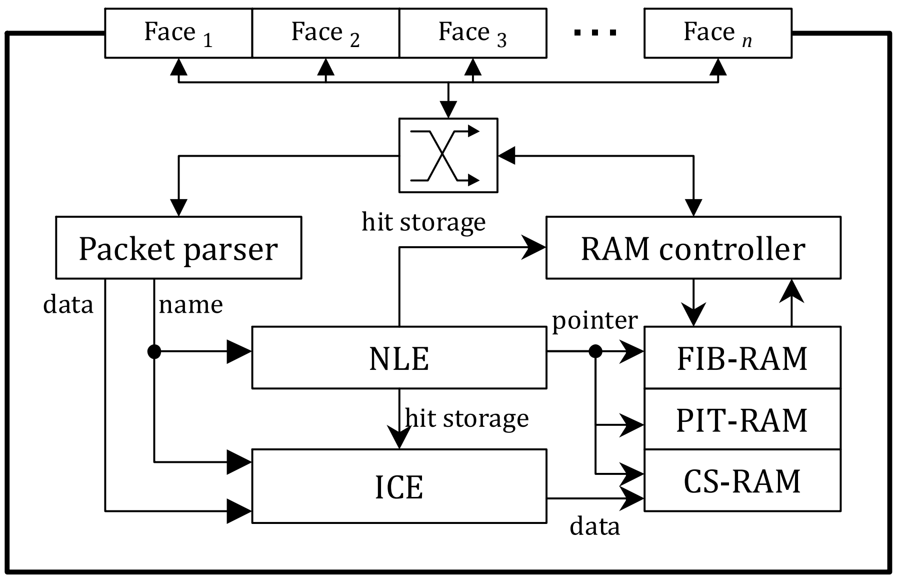


Figure 3: CCN Router Architecture

3.1 Name Lookup Entity

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.

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.

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.

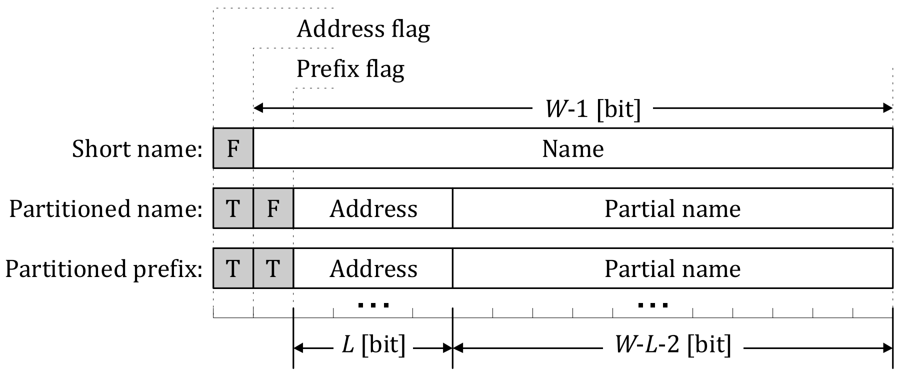
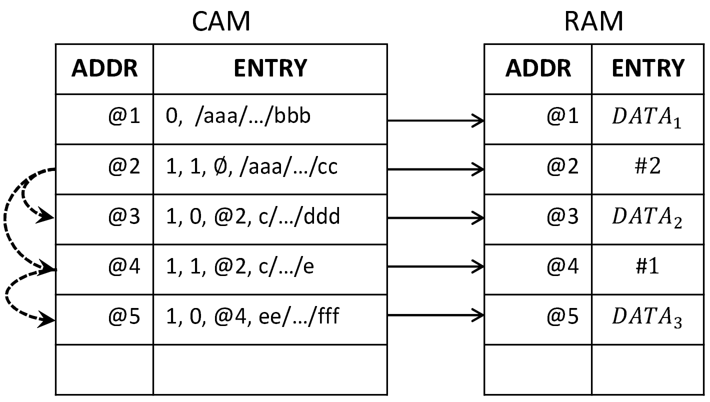


Figure 4: Definition of CAM Entry

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 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 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 of the parent node N_1^B, and



ADDR	ENTRY
@1	0, /aaa/…/bbb
@2	1, 1, ∅, /aaa/…/cc
@3	1, 0, @2, c/…/ddd
@4	1, 1, @2, c/…/e
@5	1, 0, @4, ee/…/fff

ADDR	ENTRY
@1	$DATA_1$
@2	#2
@3	$DATA_2$
@4	#1
@5	$DATA_3$

Figure 5: Example of CAM and RAM Entries in NLE

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.

Note that it is rare to perform the multiple lookups for a long name like N^B and N^C, which cause high latency. In accordance with the fact that 99% of domain names are no longer than 40 bytes [13], almost all of name addresses can be stored in SN by setting $W = 320$. This length of an entry is 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. Algorithm 1 and 2 shows the pseudo code description of this ICE processes.

Algorithm 1 Interest Process in ICE

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

Algorithm 2 Data Process in ICE

```
1: procedure DataProcessInICE($hitCS, name, data$)
2:     if $hitCS = \mathrm{True}$ then
3:         return
4:     end if
5:     $entry \leftarrow ICETable[H(name)]$
6:     $c \leftarrow entry.count$
7:     $n \leftarrow entry.name$
8:     if $c > THRESHOLD\ \&\ n = name$ then
9:         AddCS(name, data)                         ▷ The Data 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).

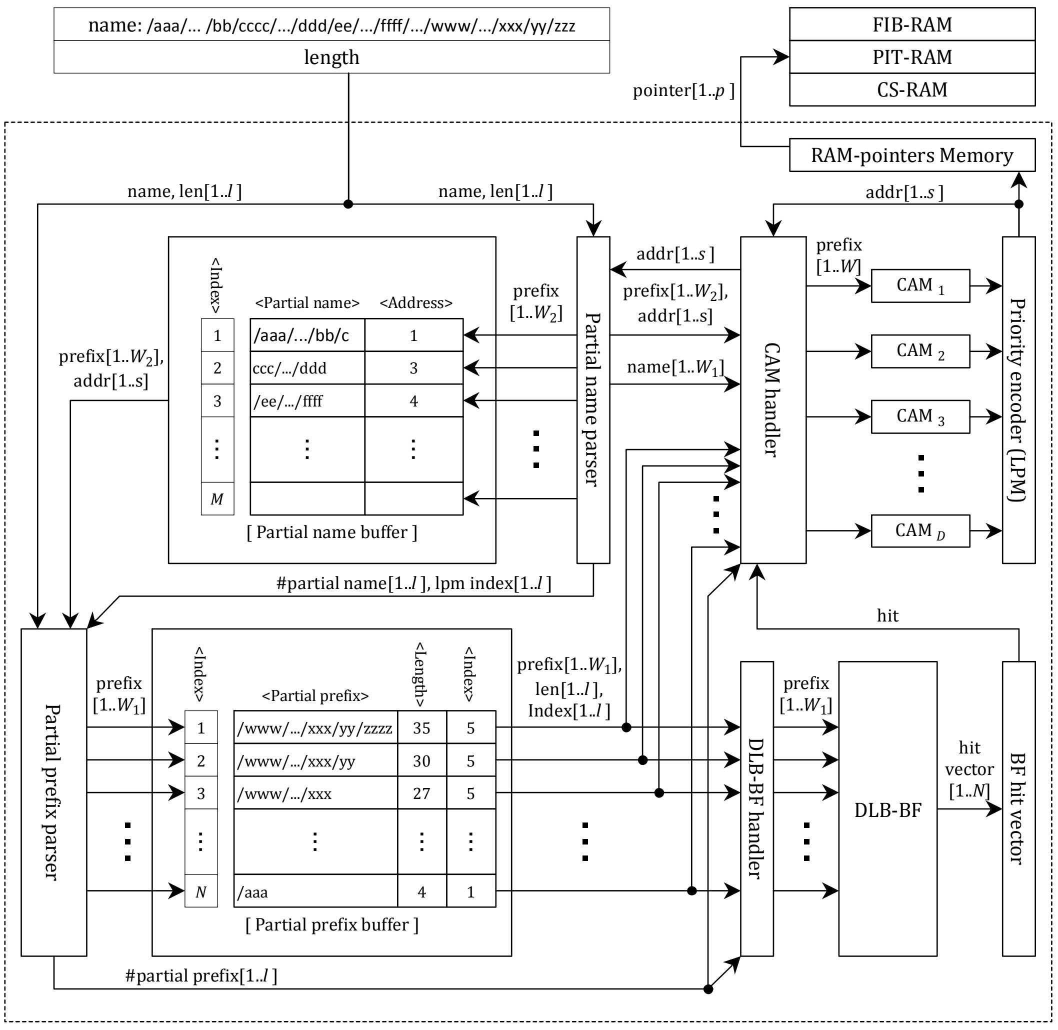


Figure 6: Hardware Design of NLE

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 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 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 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.

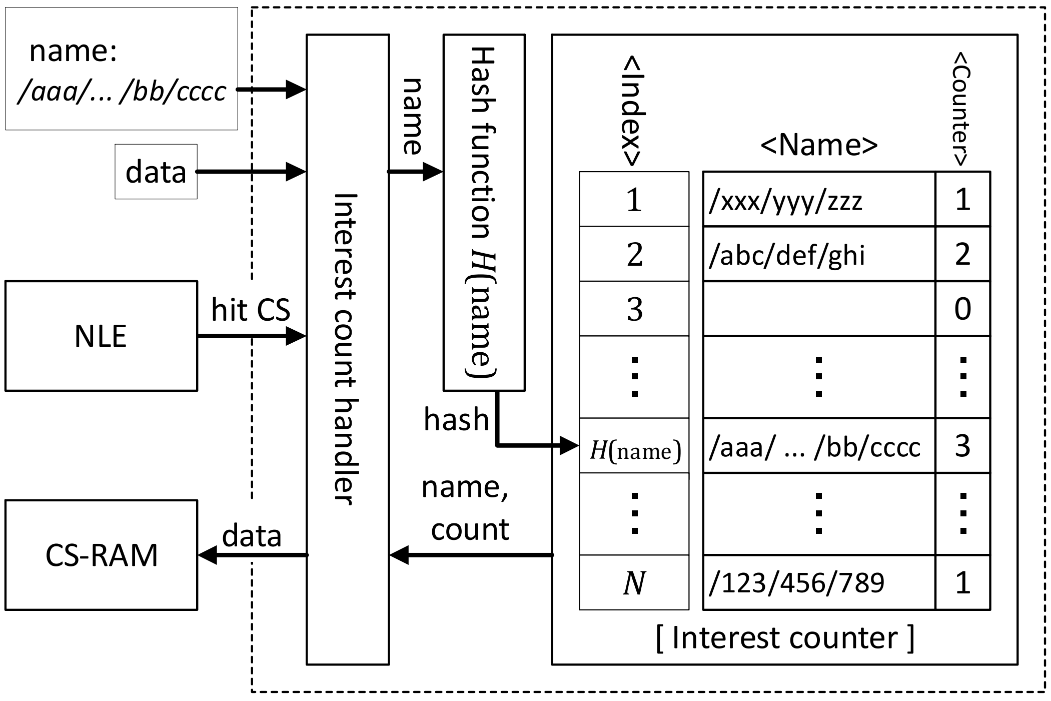


Figure 7: Hardware Design of ICE

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 [19], 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 is summarized in Figure 8. All packets received by the router are transmitted, integrated, cached or satisfied with cached Data according to the flow chart.

NLE consisting of DLB-BF and CAM implements lookup system based on LPM. The case where Interest matches PIT, however, requires to check whether the match is EM since matching and selecting algorithms follow the strategy illustrated in Table 2. Obviously, a simple implementation makes the check possible; we only have to know buffer indexes where the matching prefix is stored. If the prefix's index of both buffers is one, the matching partial prefix is essentially identical to its complete name and the match is identified with EM. Otherwise, the match is non-exact match, therefore, the processes for partial prefixes are repeated.

When Data matches an entry in CS, the process does not stop but continues to run as shown in Figure 8. Although the match appears to be proof that the Data is cached in CS and Interest requesting the Data has been satisfied by the cache, active naming requires the further processes. For example, if CS contains a cache of Data named "`/video/a.mpg/v1/s1`", Interest named "`/video/a.mpg`" mismatches 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 such a case, the Interest named "`/video/a.mpg`" cannot be satisfied.

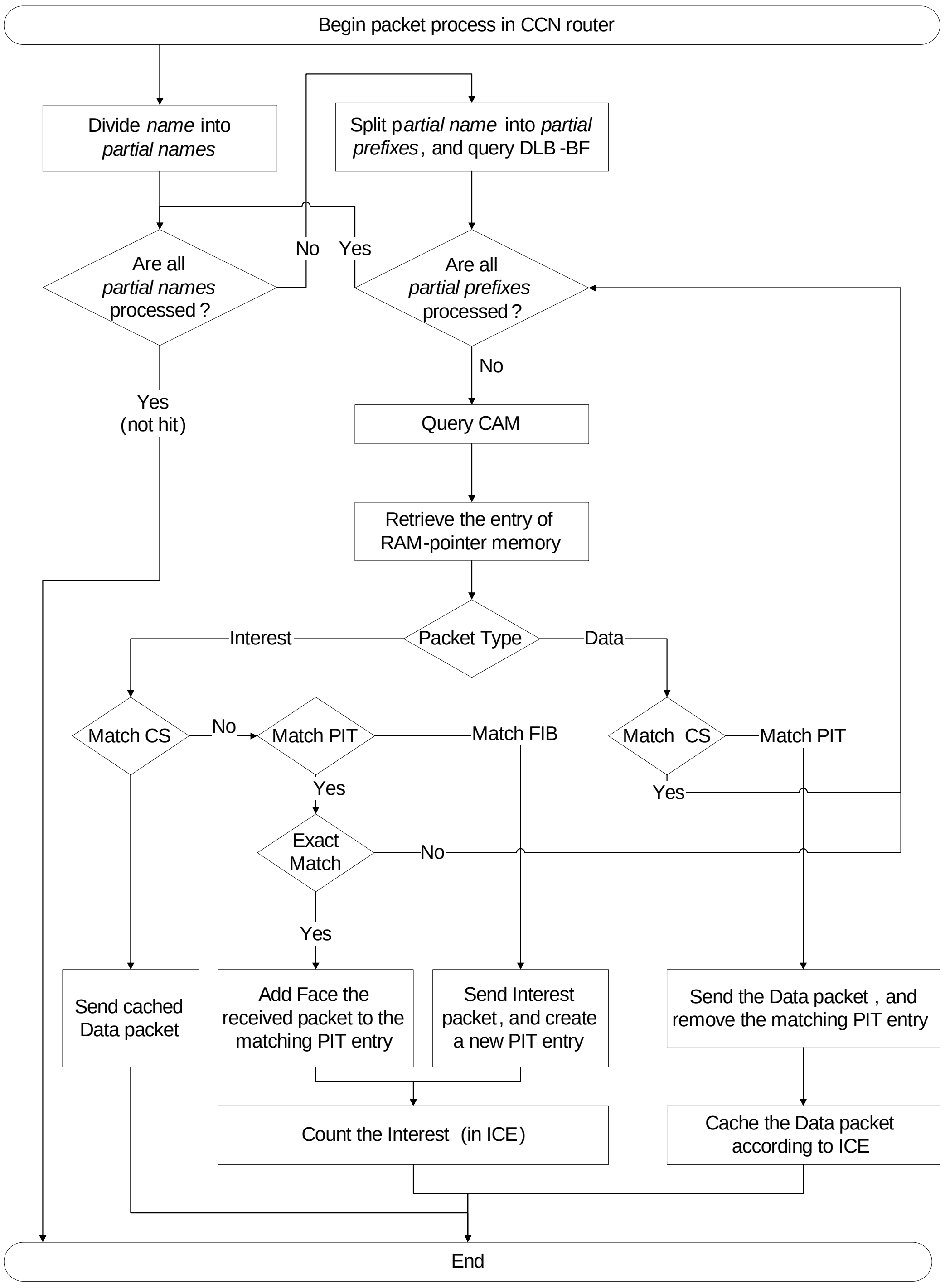

Begin packet process in CCN router
Divide *name* into *partial names*
Split p*artial name* into *partial prefixes*, and query DLB-BF
No
Yes
Are all *partial names* processed?
Are all *partial prefixes* processed?
No
Yes (not hit)
Query CAM
Retrieve the entry of RAM-pointer memory
Interest
Packet Type
Data
Match CS
No
Match PIT
Match FIB
Match CS
Match PIT
Yes
Yes
Exact Match
No
Yes
Send cached Data packet
Add Face the received packet to the matching PIT entry
Send Interest packet, and create a new PIT entry
Send the Data packet, and remove the matching PIT entry
Count the Interest (in ICE)
Cache the Data packet according to ICE
End

Figure 8: Packet Processing in Our Proposed CCN Router

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 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].

Existing lookup mechanisms that weed out a false positive achieve searches per second, throughput, and size of memory shown in Table 3. MATA-NW seems to be fast enough but its throughput is achieved by employing a pipeline using GPU. Although a pipelined process outputs the packets at high-speed, both the pipeline process and GPU make it hard to reduce the latency of each packet process.

Table 3: Performance of Existing Lookup Mechanisms

lookup mechanism	searches per second (3M/10M)[MSPS]	throughput (3M/10M)[Gbps]	memory size (3M/10M)[MB]
Character Trie[12]	3.505/3.172	7.010/6.344	282.21/1,026.34
NCE [13]	5.489/4.017	10.979/8.034	192.58/718.44
ENPT [14]	-/20.67	-/41.34	-/116.02
NameFilter [12]	36.976/37.003	73.952/74.006	64.73/234.27
ATA(200μs latency) [15]	-/6.56	-/13.12	192.97/682.55
MATA(100μs latency) [15]	-/29.75	-/60.50	149.92/490.28
MATA-NW(100μs latency) [15]	-/63.52	-/127.04	149.92/490.28

5.1 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. NLE consists of two buffers, DLB-BF and CAM.

The size of partial name buffer S_N[bit] and the size of 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 of partial name buffer, N is the number of entries of partial prefix buffer, W_1 and W_2 are the bit-lengths of buffer as shown in Figure 6, s is the bit-length of $< Address >$ field, and l is the bit-length of $< Length >$ 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 [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 [18]:

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

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 the 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. 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 [20], the 4.6 GB for the DLB-BF will cost 4600 USD.

The memory required for CAM is the most serious problem because of the limitation of the size. 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 7800 USD.

5.2 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. Table 4 describes the minimum/average/maximum number of read/write access to the memories. In Table 4, α is the false positive probability of 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 $T(m)$ is defined as follows:

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

where n_i is the number of partial prefixes obtained from i-th partial name and N is the number of entries in partial prefix buffer. Table 4 covers three operations: lookup process, add process when the adding entry exists in CAM, and add process when the adding 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 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 access to partial prefix buffer from $S(m)$ to m by parallelizing the process to write partial prefixes. As a consequence, the access times required for 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 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 line speed. These throughputs could be greatly improved by designing mechanisms for concurrent lookups.

Table 4: The Number of Read/Write Accesses to The Memories on Name Lookup Process

		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

6 Conclusion and Future Work

This thesis 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 gave a basic theoretical analysis to evaluate the throughput and cost of the 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 [21]. 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 [19].

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 thesis 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. Additionally, FIB, PIT, and CS requires the policy to manage their entries and its implementation such as timeout, retransmission, and cache replacement algorithms. Ultimately, this will result in practical network-level evaluation and a realistic analysis of network bottlenecks.

Acknowledgment

First, I would like to express my sincere gratitude to my supervisor, Professor Masayuki Murata of Osaka University, for his continuous support and expensive advice throughout my studies, and providing me this precious study opportunity in his laboratory.

I especially would like to express my deepest appreciation to Professor Shingo Ata of Osaka City University, for his elaborated guidance and invaluable firsthand advice. All works of this thesis would not been possible without his support.

There was suggestive advices of Professor Kazunari Inoue of Nara National College of Technology, so I am able to complete the discussion of hardware in this thesis.

Furthermore, I must acknowledge Associate Professor Shin'ichi Arakawa, Assistant Professor Yuichi Ohsita of Osaka University for their valuable comments and suggestions on this study.

I heartily thank Ms. Kazama, secretary's doing a lot of help, and the encouragement at times. I show will of thanks to here.

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] 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.

[3] 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.

[4] 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

[5] 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.

[6] 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.

[7] “CCNx,” PARC, 2013. [Online]. Available: http://www.ccnx.org/

[8] 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.

[9] 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

[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] 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," in *Proceedings of the ACM SIGCOMM 2013*, August 2013, pp. 99–110.

[18] 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.

[19] 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.

[20] 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.

[21] “netcraft,” December 2013. [Online]. Available: http://www.netcraft.com/