A Reality Check for Content Centric Networking

Diego Perino
Bell Labs, Alcatel-Lucent, Villarceaux, France
diego.perino@alcatel-lucent.com

Matteo Varvello
Bell Labs, Alcatel-Lucent, Holmdel, USA
matteo.varvello@alcatel-lucent.com

ABSTRACT

Content-Centric Networking (CCN) is a novel networking paradigm centered around content distribution rather than host-to-host connectivity. This change from *host-centric* to *content-centric* has several attractive advantages, such as network load reduction, low dissemination latency, and energy efficiency. However, it is unclear whether today's technology is ready for the CCN (r)evolution. The major contribution of this paper is a systematic evaluation of the suitability of existing software and hardware components in today's routers for the support of CCN. Our main conclusion is that a CCN deployment is feasible at a Content Distribution Network (CDN) and ISP scale, whereas today's technology is not yet ready to support an Internet scale deployment.

Categories and Subject Descriptors

B.3.2 [**Design Styles**]: Cache memories, Mass storage; C.2.1 [**Network Architecture and Designs**]: Network communications

General Terms

Design, Verification

Keywords

CCN, Router, Architecture

1. INTRODUCTION

The Internet is founded on a host-centric principle: machines are uniquely identified with an IP address and communication is possible between any pair of machines. Content dissemination, i.e., the distribution of a piece of information from one source to multiple consumers, is not a feature of today's communication model, but it can be easily supported because of its flexibility. For example, CDNs efficiently tackle the content distribution problem by redirecting user requests for content located at a given host to a closeby data center.

Recently, we have observed an explosion of traffic associated with content production and dissemination.This traffic is mostly generated by popular applications, such as Netflix, Facebook, and YouTube. This new trend has two main consequences: (1) from a business perspective, there is clear increase in the number of CDN providers and offers, (2) from a research perspective, a novel communication paradigm is envisaged: *Content-Centric Networking*.

Content-Centric Networking (CCN) aims to replace *machines* with *content* in the networking communication model. The fundamental principles of CCN have been investigated by a number of researchers over the last decade. Populard esigns are TRIAD [15], ROFL [11], DONA [17], and more recently NDN [3, 16]. Key CCN features are: (1) content items have an identifier and not content hosts, (2) routing leverages content identifiers and not host identifiers for forwarding operations, (3) content packets can be transparently cached, and (4) native support for multicast.

The CCN (r)evolution requires severe changes to today's routers. For example, high speed hardware and software are required to support name-based data forwarding and packet-level caching. Moreover, shifting the address space from one billion IPs to at least one trillion content names [7] causes a neat increase of the routing state to be stored at content routers. However, allowing routers to serve content from a local cache can potentially alleviate the frequency of forwarding operations. Generally, the interaction between caching and forwarding is still poorly understood, and it is not clear whether today's technology can sustain the described additional operations.

Arianfar et al. [9] are the first to explore some practical considerations in the design of content routers. Specifically, they analyze how to extend today's routers to support packet-level caching. Our work is the logical continuation of this work. Accordingly, we aim to answer the following question: *is today's router technology (both hardware and software) ready to support the CCN (r)evolution?* In order to answer this question, we leverage NDN as a design example. Our motivations are as follows: (1) NDN is founded on conventional routing and it is so largely compatible with IP routing, (2) NDN is a complete CCN proposal including naming, routing, transport and caching, and (3) a full-fledged prototype is publicly available. Nevertheless, we plan to extend our analysis to other designs as future work.

In this work, we provide an abstract model of a generic content router component. Then, we detail the processing performed by each content router component as well as their interconnections. Our analysis indicates that the additional cost and operations to support content caching and data forwarding, can be afforded by today's technology. However, today's hardware and software can only support a fraction of the routing state required for forwarding operations at Internet scale. Nevertheless, by reducing the scope of a CCN deployment, i.e., from Internet scale to CDN or ISP scale, today's routers could be easily extended to become content routers.

ICN'11, August 19, 2011, Toronto, Ontario, Canada.

Technology	Access time [ns]	Max. size	Cost [$/MB]	Power [W/MB]
TCAM	4	∼20 Mb	200	15
SRAM	0.45	∼210 Mb	27	0.12
RLDRAM	15	∼2 Gb	0.27	0.027
DRAM	55	∼10 GB	0.016	0.023
High-speed SSD	1,000	∼10 TB	0.03	0.00005
SSD	10,000	∼1 TB	0.003	0.00001

Table 1: Summary of memory technologies.

Policy	Bits/packet	Random Read	Random Write
HC-basic	40	1	1
HC-log	72	1	0
HC-log + LRU	136	1	0

Table 2: Summary of caching designs ; H-bit=40.

2. BACKGROUND

This section serves as a background for a clear understanding of the paper. We start by overviewing the NDN design. Then, we focus on hardware and software technologies that we leverage for the content router design.

2.1 Named Data Networking

NDN uses hierarchical human-readable names to address content items. A content name has several *components* delimited by a character, e.g., `/ICN/PAPERS/PaperA.pdf`. Content is requested using an *Interest* packet that contains the name of the desired component-wise content. An Interest packet propagates toward potential data using longest prefix match. The Interest propagation leaves a trail of "bread crumbs" that a matching *Data* packet follows to reach back the original requester(s), naturally realizing multicast. Content routers cache Data packets for later transmission.

2.2 Hardware

We now provide an overview of current memory technologies that may be employed for the deployment of a content router (Tab. 1). Note that memory access is the main bottleneck of today's hardware router design, while packet processing takes only few clock cycles.

Ternary Content Addressable Memories (TCAMs) minimize the memory accesses required to locate an entry by comparing it against all memory words in one clock cycle. Today's largest commercially available single-chip CAMs have a size of 18 Mbits [19].

Static Random-Access Memories (SRAMs) are volatile semi-conductor memories that do not need to be periodically refreshed. SRAMs are typically implemented on-chip. Modern SRAMs can transfer up to 4 words of data in a single access. The largest single-chip SRAM has a size of 72 Mbits, and word size ranges from 9 to 36 bits [6].

Dynamic Random-Access Memories (DRAMs) are volatile memories that need to be periodically refreshed. In today's routers, DRAMs are typically installed off-chip. The most recent DRAM can read up to 8 words of data in a single access. The largest single chip DRAM implementation has a size of 4 Gbits, and word size ranges from 4 to 16 bits. *Reduced Latency DRAMs (RLDRAMs)* provide faster access time than DRAMs but can only read up to 4 words of data in a single access. The largest single chip RLDRAM has a size of 576 Mbits, and word size ranges from 8 to 32 bit.

Flash-based Solid-State Drives (SSDs) provide persistent data storage similarly to traditional hard disks. However, SSDs are based on microchips, and have lower access time compared to hard disks. SSDs have limited lifetime and their reliability largely decreases over time. SSD size ranges from hundreds of GBytes to TBytes [2, 5, 6].

2.3 Software

We now focus on software solutions for caching and forwarding functionalities in a content router. Our terminology relates to a NDN-like environment.

Caching – Several designs have been recently proposed for cache storage systems (Tab. 2), which consist of a *packet store*, where Data packets are contained, and of an *index table*, that keeps track of Data packets in the packet store. We now describe each design in detail.

HC-basic [10] treats the packet store as a fixed size hash-table. A Data packet name is hashed and the packet is inserted in the location corresponding to the hash value modulo the number of *bins*, i.e., the addressable elements of an hash-table. Differently from the original HC-basic design, we consider an index table which is addressed similarly and stores H-bit hash value for each Data packet name[1]. In the following we simply refer to this approach as HC-basic. It follows that a request generates a read operation from the packet store only in case of index hit or false positive (2^{-H}). A write operation to the packet store is random because of the hash function. To reduce the probability of conflicting packets, a bin could store multiple items (*HC-SetMem* [10]), or multiple hash functions could be employed.

HC-log [10] allows sequential packet writing to the packet store similarly to high performance caches. The index table stores H bits per Data packet plus four additional bytes to address the packets in the content store. A similar approach is proposed in *FAWN* [8]. The deployment of simple replacement policies, such as LRU, requires additional 64 bits for pointers, and extra accesses to index for pointer update.

Packet Forwarding – Content routers use Longest Prefix Match (LPM) for Interest packet forwarding. LPM can be implemented either in hardware using a TCAM or in software using a multi-bit trie [13, 14] or Bloom filters [12]. Solutions based on software are preferred because of TCAM's excessive cost and power consumption. We only focus on LPM using Bloom filters as it is the most promising approach [3].

Dharmapurikar et al. [12] propose to perform IPv4 LPM by using a combination of Bloom filters and an hash-table. A Bloom filter is a data structure for membership queries with no false negatives probability and tunable false positive probability of about $(1-e^{\frac{-NK}{M}})^k$, where M is the Bloom filter size, N is the number of indexed elements and K is the number of hash functions used. The false positive probability at its optimal point implies $K = \frac{M}{N} \cdot ln2$ with $\frac{M}{N} \simeq [5-20]$ [12, 14]. In the following, we describe how this approach works assuming LPM over content names.

Assuming content names formed by B components, B Bloom filters are stored on on-chip memory, each corresponding to a unique prefix length. Each Bloom filter is populated with prefixes having the same length. A off-chip hash-table is used to store for each prefix the next hop information, i.e., the interface where to forward an Interest packet. Upon reception of an Interest packet whose content name has B components, we query the B Bloom filters associated to each possible prefix length. Given Bloom filters are stored on-chip all Bloom filters can be looked up in a single clock cycle. As a result, we obtain a vector of possible prefix matches. We then query the hash-table starting from the longest prefix match in order to verify eventual false positives, and retrieve the interface associated to the longest prefix match. The average number of off-chip

[1]The index table and the packet store can also be considered as arrays and addressed using a substring of the Data packet name [9].

memory accesses, E_{avg}^{FIB}, along with the access time, L_{avg}^{FIB}, are formulated as follows:

$$E_{avg}^{FIB} = B \times \left(\frac{1}{2}\right)^{\frac{Mln2}{N}} + 1$$
$$L_{avg}^{FIB} = 1 \cdot L_{on} + E_{avg}^{FIB} \cdot L_{off} \simeq E_{avg}^{FIB} \cdot L_{off} \quad (1)$$

where L_{on} and L_{off} are on-chip and off-chip memory access time, respectively. Note that the access time for the on chip memory is negligible as it is two orders of magnitude shorter than the off chip memory (Tab. 1).

3. CONTENT ROUTER MODELING

This section focuses on the modeling of a content router. We only consider the *data plane*, i.e., the part of the content router that deals with Data and Interest packets. We do not consider the *control plane*, i.e., the part of the content router responsible to populate the routing state across the network, but we plan to address this limitation as a future work.

Three main components form a content router: *Forwarding Information Base (FIB)*, *Content Store (CS)* and *Pending Interest Table (PIT)*. The FIB is a table that associates prefixes of content names to one or multiple next hop routers. The CS plays the same role of a buffer in an IP router. However, it also stores Data packets after they have been forwarded so that they can serve future requests. The PIT keeps track of the content items that have recently been requested and not yet served. Requests for the same content are aggregated within a single PIT entry.

Upon reception of an Interest packet on an interface I, a content router checks for content availability in its CS; if the content is available, the CS sends the requested data back on I. Otherwise, the content router checks in the PIT whether this content has been already requested upward; if an entry is found in the PIT, this is updated in order to track that interface I is waiting for this content. If no PIT entry is found, a new entry is created and the Interest packet is forwarded to one or multiple interfaces determined via longest prefix match on the content name prefixes stored in the FIB.

When a Data packet is received, a content router checks for pending requests in its PIT. If a pending request is found, the Data packet is stored in the CS and forwarded towards all requesting interfaces as listed in the PIT. If no matching PIT entry is found, the Data packet is discarded. In fact, either another Data packet already consumed the PIT entry or an "unsolicited" Data packet was received.

We model each component of a content router as described in Fig. 1. λ_f^{in} denotes the average packet arrival rate at component f; μ_f denotes the average rate of packets that do not require further processing after component f; λ_f^{out} denotes the rate of packets that are forwarded to the next component. A component may not be able to process all λ_f^{in} incoming packets, as a consequence only a fraction p_f is served.

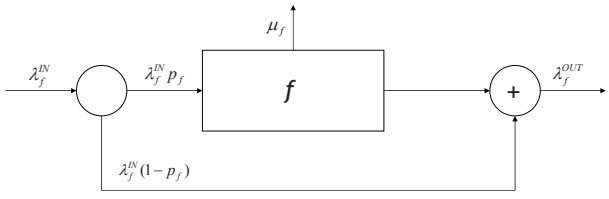


Figure 1: Generic model of a content router component.

3.1 Content Store

The CS can be implemented using one of the caching designs presented in Sec. 2. The CS is the first content router component being accessed when an Interest packet is received. The following operations are performed: (1) exact match on content name in the index table, and (2) update of the index table in case of match; this operation is not always performed as it depends on the replacement policy adopted, e.g., RANDOM, LRU and LFU.

We call *CS hit probability*, denoted by p_{CS}^{hit}, the fraction of Interest packets that are satisfied by a Data packet stocked in the content store and thus not forwarded to the PIT. The rate of Interest packets that do not require further processing within a content router is $\mu_{CS} = p_{CS}^{hit} \cdot \lambda_{CS} \cdot p_{CS}$; whereas, the rate of Interest packets forwarded to the PIT is $\lambda_{CS}^{out} = (1-p_{CS}^{hit}) \cdot \lambda_{CS}^{in} \cdot p_{CS} + \lambda_{CS}^{in} \cdot (1-p_{CS})$.

The CS is the last content router component to be accessed (right after the PIT) when a Data packet is received. It follows that $\lambda_{CS}^{in} = \lambda_{PIT}^{out}$. The following operations are performed: (1) exact match on full name lookup in the index table, (2) storage of the Data packet in the packet store and update of the index table if the Data packet is not yet stocked. λ_{CS}^{out} packets are not added to the packet store: a fraction of them is already present in the CS, while $\lambda_{CS}^{in}(1-p_{CS})$ are not processed by the component. These packets are not moved to any other component.

When a content router interface sends out a Data packet, it reads it from the content store. For this purpose, we assume each interface keeps a queue of pointers to Data packets in the packet store that are waiting for transmission. The overall access rate to the packet store is therefore determined by the sum of Data packet reception and transmission rate.

3.2 Pending Interest Table

The PIT can be implemented using an indexing scheme à la HC-basic (Tab. 2). However, the PIT does not include a store and the index contains two additional bytes per router interface which has requested a given packet[2].

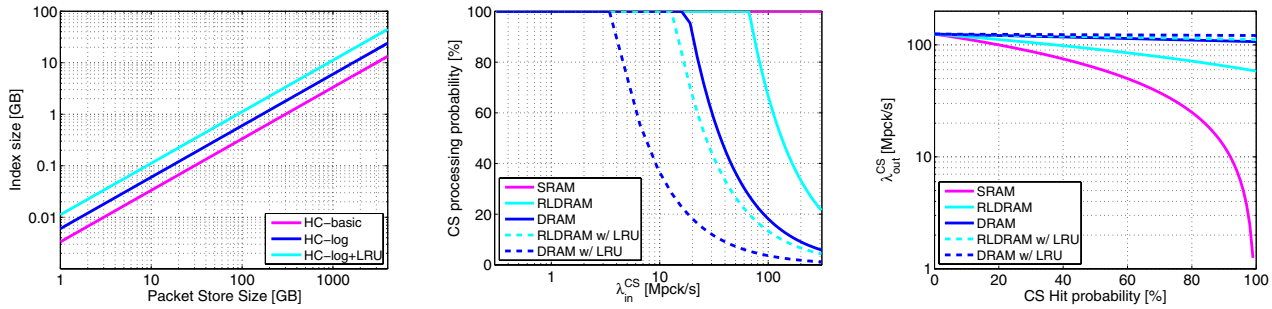
An *Interest* packet is forwarded to the PIT if not satisfied by the CS, i.e., $\lambda_{PIT}^{in} = \lambda_{CS}^{out}$. Then, the following operations are performed: (1) exact match on content name lookup in the index table, (2) update of the index table in case of positive exact match, or creation of a new entry otherwise.

We call *PIT hit probability*, denoted by p_{PIT}^{hit}, the fraction of Interest packets for which a request for the same content has already been issued. In this case, the Interest packet is not forwarded to the FIB. It follows that the rate of Interest packets that do not require further processing within a content router is $\mu_{PIT} = p_{PIT}^{hit} \cdot \lambda_{PIT} \cdot p_{PIT}$. Conversely, the rate of Interest packets forwarded to the FIB is $\lambda_{PIT}^{out} = (1 - p_{PIT}^{hit}) \cdot \lambda_{PIT}^{in} \cdot p_{PIT} + \lambda_{PIT}^{in} \cdot (1 - p_{PIT})$.

When a *Data* packet is received at a content router the PIT is the first component to be accessed and the following operations are performed: (1) exact match on content name lookup in the index table, and (2) removal of the entry from the index table in case of positive match.

We call PIT unsolicited probability, denoted by p_{PIT}^{u}, the fraction of incoming Data packets for which a PIT entry does not exist. It follows that the rate of Data packets that are not forwarded to the CS is $\mu_{PIT} = p_{PIT}^{u} \cdot \lambda_{PIT} \cdot p_{PIT}$. Note that μ_{PIT} also corresponds to the rate of incoming Data packets that are dropped by the content router because "unsolicited". Conversely, the rate of Data packets forwarded to the CS is $\lambda_{PIT}^{out} = (1-p_{PIT}^{hit}) \cdot \lambda_{PIT}^{in} \cdot p_{PIT} + \lambda_{PIT}^{in} \cdot (1 - p_{PIT})$. λ_{PIT}^{out} also corresponds to the rate of incoming Data packets that are forwarded towards requesting interfaces.

[2]As in [14], we use 16 bits for next hop information.



(a) Index size as a function of memory size. (b) p_{CS} as a function of λ_{CS}^{in}. $p_{CS}^{hit} = 0.65$. (c) λ_{CS}^{out} for several p_{CS}^{hit}

Figure 2: CS Analysis ; Interest packet size = 40 Bytes ; Data packet size = 1,500 Bytes.

We call RTT_{avg} the average amount of time a request remains in the PIT, i.e., the time between the creation of an entry and the arrival of the corresponding Data packet. In the worst case where Interest packets cannot be aggregated, i.e. every Interest packet carries a request for a different Data packet, the average number of PIT entries can be easily derived as $\lambda_{PIT}^{in} \cdot RTT_{avg}$. In a real deployment, a timeout would be associated to every PIT entry in order to bound the maximum time-frame a request is pending and consequently the PIT size.

3.3 Forwarding Information Base

The FIB consists of bloom filters stored on on-chip memory, as described in Sec. 2, and of an off-chip hash-table that can be designed using an indexing scheme à la HC-basic (Tab. 2) plus two additional bytes for next hop information[2].

The FIB is the last content router component reached by an Interest packet, i.e., $\lambda_{FIB}^{in} = \lambda_{PIT}^{out}$. When an interest packet reaches the FIB, a longest prefix match (LPM) on the content name prefixes stored in the FIB is performed. Although the FIB is the last component of a content router, i.e., $p_{FIB} = 1$, we anticipate a possibility to skip LPM operations by flooding all the interfaces or a subset of them. Thus, $\lambda_{FIB}^{out} = \lambda_{FIB}^{in} \cdot (1 - p_{FIB})$ is the rate of flood operations to be performed and $\mu_{FIB} = \lambda_{FIB}^{in} \cdot p_{FIB}$ is the fraction of Interest packets forwarded upward.

4. EVALUATION

This section focuses on content router performance evaluation. For every component, we analyze storage requirements and access latency constraints related to the processing of incoming Interest and Data packets. We then discuss the design of edge and core content routers along with an estimation of their cost and energy consumption.

We consider routers with up to 64,000 interfaces[2] with line-speed in the range 100 Mbps - 100 Gbps, i.e., from PC to high-speed router line cards. As in [9], we assume 40-Byte Interest packets and 1500-Byte Data packets. For CS and PIT indexing we use an H-bit value of 40 bits, i.e., the false positive probability is lower than 1% even for large tables (10^{10} entries).

Content names can potentially be formed by an infinite number of components B. In order to derive a reasonable value of B for our analysis, we analyze 10 hours web traffic collected at a wireless service provider [18]. This trace spans 10,000 unique users and 15,000 unique URLs. We find that 99.9% of the URLs are composed by less than 30 components, whereas the longest URL has about 70 components. For this reason, unless otherwise specified, we set $B = 30$.

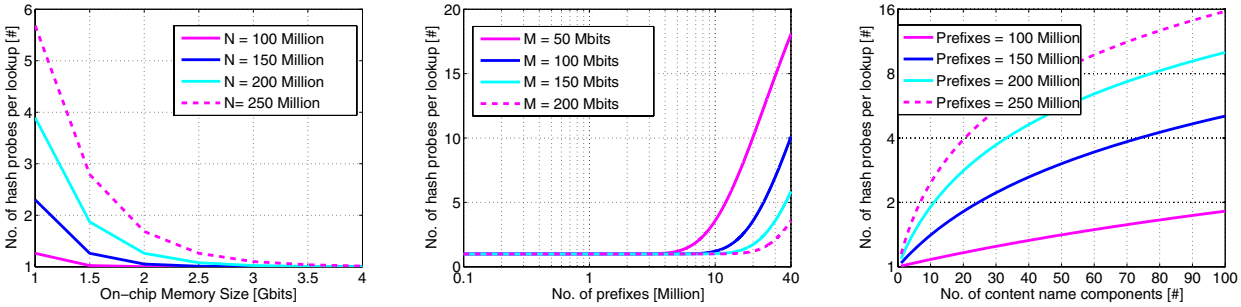
We also need some indications about the size of the content name space along with how much aggregation is possible. Google reports that the web contains more than 1 trillion webpages [7]; as for January 2011, these webpages are mapped to about 280 million globally unique and routable hostnames, although only 86 million are active [4]. In case, of maximum aggregation, a high-end router would only be aware of the hostname component of content name, i.e., about 280 million prefixes.

4.1 Content Store

We first analyze the memory required to store the index table as a function of the packet store size assuming indexing techniques à la HC-log, HC-basic and HC-log+LRU (Fig. 2(a)). Fig. 2(a) shows that a packet store of 10 GBytes, which can easily be implemented using a DRAM (Tab. 2), requires an index table with size between 30 and 100 MBytes according to the indexing scheme used. Such small amounts of memory can easily be stored on SRAM or RLDRAM. However, as the packet store size grows over 1 Terabyte, which requires an SSD-based implementation, the index table associated grows over several GBytes and thus requires to be stored on a DRAM memory.

We now analyze the impact of the average Interest packet arrival rate at the CS, λ_{CS}^{in}, on the fraction of Interest packets that can be processed (Fig. 2(b)). We consider different memories (SRAM, DRAM and RLDRAM) along with random and LRU replacement strategies. We only focus on the Interest packets as the operations performed by the CS on the Data packets are negligible, e.g., they account for less than 5% of the total operations in the worst case. Fig. 2(b) shows that an SRAM-based index table implementation ensures that all incoming Interest packets are correctly processed even for the highest value of λ_{CS}^{in}, i.e., 300 Mpck/s or 100 Gbps, with LRU replacement. Conversely, RLDRAM and DRAM-based index table implementations can only sustain a maximum rate of 75 and 18 Mpck/s respectively, that reduces to 15 and 3.5 Mpck/s with LRU replacement, without skipping any Interest packet processing. Then, as λ_{CS}^{in} increases, the fraction of Interest packets that can be served decreases and gets close to 0 for most technologies.

Finally, we analyze the impact of the CS hit probability, p_{CS}^{hit}, on the rate of packets forwarded to the PIT, $\lambda_{CS}^{out} = \lambda_{PIT}^{in}$, assuming $\lambda_{CS}^{in} = 125$ Mpck/s (Fig. 2(c)). Intuitively, as the probability to find a requested content item in the CS increases, λ_{CS}^{out} decreases. However, this holds only when the memory used for the index table is fast enough to process all incoming packets. Fig. 2(c) shows that when the CS hit probability is very low, even a DRAM-based index table implementation ensures a reduction on λ_{CS}^{out}. However, as the CS hit probability increases there is a clear need to move



(a) $B = 30$; $N = [100, 150, 200, 250]$ million. (b) $B = 30$; $M = [50, 100, 150, 200]$ Mbits. (c) $B = 1 - 100$; $N = [100, 150, 200, 250]$ million

Figure 3: FIB Analysis ; Avg. number of hash probes per lookup.

towards faster memories, e.g., RLDRAM or SRAM. For example, if we consider a CS hit probability of 90%, e.g., similar to what we observe in today's CDNs, an SRAM reduces λ_{CS}^{out} seven times more than an RLDRAM.

4.2 Pending Interest Table

The PIT is implemented using an indexing à la HC-basic whose performance has been previously analyzed. Here, we evaluate the memory size required to store the PIT. Although not shown due to space limitations, we find that even in the worst case, i.e., each request generates a new PIT entry and $\lambda_{PIT}^{in} = \lambda_{CS}^{in}$, if we consider $RTT_{avg} = 80$ ms [1] and a line-speed of 1-100 Gbps the average PIT size is 14 Mbits to 1.4 Gbits. The PIT can therefore be easily stored on RLDRAM chips but is too large for SRAM memory.

4.3 Forwarding Information Base

To start with, we analyze the evolution of the average number of lookups in the off-chip memory (DRAM or RLDRAM) as a function of the on-chip memory (SRAM) size M (Fig. 3(a)). We assume Internet scale values for total content name prefixes, i.e, $N = [100, 150, 200, 250]$ million. Given for Bloom filters it holds $\frac{M}{N} \simeq [5 - 20]$ (Sec. 3), we vary M in the range 1-4 Gbits. Fig. 3(a) shows that as M increases, the number of hash probes per lookup tends to one. For example, if we consider 100 million prefixes, i.e., the number of active prefixes in today's Internet, about 1 Gbit of on-chip memory is required to achieve a single lookup on the off-chip memory. Unfortunately, this value is too high for today's standard; 200 Mbits is a reasonable value, e.g., the amount available in current NetFPGA.

We now analyze the impact of the prefix set size N on the average number of lookups in the off-chip memory; we assume standard on-chip memory in the range from 50 to 200 Mbits. Fig. 3(b) shows that 40 million prefixes can be sustained at maximum; when N = 40 million, the average number of lookups in the off-chip memory varies between 4 and 18 as the memory size grows from 50 to 200 Mbits. In order to achieve a single lookup in the off-chip memory, a maximum of 20 million prefixes can be supported assuming 200 Mbits on-chip memory. Note that 20 million prefixes equals 10% of today's hostnames and 25% of the active ones.

Figure 3(c) shows the impact of the content name lenght B on the average number of lookups in the off-chip hash-table. We vary B in the range 1-100 and consider $N = [100, 150, 200, 250]$ million. We assume an on-chip SRAM of 1 Gbps. Fig. 3(c) shows that as B increases, the number of hash probes per lookup becomes very large. This can reduce the fraction of packets that are processed by the FIB, increasing the rate of flood operation. On the contrary, if we consider short content names, e.g., $B < 10$, the number of hash probes per Interest packet gets close to 1. This result indicates that a regulation on the content name length would be very beneficial to CCN.

Finally, we analyze the impact of the average Interest packet arrival rate at the FIB, λ_{in}^{FIB}, on the memory size M. Fig. 4 plots M as a function of λ_{in}^{FIB} when N equals 5 and 20 million and the off-chip memory is implemented using a DRAM and a RLDRAM. Fig. 4 shows that a DRAM-based implementation of the off-chip memory allows the FIB to sustain a maximum λ_{in}^{FIB} of about 18 Mpck/s; this value goes up to 54 Mpck/s when using a RLDRAM. Fig. 4 also shows that M increases with λ_{in}^{FIB}; for example, when N = 20 million 300 and 450 Mbits of on-chip memory are required to support 18 and 54 Mpck/s, respectively.

Finally, we discuss the off-chip memory dimensioning. In a realistic scenario, we have M=200 Mbits, N=20 million and a maximum tolerable λ_{in}^{FIB} of 50 Mpck/s, assuming RLDRAM for the off-chip memory. In a classic HC-basic implementation, the off-chip memory occupies just 140 MBytes.

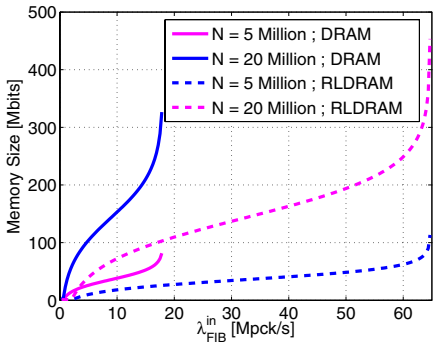


Figure 4: FIB Analysis ; Memory size ; $N = [5, 20]$ million ; DRAM and RLDRAM.

4.4 Global Content Router

We now discuss the design of a backbone and edge content routers. Given multiple designs are possible, we present two solutions that minimize cost and power consumption.

As an edge router, we consider a Cisco 7507; this router has five Gigabit Ethernet line cards and a peak power consumption of 400 W. Under CCN premise, the five line cards generate an overall Interest rate of 15 Mpck/s. The packet store can be implemented

using a 1-TByte high speed SSD (similarly to today caches) as the overall access rate is of 0.417 Mpck/s. It follows that the index table with HC-log indexing can be implemented on a 6-Gbyte DRAM, which easily sustains an Interest rate of 15 Mpck/s (Fig. 2(b)). The average PIT size is 70 Mbits and the table should be implemented in a RLDRAM chip to sustain a rate of $\lambda_{PIT}^{in} = 15$ Mpcks/s. Finally, we use 200-Mbit SRAM for the on-chip FIB memory, i.e., the amount available in current NetFPGA. It follows the FIB off-chip memory requires 140 MBytes and can be implemented using a single DRAM chip. Such a configuration allows to perform LPM on about 20 million prefixes at a maximum speed of 15 Mpck/s (Fig. 4). A rough estimate of the additional cost and energy consumption to extend an edge router to CCN functionalities would be of 195 W and 31,000 $, respectively. Remark that, 30,000 $ of this additional cost are due to high-speed SSD; reducing the Data packet rate by a factor of ten would cut the SSD price by the same factor, i.e., more reasonable 3,000 $.

As a backbone router, we consider a Cisco CRS 1 series router with eight line cards at 40 Gbps with peak power consumption of 4,834 W. Under the CCN premise, eight line cards at 40 Gbps each would generate an overall Interest packet rate of about 1 Gpck/s, while the overall access rate to the packet store would be of 0.026 Gpck/s. Thus, we need to deploy a packet store per line card and to use 10-GByte DRAM memory (Fig. 2(a)). Each 10-GByte packet store requires an index table of 266 Mbits duplicated in two RLDRAM chips in order to sustain 40 Gbps or 120 Mpck/s (Fig. 2(b)). 560 Mbits of PIT per line card are required implemented using SRAM memory. At a core router, about 250 million content name prefixes should be stored (Sec. 4.3); it follows that a 4 Gbits on-chip SRAM[3] (Fig. 3(a)) and about 1.5 GBytes off-chip RLDRAM memory should be employed. Under this configuration the FIB can support up to $\lambda_{FIB}^{in} = 66$ Mpck/s; in order to support the peak Interest rate of 120 Mpck/s the CS and the PIT should serve half of the incoming Interests, or the off-chip FIB should be duplicated. A rough estimate of the additional cost and energy consumption to extend a core router to CCN functionalities would be of 3,302 W and 130,000 $.

5. CONCLUSIONS AND FUTURE WORK

Content-centric networking (CCN) is a novel networking paradigm that centers around content dissemination. In the past, CCN concepts have been extensively analyzed, and many interesting designs have been proposed. However, the practical impact of a CCN (r)evolution on today's routers is an area that has not been fully explored yet. The major contribution of this paper is a reality check for CCN.

We provide models that describe the design of each content router component. Then, we analyze candidate hardware and software solutions for each component. We find that today's technology is not ready to support an Internet scale CCN deployment, whereas a CDN or ISP scale can be easily afforded.

As a future work, we aim to derive a better understanding of how the described results would change when considering the interconnection of multiple routers as well as a real traffic component. Another interesting avenue of future work consists of leveraging the tools introduced in this paper to compare several CCN designs. The final goal would be to give recommendations on CCN designs at different networking scales, e.g., Internet, ISP and CDN.

[3]Note that this value is too high for today's standard.

Acknowledgments

This work has been partially funded by the French national research agency (ANR), CONNECT project, under grant number ANR-10-VERS-001.

6. REFERENCES

[1] Meridian. `http://www.cs.cornell.edu/`.
[2] Micron. `http://www.micron.com`.
[3] Named data networking. `http://www.named-data.net/`.
[4] Netcraft web server survey. Website. `http://news.netcraft.com/archives/category/web-server-survey/`.
[5] Ramsan. `http://www.ramsan.com`.
[6] Samsung. `http://www.samsung.com/`.
[7] We knew the web was big... Blog. `http://googleblog.blogspot.com/2008/07/we-knew-web-was-big.html`.
[8] D. G. Andersen, J. Franklin, M. Kaminsky, A. Phanishayee, L. Tan, and V. Vasudevan. Fawn: a fast array of wimpy nodes. In *SIGOPS'09*, Big Sky, Montana, USA, Oct. 2009.
[9] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *ReARCH'10*, Philadelphia, PA, USA, Dec. 2010.
[10] A. Badam, K. Park, V. S. Pai, and L. L. Peterson. Hashcache: cache storage for the next billion. In *NSDI'09*, Boston, Apr. 2009.
[11] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. In *SIGCOMM'06*, Pisa, Italy, Sept. 2006.
[12] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using bloom filters. In *SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.
[13] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software ip lookups with incremental updates. *CCR*, 34:97–122, Apr. 2004.
[14] K. Fall, G. Iannaccone, and S. Ratnasamy. Routing tables: Is smaller really much better? In *HOTNETS '09*, NY, USA, Oct. 2009.
[15] M. Gritter and D. Cheriton. An Architecture for Content Routing Support in the Internet. In *USITS'01*, SFO, CA, USA, Mar. 2001.
[16] V. Jacobson, D. K. Smetters, J. D. Thronton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Network Named Content. In *CoNEXT '09*, Rome, Italy, Dec. 2009.
[17] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In *SIGCOMM '07*, Kyoto, Japan, Aug. 2007.
[18] C. Lumezanu, K. Guo, N. Spring, and B. Bhattacharjee. The effect of packet loss on redundancy elimination in cellular wireless networks. In *IMC'10*, Melbourne, Australia, Nov. 2010.
[19] K. Pagiamtzis and A. Sheikholeslami. Content-addressable memory (CAM) circuits and architectures: A tutorial and survey. *IEEE Journal of Solid-State Circuits*, 41(3):712–727, March 2006.