Caesar: a Content Router for High Speed Forwarding

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

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

Jairo Esteban
Bell Labs, Holmdel, USA
jairo.esteban@alcatel-lucent.com

ABSTRACT

Today, high-end routers forward hundreds of millions of packets per second by means of longest prefix match on forwarding tables with less than a million IP prefixes. Information-Centric Networking, a novel form of networking where content is requested by its name, poses a new challenge in the design of high-end routers: process at least the same amount of packets, assuming a forwarding table that contains hundreds of millions of content prefixes. In this work we design and preliminarily evaluate *Caesar*, the first content router that supports name-based forwarding at high speed. Caesar efficiently uses available processing and memory units in a high-end router to support forwarding tables containing a billion content prefixes with unlimited characters.

Categories and Subject Descriptors

B.3.4 [**Arithmetic and Logic Structures**]: High-speed Arithmetic; C.2.1 [**Network Architecture and Designs**]: Network communications

General Terms

Design, Verification

Keywords

ICN, Forwarding, Router, Architecture

1. INTRODUCTION

Information-Centric Networking (ICN) [2] is a novel network paradigm where information or content are requested by their name instead of their location. This paradigm is realized by *name-based forwarding*, where a packet is forwarded based on "content name" instead of "content host". ICN enables tremendous evolution of routers that gain knowledge on what is transferred in a network.

Designs for name-based forwarding are classified as *flat* and *hierarchical*. Flat designs use a flat namespace and a resolution step, either explicit as in [4] or integrated with forwarding as in [15]. Hierarchical designs [13] use a hierarchical name space and forwarding based on Longest Prefix Match (LPM), as in IP. In this work, we target hierarchical designs, and for ease of notation we simply refer to them as ICN.

Today, speed is the main challenge in router design and *parallelization* is widely used to meet the speed requirements. In high-end routers, which rely on specialized hardware and software, parallelization is realized by allowing each line card to perform packet processing (e.g., packet forwarding and classification). Packet switching (transferring of a packet from an input to an output line card) is done by a centralized switch fabric [5]. In software routers, which are based on software running on general-purpose platforms, parallelization is realized by distributing packet processing on a cluster of thousands of servers [8, 11] (Section 2).

Parallelization is possible since each processing unit (line card or server) holds a complete forwarding table, *i.e.,* association of IP prefix with output interface (Section 3). This is feasible since today's forwarding tables are relatively small. IP prefixes have lengths from 8 to 24 (IPv4) and 16 to 48 (IPv6) bits; furthermore, IP is location dependent which guarantees efficient aggregation, e.g., 400,000 IPv4 prefixes cover about $4 \cdot 10^9$ IPv4 addresses. These conditions do not hold in ICN where content prefixes have unlimited characters and are location independent. It follows that ICN forwarding tables are expected to contain hundreds of millions of extremely long content prefixes [14, 3], making classic parallelization unrealizable.

The question we aim to answer in this paper is the following: *how do we design a router that supports name-based forwarding at high-speed?* We focus on a high-end router since software routers already struggle to compete in terms of speed with their specialized counterpart. However, as a future work we plan to extend our design to software routers as well, since novel research questions arise (Section 6).

This paper designs *Caesar* (Section 4), the first high-end router that sustains name-based forwarding at high-speed. Caesar has two main design forces. First, as software routers distribute processing in a cluster of servers to maximize speed, Caesar distributes the forwarding table across line cards to maximize size. The downside of this approach is a possible increase in switching operations that is absorbed by additional switch fabrics as commonly done in commercial routers [5]. Second, LPM is designed to be as much independent as possible from the length of content names: this is achieved by adapting the distributed Bloom filters approach proposed in [16] to content names.

We perform a preliminary evaluation of Caesar by means of a numerical evaluation (Section 5). Our analysis shows the following results. First, each Caesar line card can process up to 160 million packets per second, *i.e.,* 100Gbps assuming 80-byte packets. Second, coalescing 400 line cards, *i.e.,* one third of the line cards available in a state of the art router [5], Caesar supports high-speed

ICN'12, August 17, 2012, Helsinki, Finland.

forwarding on a billion content prefixes with up to 128 components and unlimited characters per component. Given these encouraging results, in the future we plan to implement Caesar on a FPGA and run an extensive evaluation (Section 7).

2. RELATED WORK

Software routers are novel router designs that consist of software running on general-purpose platforms. The main advantages of software routers are their extensibility and programmability. The challenge lies in their scalability when competing with high-end routers, under constraints imposed by general-purpose hardware, such as a limited number of physical ports. Caesar was largely inspired by novel advances in software router designs. For this reason, we now briefly overview RouteBricks [8] and PacketShader [11], today's most popular designs for software routers.

RouteBricks implements router functionalities using a decentralized architecture running on commodity hardware, and achieves linear capacity scalability by increasing the number of servers. The key to its performance lies in the careful exploitation of parallelization across and within servers. An important design decision is the use of Direct VLB [17], a load-balancing routing algorithm which parallelizes packet processing and switching. This algorithm is coupled with carefully tailored implementation strategies such as the use of parallelism in memory access, multiple receive and transmit NIC queues, and amortization of the cost of per-packet overhead by performing batch processing of packets. As a result, a parallel router consisting of 4 Nehalem servers, each one equipped with a 10Gbps NIC can sustain a load of 35Gbps, while introducing negligible delay, jitter, and packet reordering. The limiting factor in RouteBricks' performance is CPU, which is a typical bottleneck of PC applications, and a departure from previous software routers whose bottleneck resided in shared memory. Further, it is expected that the next generation of servers can provide a performance boost, not only by allowing an increase in the number of cores, but also by providing a larger number of PCIe slots.

PacketShader is another interesting approach to software routers. Similarly to RouteBricks, it makes abundant use of parallelization. The main departure from RouteBricks is the usage of Graphic Processing Unit (GPU) acceleration. PacketShader overcomes RouteBricks' performance limitations by exploiting the massively-parallel processing power of GPUs. The authors show that PacketShader can forward 64B IPv4 packets at 39Gbps on a single commodity PC equipped with four 10Gbps cards, essentially quadrupling the number of NICs that were supported by a single machine using RouteBricks. The new limiting factor in PacketShader's architecture has been shifted from the CPU to I/O, and more specifically, it is a consequence of an asymmetry of PCIe data transfer performance resulting from the use of dual-IOH motherboards.

Caesar differs from both RouteBricks and PacketShader in several aspects. First, Caesar is a design for high-end routers. Second, each Caesar line card does not have a full copy of the forwarding table. Third, Caesar supports name-based forwarding, which neither RouteBricks nor PacketShader support in their current stage. Furthermore, we are not aware of any design for a content router.

3. BACKGROUND

This section serves as a background for a clear understanding of the paper. First, we overview the most popular solutions for Longest Prefix Match (LPM); then, we summarize the design of today's high-end routers.

3.1 Longest Prefix Match

Routers use LPM to select the entry from the local forwarding table which shares the longest prefix with the destination address contained in a packet header. LPM is implemented either in hardware using a TCAM or in software using a Tree Bitmap [9] or Bloom filters [7, 16], to name a few. Solutions based on software are preferred because of TCAM's high cost and power consumption. The Tree Bitmap's performance degrades linearly as the tree depth increases; thus, it should not be used in presence of fast line cards, e.g., 100Gbps [16], and large forwarding tables. We now overview the Bloom filter approaches as they better suite the ICN requirements.

A Bloom filter is a data structure for membership queries that stores a "signature" of an item using k bits, independently of the original size of the item. The k bits are obtained as a result of k hash functions computed on the item, and stored in an array. Membership queries have no false negative probability and tunable false positive probability. In order to remove elements from a Bloom filter, a Counting Bloom Filter maintains a vector of counters corresponding to each bit in the bit-vector.

Dharmapurikar et al [7] are the first to use Bloom filters for LPM. Assuming packet addresses formed by B "components", e.g., $B = 32$ in IP, B Bloom filters are stored on on-chip memory (SRAM). Each Bloom filter i is populated with prefixes of length i. An off-chip hash-table (SRAM, RLDRAM or DRAM) stores for each prefix the output interface. Upon reception of a packet whose address has $B^{'}$ components, we query the $B^{'}$ Bloom filters associated to each possible prefix length obtaining a "matching vector". Then, we query the hash-table to verify eventual false positives, and retrieve the output interface.

Song et al. [16] observe that it would be more convenient to use a single Bloom filter (SBLF), and associate k different hash functions to each of the B possible prefix lengths. The SBLF is independent of both prefix length and distribution of prefix lenghts, but it cannot be accessed in parallel, detracting from the LPM speed. So, they suggest to split the SBLF into B smaller Bloom filters called "distributed Bloom filters" (DLB-BFs) and generate k groups of B hash functions, one from each of the B original group of k hash functions in the SBLF. Each DLB-BF should be implemented on at least B 2-port SRAM blocks in order to compute B hash functions in parallel. The DLB-BF has same false positive probability and optimal number of hash functions as the SBLF and the design in [7]. Finally the authors propose "ad hoc expansion", an optimization of the off-chip hash-table to bound worst case performance. If a particular LPM causes several false positives, a rule that matches the false positive is added to the hash-table; the rationale is to prevent future packets to incur several accesses to the off-chip hash-table to solve the false positive.

3.2 High-end Routers

The different components of a router can be logically divided in two categories: *data plane* and *control plane*. The data plane consists of N line cards with rate R, each containing several I/O interfaces, and one or more switch fabric with overall rate NR. When a packet arrives at a line card, the forwarding engine identifies the interface where the packet should be forwarded based on LPM computed over the Forwarding Information Base (FIB), a structure that stores pairs <prefix,output interface>. Additional processing such as policy routing and packet classification might be performed depending on the router features. Afterward, the packet is sent to a switch fabric that transfers the packet to the line card where the output interface resides.

The control plane is mostly composed by one or more route

controllers. Route controllers process route updates received from neighbor routers, and computed the best next hop for each destination. This information is stored in the Routing Information Base (RIB) and is used to compute the FIB that is then distributed to the different line cards.

The Cisco CRS-1 [5] is today's most advanced core router. It runs both on a single shelf and multi-shelves mode. In single-shelf mode, it has a single line card shelf with N=4,8,16 full-duplex line cards, each running at R=40Gbps. A single switch fabric supports a speed NR of 320, 640 Gbps, or 1.2 Tbps. In multi-shelves mode, the CRS-1 has up to 72 line card shelves, each with 16 full-duplex line cards at 40Gbps, for a total of 1,152 line cards. Overall, it switches packets across line cards at 92Tbps by means of 8 switch fabrics. The control plane contains two independent route processors (called "Cisco route controllers") in single-shelf mode, and a distributed route processor in multi-shelves mode.

4. Caesar

This section designs Caesar, the first high-end router that sustains name-based forwarding at high speed. We assume a NDN-like naming scheme. Content items are split in a sequence of packets identified by a content address. The content address is a hierarchical human-readable name formed by $B+1$ components delimited by a character: B components compose the *content name*, whereas the last component identifies a specific packet, for example `/ICN2012/PAPERS/PaperA.pdf/packet1`, where the delimiter is "/". Routers maintain forwarding information for *content prefixes* that are formed by any subset of components from the content names, for example `/ICN2012/PAPERS/*`.

4.1 Rationale

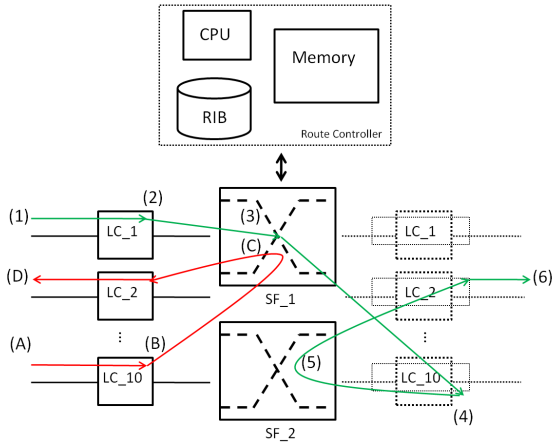
The design of a content router has two major challenges. First, a content router has to process at least the same amount of packets processed today by a high-end router, assuming forwarding tables that are several orders of magnitude larger. Second, its forwarding tables are filled with content prefixes that have a large number of components and unlimited characters per component.

Our rationale to support such large forwarding tables is to distribute them across the line cards. Thus, rather than duplicating the same forwarding table with S entries at each line card, each line card stores a (different) subset of entries $S^{'}$ with #$S^{'}$ = #S, such that, overall, the content router can serve n content prefixes, with $n = \sum_{i=1}^{i=N} S_i^{'} = NS^{'}$. Since the size of each forwarding table does not change, each line card still operates at a rate R. In order to distribute packet processing across line cards, we need a mechanism which is lightweight compared to LPM, while enforcing load balancing across line cards. Transferring packets across line cards causes additional switching operations. In the worst case, *i.e.,* when a packet is never processed at the line card where it is received, the switch fabric has to operate at a rate $2NR$.

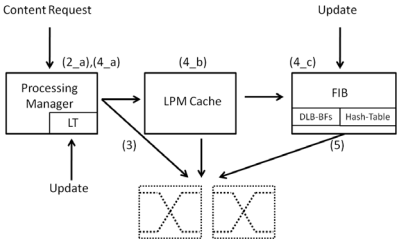
We now focus on the second challenge. Our goal is to build a LPM solution that scales well with the number of components in a content name and that is independent of the number of characters in each component. Based on these motivations, the DLB-BF is the perfect candidate [16] (cf. Section 3). In fact, it leverages a number of DLB-BFs k which is independent of the prefix length, and it maps each content prefix to k bits, independently of the number of characters used for each component.

4.2 Design

A Caesar router consists of N full-duplex line cards (LC_i) with rate R interconnected by at least two switch fabrics (SF_1 and SF_2)



(a) Control and data plane ; $N = 10$



(b) Line card.

Figure 1: Sketch of a Caesar router.

each supporting a rate NR, and a classic route controller (Figure 1(a) where $N = 10$). In the following, we detail the design of a Caesar router by focusing on both control and data plane. We conclude the section with an example.

4.2.1 Control Plane

The route controller (CPU, memory and RIB) is responsible for Caesar's control plane. We do not consider the requirements related to routing protocols execution as they are out of the scope of this paper. We only focus on the additional or modified operations a Caesar controller performs.

FIB Calculation and Distribution – A classic route controller distributes to each line card the same FIB derived from the central RIB. In Caesar, such FIB is then equally split across line cards by respecting the following condition: if a content prefix A is stored at a line card LC_i, all further packets whose content names have A as LPM should be processed by LC_i as well. We proceed as follows. We assign to each line card LC_i an integer i from 1 to N, and to each content prefix in the RIB a "contentID" derived by hashing[1] the first component of the content prefix. Then, each content prefix populates the FIB of the line card LC_i, such that $i = contentID \mod N$. Since content names are hierarchically organized and, to be unique, need to differ at least in the first component, the result of this operation is the same for a content prefix as well as for for all content names that can be originated by expanding this prefix. In addition, the hash operation enforces "load balancing": in presence of skew prefix popularity, the hash function uniformly distributes prefixes to each line card independently of their popularity value.

[1]We suggest to use the CRC-64 algorithm as hash function. CRC is simple to implement in binary hardware and it is sufficiently resistant to collision for our scope.

Thus, each line card sees the same popularity curve for each subset of prefixes. Finally, the route controller distributes the FIB to each line card along with the mapping $<LC_i$,interface$>$ for all i.

Failure Recovery – The failure of a line card jeopardizes reachability for the subset of prefixes it manages. Caesar enables redirection of traffic from a failing line card to a second line card by virtue of two design features. First, the pairs $<LC_i$,interface$>$ can be updated live so that traffic is forwarded to the new interface in presence of a failure. Second, the route controller has a copy of the RIB from which it can derive an up-to-date FIB for each line card. Therefore, when a failure occurs at line card LC_i, the route controller sends the FIB of LC_i to one of the L additional line cards pre-installed to handle individual line card failures. Then, it updates the pair $<LC_i$,interface$>$ with the interface selected from the new line card. Finally, it propagates this update to the N line cards. This mechanism still deserves a detailed understanding. For example, quantifying the off-service probabilities based on the value of L, as well as how to detect a line card failure. Such analysis is out of scope of this paper, but we intend to address this subject in the future.

4.2.2 Data Plane

We now focus on the data plane which covers line cards, and switch fabrics. We first describe the design of a line card (Figure 1(b)). This consists of four modules: Line card Table, Processing Manager, LPM Cache and FIB.

Line card Table (LT) – The LT is a simple array that contains at position i the interface to be used in order to reach line card i, for $i = 1 : N$. As advised in [10], we use 16 bits to index all the available interfaces in a router. Even assuming 2,048 line cards (CRS-1 has up to 1,152 line cards), 32Kbits of memory are enough. Thus, the LT table fits into a single 2-port SRAM block of the Xilinx Virtex 6 FPGA [1], which guarantees fast access time.

Processing Manager (PM) – The PM identifies the line card where a packet should be processed. First, it computes the contentID of the packet by hashing the first component of the content name. Then, it computes $i = contentID \mod N$, and reads from the LT at position i the interface of the line card where processing has to be done. These operations are negligible compared to an LPM operation, thus meeting our main design goal. If the retrieved interface resides on the local line card, the packet is processed locally. Otherwise, the PM sends the packet to the switch fabric that moves it to the line card where packet processing should be performed.

FIB – It receives as input a content name and outputs the interface where the corresponding packet should be forwarded. We implement the FIB using k on-chip DLB-BFs and an off-chip hash-table (cf. Section 3.1). As in [16, 7] we support prefix insertion and deletion through off-line mirror counting Bloom Filters. We call B_{max} the maximum length in terms of components of a content prefix; this is defined by the maximum number of hash functions $B_{max} \cdot k$ that can be originated in hardware and is discussed in the following section. The off-chip hash-table can be implemented in SRAM or RLDRAM using indexing schemes for high-speed caching [14]. The table is considered as a fixed size non-chained hash table. Every prefix is hashed with the CRC-64 algorithm; the first H-bits of the hash-value are stored along with next-hop interface information (16 bits) in the bucket corresponding to the hash value modulo the number of buckets[2]. In order to reduce the bucket overflow probability, as in [16] we hash every prefix two times and we pick the less loaded bucket. As RLDRAM and SRAM chips provide a bucket size of 144 bits, we pick H=56bits in order to store two entries in a single bucket. Ad hoc prefix expansion is used to bound performance in presence of false positives. Expanded prefixes are removed when they are no longer necessary or to limit the size of the hash table.

LPM Cache (LPM-C) – Packets with extremely long content names can cause high burden to the router due to the $B \cdot k$ hashes to be computed. Inspired by the ad-hoc expansion, we propose to cache in the LPM-C results from the LPM originated by content names with more than T components. At packet processing, we simply need to verify whether $B > T$ to decide to check or not the LPM-C. The first time a content name is seen at a Caesar router, a cache miss will occur causing an expensive LPM operation. This won't happen for upcoming packets of the same *flow*, where a flow is a sequence of packets that share the same content name. The LPM-C is implemented as a hash-table where we use as a key the content name (B components of the content address) and as value the output interface obtained as a result of the first LPM operation. Cache entries are managed with any simple replacement policy.

In the worst case, *i.e.,* when a packet is never processed at the line card where it is received, the switch fabric has to operate at a rate $2NR$. Similarly to the multi-shelves CRS-1 [5], we distribute the additional switching operations across a set of switch fabrics. As showed in [12], it is feasible to combine multiple switch fabrics together with no loss of performance at the price of small coordination buffers.

4.2.3 Example

Figure 1(a) shows the packet flow in case of a remote LPM operation (green line) and of a local one (red line). We focus on the remote LPM operation as it is more complete. Line card LC_1 receives a packet (1). L_1's PM computes the contentID and finds that line card LC_{10} is responsible for processing this packet as a result of the operation $i = contentID \mod N$ (2_a). The packet is then sent to one of the two switches, e.g., SF_1 in this example, that transfers it to LC_{10} (3) . The packet reaches LC_{10} (4), where the PM verifies that this line card is responsible for packet processing (4_a); then, if $B > T$ it checks the LPM cache to verify if a LPM result for this content name was previously cached (4_b). Assuming a cache miss happens or $B \leq T$, the FIB performs LPM giving as a result LC_2 (4_c). The packet is sent to one of the two switches, e.g., SF_2 in this example, that transfers it to LC_2 (5); finally the packet exits the router using line card LC_2 (6).

5. EVALUATION

We perform a numerical evaluation of Caesar. As a reference design, we consider the Xilinx Virtex-6 FPGA family; these devices have between 156 and 1,064 2-port SRAM blocks each storing up to 32Kbit of data, for a total of 34-Mbit on-chip memory, at most. We use the on-chip memory to store the DLB-BFs and the LT. We assume our reference board is further equipped with 216Mbits SRAM and 2.3Gbit RLDRAM off-chip memory, as the NetFPGA-10G [1], where we store the off-chip hash-table. Assuming the design of the off-chip hash-table described in Section 4, the off-chip SRAM and RLDRAM hold a maximum of 1.5 and 16 million content prefixes, respectively. Also, a maximum of 250 and 6.6 million accesses per second is tolerated.

We first focus on a single Caesar line card to quantify how fast it can process packets, and how many content prefixes it can handle in its forwarding table. Figure 2(a) plots the number of content prefixes that can be stored in the DLB-BFs, S', as a function of the average packet size P, considering line card rates λ=10,40,100Gbps. A line card performs an average number of LPM operations equal

[2]Buckets are the addressable elements of an hash table.

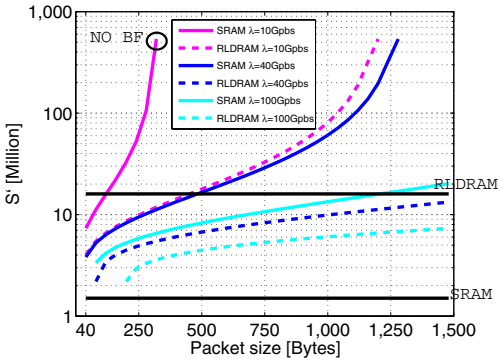
to $\frac{\lambda}{P}$, e.g., when λ=100Gbps and P=40Bytes about 320 million LPM operations per second are required. The curves are obtained assuming an off-chip hash-table implemented both in SRAM and RLDRAM with infinite size; the two horizontal lines represent the maximum number of content prefixes that can be stored in the reference design, *i.e.,* SRAM (1.5M) and RLDRAM (16M). Finally, we set the threshold for caching LPM results, T, to 64, and the maximum number of components in a content name, B_{max}, to 128. We discuss these parameters later in the section.

We first focus on the curves obtained assuming an SRAM-based hash-table (solid lines). The Figure shows that for λ=10,40Gbps a Caesar line card supports even 40Byte packets, whereas when λ=100Gbps packets need to be at least 80Bytes long. For each λ, as P increases $S^{'}$ increases as well; in fact, as P increases the rate of LPM operations decreases, which means that more accesses to the off-chip hash-table are tolerated. It follows that the DLB-BFs can be dimensioned to support a higher false positive probability, which allows to increase $S^{'}$. In the extreme case, the false positive probability reaches a value of 1 which means that the DLB-BFs can be removed, e.g., for P=160Bytes when λ=10Gbps ("No BF" label in the Figure). At this point, the line card can theoretically handle infinite content prefixes since we assume no size limit for the off-chip hash-table; for this reason, the curve is interrupted. Assuming our reference design, the DLB-BFs should not store more than 1.5 million content prefixes due to the size limit of the SRAM-based off-chip hash-table. This threshold is always smaller than the derived values of $S^{'}$, independently of P and λ. Thus, the SRAM memory in the off-chip hash-table is the bottleneck for the size of the line card's forwarding table.

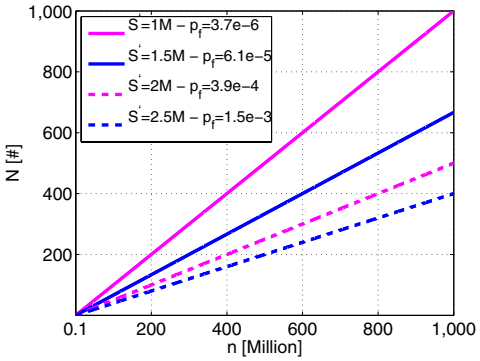
When RLDRAM technology is used for the off-chip hash-table (dashed lines), $S^{'}$ assumes values lower than the 16 million limit imposed by the RLDRAM for most values of P and λ. However, due to the longer access time compared to the SRAM (15 vs 4ns), only at 10Gbps we can support 40Byte packets, whereas an average packet size of 80 and 200Bytes is needed to support 40 and 100Gbps, respectively. Thus, the RLDRAM memory in the off-chip hash-table is the bottleneck for the line card's speed.

We now focus on four configurations derived from the left-hand side area of Figure 2(a): we set $S^{'}$ to 1-1.5 million with off-chip SRAM, and 2-2.5 million with off-chip RLDRAM. These configurations allow to handle both high rates and small packet sizes, while respecting the maximum amount of prefixes that each off-chip memory can store. An additional design goal is to minimize the false positive probability p_f of the DLB-BFs, in order to reduce the ad-hoc prefix expansion (cf. Section 4). For each configuration, we can derive the number of hash-functions k that minimizes p_f. We choose $k = 8$ for three reasons: i) it minimizes the highest false positive probability across the four configurations, while increasing the lowest one from 10^{-8} to 10^{-6}; ii) a power of two value is easier to implement in hardware; iii) a small value of k is beneficial to reduce the overall number of hash-functions. With k=8, the false positive probability for the four configurations varies between 10^{-6} and 10^{-3}; this means that one flow every million and thousand new flows, respectively, generates a new entry for the hash-table due to the ad-hoc prefix expansion.

We divide the 1,064 2-port SRAM blocks across 8 DLB-BFs, such that each DLB-BF is composed by 64 SRAM blocks of 64Kbit, each composed by two 32Kbit 2-port SRAM blocks. Then, we use one of the remaining 2-port SRAM block to store the LT. We generate 128x8=1,024 hash functions; as discussed in [16], this can be efficiently implemented in hardware. With this configuration, each DLB-BF can compute 64 hash functions in one clock cycle, and



(a) Number of prefixes stored on a Caesar line card $S^{'}$ as a function of the packet size P; $\lambda = [10; 40; 100]$Gbps.



(b) Number of line cards N as a function of the number of prefixes n ; $S^{'} = [1; 1.5; 2; 2.5]$million ; $p_f = [2.0e^{-6}; 4.4e^{-5}; 3.4e^{-4}; 1.5e^{-3}]$.

Figure 2: Caesar preliminary evaluation.

support content names and prefixes with up to B_{max}=128 components. The LPM for content names with more than 64 components might require up to 2 clock cycles, thus impacting worst case performance. However, since T=64, only the first packet of a flow reaches the FIB, whereas the remaining packets are directly processed in the LPM-C. In [14], we measured that out of 10,000 unique users and 15,000 unique URLs the longest URL has 70 components. This suggests that T=64 is enough to guarantee a small LPM-C. Note that we could increase T to 128 by assigning 128 2-port SRAM blocks to each DLB-BF. However, we prefer to over dimension the DLB-BFs due to some subtle load balancing issues across 2-port SRAM blocks, as discussed in [16].

We now evaluate a Caesar router as a whole. Figure 2(b) plots the number of line cards N as a function of the prefix size n for each of the above configuration. The Figure shows that with 400-1,000 line cards (tunable false positive) we support up to a billion content prefixes. This value of n is five times larger than the number of active hostnames in today's Internet [14]. A router with 1,000 line cards is feasible, e.g., the Cisco CRS-1 has up to 1,152 line cards. However, a Caesar router requires a double switching capacity compared to a classic router; to avoid a bottleneck in the switch fabrics, we should not consider more than ∼600 line cards. Nevertheless, a Caesar router with 600 line cards handles up to 600 million content prefixes with $p_f = 10^{-6}$, or even a billion content prefixes tolerating $p_f = 10^{-4}$.

6. DISCUSSION

In this paper, we focus on the design of a high-end content router rather than software, since software routers barely compete in terms of performance with their specialized counterpart. However, software routers provide the attractive feature that they can be programmed, configured and extended by using general-purpose platforms, commodity programming languages and tools (cf. Section 2). We believe that Caesar is generic enough to be extended to a software router as well.

There are few research issues that need to be solved before applying Caesar to a software router. In the following, we shortly summarize the main challenges and propose a modification of the Caesar design to meet each requirement. In the following, we refer to software implementation of Caesar as S-Caesar.

Limited per-node processing rate – Per-node processing in a distributed software router is constrained by general-purpose CPU and memory. PacketShader has proposed the use of GPUs for software routers. GPUs provide the added advantage of hundreds of cores and fast memory access, which significantly increases the operational speed of a router. In addition, graphics processing requires extreme parallelism and GPUs are designed with parallel processing as a primary guideline.

Similar to PacketShader, we plan on using GPUs for S-Caesar, in order to avoid packet processing bottlenecks. Specifically, we will implement the distributed bloom filters using GPUs: we will store DLB-BFs on the shared memory that exists on each streaming multiprocessor of the GPU, and the off-chip hash-table on device memory of the GPU. Furthermore, we will store control plane announcements of the content prefixes in the general purpose host memory of the node.

Limited internal link rates – Given the sheer size of FIB in a content router, it is unlikely that a single GPU-enabled node will suffice. As a result, we will use multiple nodes, each performing lookup on a portion of FIB. However, similar to RouteBricks, connecting these nodes creates another bottleneck: the interconnection links. Specifically, the use of commodity NICs and link technologies may significantly affect the performance of S-Caesar, as mentioned in RouteBricks.

We plan on exploring this problem by investigating various interconnection topologies, e.g., butterfly, torus, and full mesh [6]. We will start with a full mesh topology of GPU-enabled nodes and examine whether it can fulfill the needs of a content router. We prefer a full mesh because, in the worst case, it only adds one hop to the lookup. However, if a full mesh proves insufficient, then we will experiment with multi-hop topologies.

7. CONCLUSION AND FUTURE WORK

Future Internet architectures are expected to be centered around content and not machines. This paradigm shift causes a high burden on routers, due to the explosion of the forwarding tables and the usage of extremely long content names. Motivated by these observations we have designed Caesar, a high-end router that supports name-based forwarding at high speed. Caesar is built on two founding principles: i) the forwarding table is distributed across line cards to maximize size, and ii) Longest Prefix Match is designed to be as much independent as possible from the content prefix length. Our preliminary evaluation shows that Caesar supports high-speed forwarding on a billion content prefixes with up to 128 components, each with unbounded number of characters.

Currently, we are implementing Caesar on a network processor to verify the numerical results discussed in the paper. An exciting avenue for future work is the extension of Caesar to software routers. There are few research challenges that need to be solved before applying Caesar to a software router. For example, the number of physical connections among servers should be small since commodity servers have few NIC slots. In Caesar, each line card eventually communicates with all other line cards to support distributed packet processing. We thus need to rethink the latter operation so to map it to a realistic overlay constructed among servers.

Acknowledgments

This work has been partially funded by the French national research agency (ANR), CONNECT project, under grant number ANR-10-VERS-001. We also would like to thank Ghulam Memon (University of Oregon) for his insightful comments and suggestions.

8. REFERENCES

[1] Netfpga. `http://netfpga.org/`.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A Survey of Information-Centric Networking. In *Dagstuhl Seminar on Information-Centric Networking*, 2011.

[3] Ashok Narayanan and David Oran. NDN and IP Routing Can It Scale? `http://trac.tools.ietf.org/`.

[4] 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.

[5] Cisco CRS-1. `http://www.cisco.com`.

[6] W. Dally and B. Towles. *Principles and practices of interconnection networks*. Morgan Kaufmann, 2004.

[7] S. Dharmapurikar, P. Krishnamurthy, and D. E. Taylor. Longest prefix matching using bloom filters. In *SIGCOMM '03*, Karlsruhe, Germany, Aug. 2003.

[8] M. Dobrescu, N. Egi, K. Argyraki, B.-G. Chun, K. Fall, G. Iannaccone, A. Knies, M. Manesh, and S. Ratnasamy. Routebricks: exploiting parallelism to scale software routers. In *SOSP '09*, Big Sky, Montana, USA, 2009.

[9] W. Eatherton, G. Varghese, and Z. Dittia. Tree bitmap: hardware/software ip lookups with incremental updates. *Computer Communication Review*, 34(2):97–122, Apr. 2004.

[10] K. Fall, G. Iannaccone, and S. Ratnasamy. Routing tables: Is smaller really much better? In *HOTNETS '09*, NY, USA, Oct. 2009.

[11] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: a gpu-accelerated software router. *CCR*, 40(4):195–206, Aug. 2010.

[12] S. Iyer and N. W. McKeown. Analysis of the parallel packet switch architecture. *Transaction of Networking*, 11(2):314–324, Apr. 2003.

[13] 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.

[14] D. Perino and M. Varvello. A reality check for content centric networking. In *ICN'11*, Toronto, Canada, Aug. 2011.

[15] A. Singla, P. B. Godfrey, K. Fall, G. Iannaccone, and S. Ratnasamy. Scalable routing on flat names. In *Co-NEXT'10*, PA, USA, 2010.

[16] H. Song, F. Hao, M. S. Kodialam, and T. V. Lakshman. Ipv6 lookups using distributed and load balanced bloom filters for 100gbps core router line cards. In *INFOCOM'09*, Rio de Janeiro, Brazil, 2009.

[17] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC'81*, Milwaukee, WI, USA, 1981.