Augustus: a CCN router for programmable networks

Davide Kirchner[1*], Raihana Ferdous[2*], Renato Lo Cigno[3],
Leonardo Maccari[3], Massimo Gallo[4], Diego Perino[5*], and Lorenzo Saino[6*]
[1]Google Inc., Dublin, Ireland; [2]Create-Net, Trento, Italy; [3]DISI – University of Trento, Italy
[4]Bell Labs – Nokia, Paris, France; [5]Telefonica Research, Spain; [6]Fastly, London, UK
davide.kirchner@yahoo.it; rferdous@create-net.org; renato.locigno@unitn.it;
leonardo.maccari@unitn.it; massimo.gallo@nokia-bell-labs.com;
diego.perino@gmail.com; lsaino@fastly.com

ABSTRACT

Despite the considerable attention that the ICN paradigm received so far, its deployment has been hindered by the scale of upgrades required to the existing infrastructure. Software programmable networking frameworks would constitute a remarkable opportunity for ICN as they enable fast deployment of novel technologies on commodity hardware. However, a software ICN router implementation for commodity platforms guaranteeing adequate packet processing performance is not available yet. This paper introduces *Augustus*, a software architecture for ICN routers, and detail two implementations, stand-alone and modular, released as open-source code. We deployed both implementations on a state-of-the-art hardware platform and analyzed their performance under different configurations. Our analysis shows that with both implementations it is possible to achieve a throughput of approximately 10 Mpps, saturating 10 Gbit/s links with packet as small as 100 bytes. However, to achieve such performance, routers must be carefully configured to fully exploit the capabilities of the hardware platforms they run on.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Network communications, Packet-switching networks

Keywords

Information centric router; Commodity hardware; Modular ICN router; System Design; Prototype; Performance evaluation; Experimental evaluation

*This work was done while D. Kirchner and R. Ferdous were at the University of Trento, and D. Perino and L. Saino at Bell Labs.

ICN'16, September 26 - 28, 2016, Kyoto, Japan

ISBN 978-1-4503-4467-8/16/09…$15.00

DOI: http://dx.doi.org/10.1145/2984356.2984363

1. INTRODUCTION

Information-centric networking (ICN) is a novel networking paradigm putting content at the core of the communication model. The location-based (or host-based) communication model of the current Internet infrastructure is replaced by a content-based one in which unique routable names are used to identify and locate content items. Several ICN approaches have been proposed, including NetInf [4], Data-Oriented Network Architecture (DONA) [11] and NDN [6] to cite a few. Despite the benefits such architectures provide, their deployment is held back by the significant changes required to existing network equipment.

Network Function Virtualization (NFV), on the other hand, enables fast deployment of novel services at a small cost on commodity hardware. NFV represents an opportunity to ease the introduction of ICN on the existing infrastructure, but it requires designing high-performance ICN software systems running on commodity hardware platforms.

In this work, we focus on the Content-Centric Networking (CCN) approach, initially proposed in [6], for which two distinct protocol definitions are available, Content Centric Networking (CCN), http://ccnx.org and Named Data Networking (NDN), http://named-data.net. Open source software prototypes are available and have been widely used for research on fundamental NDN protocols and algorithms, but such prototypes are not intended for production environments as they do not provide adequate performance.

Only few research efforts investigated the design and implementation of high-performance ICN software systems [13, 18, 19, 20]. These studies rely either on proprietary software and/or hardware, or focus on high-speed caching only. This paper presents the design and performance evaluation of *Augustus*, a high-performance, opens-source ICN router, that uses the fast packet processing libraries DPDK (Intel® Data Plane Development[1]) to leverage the potential of commercial general purpose hardware.

Augustus follows the basic design and forwarding principles of the CCN protocol, which we implemented in two different flavors: a standalone monolithic design based on DPDK to manage packet I/O, and a modular design prototype based on the Click framework. A monolithic design gives full control of the platform, and allows pushing the architecture to its throughput limit; a modular approach

[1]http://dpdk.org

simplifies the extensibility, composition, reuse, and replacement of the router's features. The modular implementation builds upon the well-established Clock software router framework with its several variants, from optimized versions like FastClick [2] to the integration with different drivers and virtualization through projects like ClickOS [15], and to the hardware offloading [9, 21].

Summarizing the main contributions of the paper:

- We present the design of a high speed CCN router running on commodity hardware;
- We explore both monolithic and modular software router designs highlighting advantages and disadvantages of the two proposals;
- We release the developed code as two separate open-source projects: the stand-alone version of *Augustus* https://github.com/nokia/Augustus and the FastClick version https://ans.disi.unitn.it/software-projects;
- We conduct a performance evaluation for comparing monolithic and modular designs;
- We provide general guidelines for software CCN router design, and insights on the most delicate steps of their implementation.

2. RELATED WORK

A considerable amount of research focused on the design and performance evaluation of the fundamental mechanisms of the CCN protocol. Such body of literature is out of the scope of this paper, so we limit our review to the feasibility and high-speed performance of CCN. Most of the published works focus on optimizing single components of the forwarding procedure, such as the Forwarding Information Base (FIB), Pending Interest Table (PIT) or the Content Store (CS).

Among the design proposals for the FIB and the related longest-prefix queries, [20] proposes to augment a hash table by storing the longest prefix associated to each sub-prefix, so that worst-case lookup time only depends on the FIB itself and not on the number of components of the interest's name. The authors of [18] propose and implemented a so-called Prefix Bloom Filter, a data structure that relies on multiple Bloom filters to estimate the length of the prefix before performing an exact match on a standard hash table. Several software hashing techniques are compared in [16]; the same paper proposes a hardware-assisted hash table design for CCNx forwarding on variable length names.

One of the first results on PIT scalability is represented by [22]. Authors explore candidate PIT designs, evaluate them numerically, and implement a DHT-based PIT (the most promising scheme according to their study) on a network processor. The authors of [23] propose to use per-interface Bloom filters in order to reduce the memory footprint of the PIT and central Bloom filter for limiting the false positives. In [3], the authors estimate the size and the number of accesses of the PIT by using an approximate translation of IP traffic. Moreover, they also propose a Name Component Encoding solution to limit PIT size and accelerate lookup operations. Finally, in [24] the authors propose to store fixed-length fingerprints instead of name strings relaxing some of the original CCN architecture principles (*i.e.*, interest aggregation) in core routers.

The first line-speed CS was proposed in [1] with the estimation of the amount of resources necessary for a router. The authors of [13] present a high-speed hierarchical CS design implemented in user space leveraging DPDK, and explore its performance trade-offs.

In contrast with those studies, which investigate only specific components, this paper focuses on a complete high-performance CCN software router. Few studies exist on this topic and the mainstream reference implementations (CCNx and NDN Forwarding Deamon) have so far focused on functionality and flexibility rather than performance. The feasibility of CCN forwarding at line speed based on the available hardware and software technologies is explored in [17]. Authors of [20] discuss the implementation of a closed-source high-performance CCN forwarder capable of reaching 8 million packets per second (Mpps) on a 12-cores, Linux-powered Integrated Service Module (ISM) installed in a Cisco router. Authors of *Caesar* [18] propose a (closed-source) forwarder running on an NPU-based enterprise router architecture yielding up to 13 Mpps.

To the best of our knowledge, the present paper is the first to propose an open-source software implementation of a CCN router for programmable networks providing high-speed packet forwarding on commodity hardware.

3. SYSTEM ARCHITECTURE

This section describes the architecture and implementation of the *Augustus* content router. Its design leverages the experience gained with *Caesar* [18], a content router for Network Processors, but is conceived to run on standard, off-the-shelf, general purpose hardware without requiring any additional component.

3.1 Design

The *Augustus* content router is a multi-threaded user space application running on a general-purpose x86 server equipped with commodity NICs. This design decision allows for greater flexibility and cost-effectiveness, as commodity servers can provide great processing power at a considerably lower cost compared to specialized hardware designed for packet processing, like network processors. Accordingly, they are more suitable for experimenting with named-based forwarding, without requiring investment on dedicated hardware.

To achieve high packet processing performance (without using specialized hardware), we design our system to take advantage of the specific capabilities offered by x86 architectures through a number of optimization techniques. First, with appropriate multiple core design we exploit the massive parallelism offered by modern hardware: multiple hardware queues of NICs, multiple CPU cores, multiple DRAM memory channels and multiple NUMA sockets. Second, we implement frequently executed operations using the advanced capabilities offered by the x86 instruction set. As an example, we accelerate the execution of 32-bit CRC hashing by using instructions provided by the Intel Streaming SIMD Extensions (SSE). Third, we carefully place data in memory in order to maximize the probability that CPU cores access data located in the local NUMA node, hence reducing the latency caused by remote memory access.

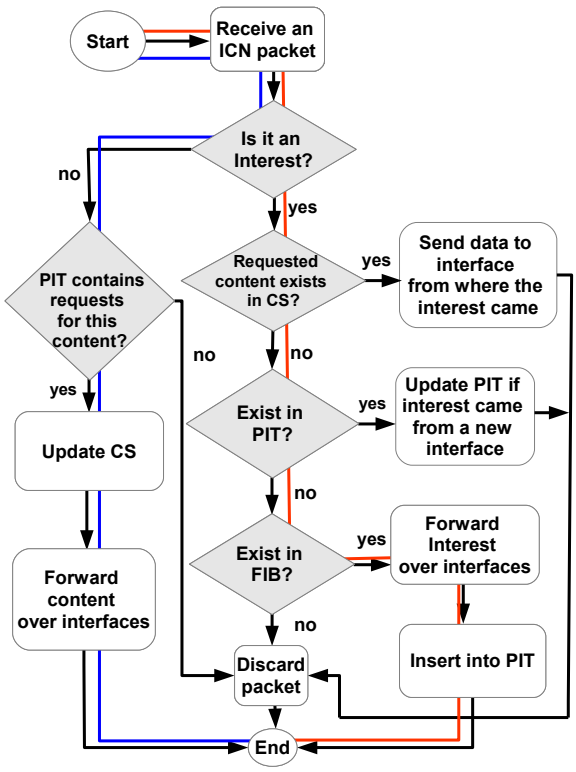

Figure 1: Forwarding principle of *Augustus*; the red and blue lines correspond to the worst case performance evaluation paths for interest and data packets respectively.

3.1.1 Data structures and packet processing

The CCN data plane comprises three main data structures: FIB (Forwarding Information Base), PIT (Pending Interest table), and CS (Content Store). In *Augustus* data structures and lookup algorithms are similar to those used in *Caesar*, (open address hash tables to perform exact match on PIT and CS and name-based longest prefix match on FIB). Content items are retrieved using two packet types: *Interest* packets, which are sent by a client requesting a resource with a given name, and *Data* packets, which carry the resource itself.

Augustus implements the packet format defined in [5] that uses component's offset (i.e., hierarchy is defined by offsets), avoiding Type Length Value fields for name components that would force the router to parse the entire name. Moreover, the implemented protocol requires that both interest and data packets hold the full name of the content, similarly to the CCNx 1.0 protocol. This design decision forces each data packet to be explicitly requested by a corresponding interest packet and allows exact match on PIT and CS, avoiding a more computationally expensive longest prefix match and attribute-based filtering that will significantly increase interest and data forwarding overhead.

The packet processing logic is shown in figure 1. When an interest packet is received, its name is first looked up (exact match) in the CS. In case of a match, the router serves the requested data from its cache, otherwise its name is looked up in the PIT. If a match is found, the interface from which the Interest was received is added to the existing PIT entry. Otherwise, the router searches for the longest prefix match in the FIB for the next-hop information about where the content can be found. The interest is then temporarily stored in the PIT and forwarded. Since the FIB lookup mechanism is not the main focus of the present work, we implement a direct hash-based search without further optimization.

When a data packet is received, its name is first looked up in the PIT: in case of a match, the data is stored in the CS, forwarded to all neighbors that had requested it, and the associated PIT entry is dropped.

Since DRAM has a higher access latency than more expensive memories used on high-end hardware routers, like TCAM and SRAM, we implement all data structures with the objective of minimizing the number of memory accesses, and to better exploit the transparent cache layers. This is achieved by carefully aligning data structures to CPU cache lines, compacting as much as possible all information that is needed at once.

The management of processing cores is aided by the DPDK abstraction library, which supports multi-threading, but requires to pin at initialization time each thread to a specific CPU core to prevent context switches and better exploit cache locality.

3.1.2 Concurrent memory access

To process packets at high speed, it is necessary to minimize the contention caused by concurrent access to shared data structures with high write frequency, specifically PIT and CS. To achieve this objective, we *shard* PIT and CS across all threads. As a result, each thread has exclusive access to a portion of such data structures, and can therefore execute read/write operation without synchronizing with other threads. Differently, read-only data structures, like the FIB, are concurrently accessed by multiple threads but they are nevertheless replicated on each NUMA socket, so that each core can access a copy from its local socket.

To enable correct sharding operation, it is necessary that Interest and Data packets for a specific chunk are always processed by the same thread, otherwise this would lead to PIT and CS misses and pollution. This can be achieved using two alternative methods. The first method consists in using the Receive Side Scaling (RSS) functionality offered by modern NICs. RSS allows to allocate received packets to specific hardware queues based on the hash of specific fields. By hashing packets based on the content name and assigning each hardware queue to a specific thread, all Interest and Data packets referring to a specific content chunk will be always processed by the same thread.

This approach is very simple and allows good processing performance as it does not require any synchronization among threads. However it requires NICs to support hashing arbitrary variable-length fields (*i.e.*, the content name), which may not be supported by lower-end NICs, which normally support only RSS based on the 5-tuples (source IP, destination IP, transport layer protocol, source port, destination port). This limitation can be addressed, for example, reserving a number of cores to operate as dispatchers, that read packets from NICs, compute the hash on the name and pass it to the responsible processing thread via lockless ring queues. However, this solution reduces the number of cores that can be used to perform packet forwarding. The solution we adopt is to encapsulate ICN packets into a UDP

datagram and using a different destination port depending on the hash of the content or assigning to each router given IP addresses and use a different address depending on the content hash. This is a common technique adopted also in [12].

3.2 Implementation

Augustus is implemented following two different strategies providing different trade-offs between performance and convenience of increased flexibility:

- A standalone monolithic implementation, written in C and based directly on the DPDK packet processing API for fast packet-based I/O in user space and for its optimized low-level utilities. This approach provides direct control over all the operations involved in the data plane, so as to be able to strip them down to the bare minimum.

- A modular implementation, which runs in the Click modular router framework [10]. This approach causes higher overhead, but it allows a CCN forwarder to be deployed as a component of a more comprehensive router. Although this approach trades inefficiency with increased flexibility, it can still make use the DPDK library for efficiency, and it relies on the open-source project FastClick [2], a performance-oriented version of Click.

Augustus's monolithic implementation is directly derived from *Caesar*, and we refer the readers to the original paper [18] for details. In the following we introduce the modular architecture of Click and a more detailed description of the modular implementation.

3.2.1 Click Modular Router

A Click router is defined by a configuration file that describes the router in terms of a directed graph connecting basic packet processing modules, called elements. Each element provides a simple packet processing function (e.g., packet classification, queuing), and each edge in the configuration graph connects two elements' ports. Ports can be of two different types: *push* or *pull*. In push connections (two push ports connected in the graph) packets are transferred downstream from source to destination element. In contrast, in *pull* connections (two pull ports connected in the graph), packets are pulled from the destination element upstream and transferred when available. Figure 2 shows an example of Click element's interface with four push ports (one input and three outputs): *CheckICNHeader* works as a filter to check the validity of an ICN packet. It takes packets from its single input port and has three output ports: valid Interest packets are pushed out on port 0, Data packets on output port 1 while any invalid/non-CCN packet is pushed out on port 2 or dropped.

In Click, a packet object consists of a *playload*, the actual packet (including headers), and *annotations*, metadata used to carry information downstream.

3.2.2 Modular Content Router

For the modular *Augustus* prototype built in Click, our focus is to design a set of small, discrete, scalable, and reusable elements/modules that connected together perform packet

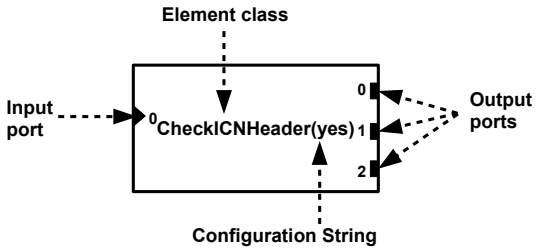

Figure 2: A sample click element

processing, thus exploring the full advantages of a modular architecture: flexibility, extensibility, composition, and reuse.

However, Click's reliance on packet flow as an organizational principle means that use of a set of smallest elements for designing a system is not always appropriate. In fact, large elements are required when control or data flow does not follow the flow of packets. Moreover, in some cases the control flow required to process a protocol is too complex to be split into smallest elements.

In this context, consideration of the data structures and packet processing operations of *Augustus* leads to the design of the CCN forwarding in three core elements: i) *ICN_CS*, ii) *ICN_PIT*, and iii) *ICN_FIB*, corresponding to the three packet processing data structures. These elements carry their own data structures, some of them reused across different modules, and all logic related to accessing CS, PIT and FIB as discussed in Sect. 3.1.1. The composition of the data-structure and packet processing operations in three main modules provides a strong control in operation and flexibility in modification without affecting the rest of the system in contrast to the tightly-coupled monolithic implementation. To simplify the configuration of the routing process, a few other smaller elements are implemented:

- Elements *InputDeMux* and *OutputMux* are introduced to simplify the input and output flow. Particularly, the forwarding procedure is based on the notion of interfaces with point-to-point links, and throughout the procedure queries on all three data structures may result in a packet being emitted for transmission from all the three core elements. These modules help managing input and output links;

- Element *CheckICNHeader* is introduced as an initial filter to validate the formatting of CCN packets to avoid the overhead of processing non-CCN packets.

Figure 3 shows the packet processing flows between the elements of the modular *Augustus* prototype. The modules are carefully designed to avoid additional packet copies in the forwarding of the packet. For instance, after a data packet hits the PIT, it is sent over two ports: port 1 for storing the packet in the CS and port 2 to forward the packet, without additional packet copies.

Packet processing starts with the *InputDeMux* element. This module works as a demultiplexer taking an arbitrary number of inputs and emitting all packets on its single output. Packet annotations are used by this module to store important information about the packet (the input interface from where the packet was received, the hash value of the

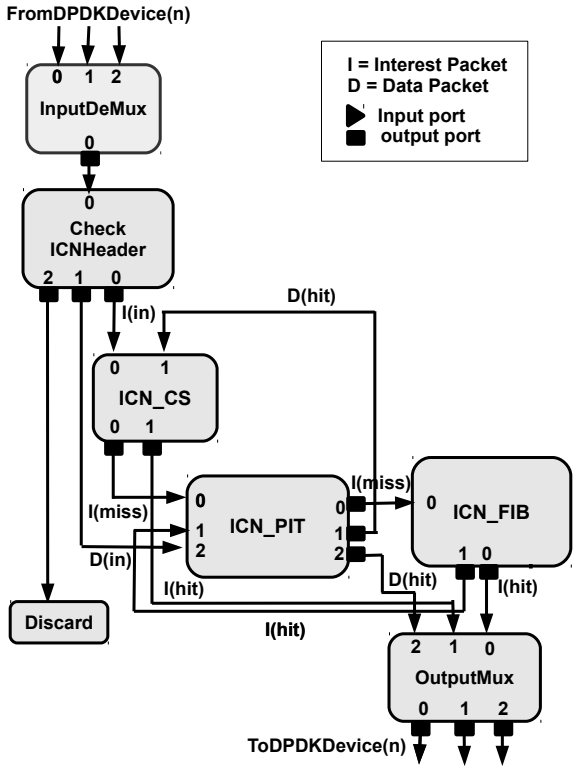

Figure 3: Packet processing flow of FastClick modular implementation.

name, etc.) to avoid that other core elements recalculate them.

After the initial verification of the CCN packet validity, the *CheckICNHeader* element forwards the packet to the main core elements (*ICN_CS*, *ICN_PIT*, and *ICN_FIB*) according to the forwarding logic shown in figure 1. The *OutputMux* element is responsible for forwarding processed Interest and Data packets to the output interfaces. This element takes care of reading the packet annotation, retrieving the information about the output interfaces and sending the packet to its corresponding output port (or ports).

4. PERFORMANCE EVALUATION

The main focus of our evaluation is to show that our implementation can operate at wire speed and that it scales well with respect to FIB size. For this reason we do not need a detailed traffic model, instead, we generate small packets with random content so that the transmission speed of the links is not a bottleneck, and lookup functions are fully stressed as all interests are new and do not hit local caches.

Intuitively, one may argue that increasing the number of threads would make it possible to exploit better the availability of multiple cores and hence increase performance. We show that in realistic operational conditions this is not always the case, as modern architectures share data caches that represent the key bottleneck for forwarding functions in

CPUs	2 ×	Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz
Memory	4 ×	16 GB @ 1866 MHz
NICs	1 ×	Intel 82599ES (two 10GbE ports)
	1 ×	Intel I350 (two 1GbE ports)

Table 1: Hardware configuration for the two test servers.

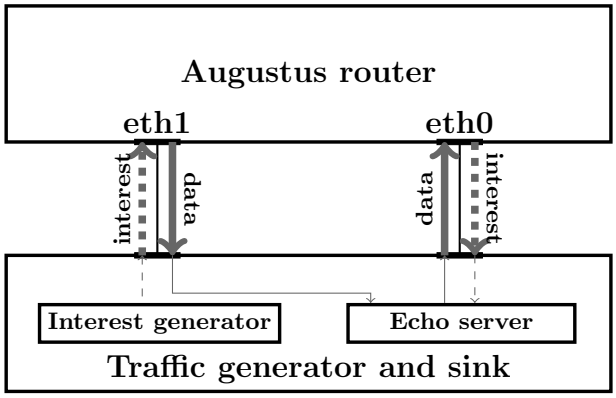

Figure 4: Traffic flow for the router evaluation experiments.

CCN software-based implementations. In principle this is a known issue, we quantify its impact and we provide important insights on the number of threads that is convenient to use to maximize throughput.

We test both *Augustus* implementations. The FastClick implementation trades the modularity of its approach with an increased internal complexity and some loss of performance. Our evaluation quantify the loss, and gives an initial understanding of the reasons behind it.

4.1 Experimental setup

The testbed is composed of two identical general-purpose servers, each equipped with two 10 Gbit/s Ethernet network interfaces. The details of the hardware architecture are reported in table 1. One server runs the software router, and the other runs a custom traffic generator and a data sink as depicted in figure 4.

Before any performance evaluation of the router took place, the traffic generator and data sink were tested with a physical loop between its interfaces. The machine is able to generate and measure a throughput up to 10 Gbit/s with data packets containing 87 bytes of Ethernet payload, thus it is never a bottleneck. As forwarding limits are measured in packets per second, we present results obtained with data packets that are just as large as the minimum interest packets (58 bytes of Ethernet payload, including the IP header, smaller data-packets would not make sense). It is important to remark that with any "reasonable" mix of interest and data payload size the router architecture we present very easily support 10 Gbit/s links with off-the-shelf hardware.

For the throughput experiments, the servers were connected as shown in figure 4, with interest packets forwarded in one direction through the router and data packets in the opposite one. The traffic generator sends interest packets attempting to overload the router and replies to incoming

interests with random data packets matching the interests' content name. It also acts as a data sink, counting and then discarding incoming data packets. Throughput measurements are expressed in packets per second (pps) and refer to the number of data packets received at the traffic sink. Given the present setup, the results imply that the router was able to handle also an equivalent rate of interest packets, effectively sustaining a bi-directional throughput that is twice as large as what we report.

4.2 Parallelism level and threads placement

In order to explore the maximum performance achievable with the described setup, we ran both router implementations with an increasing number of parallel threads. As the workload can be split among virtually independent threads, one would expect performance to scale with the number of threads enabled on our 16-core (32 with hyperthreading) architecture. A key element to consider, however, is that the number of available processing cores alone is only one of the parameters that influence the performance boost. In the servers we used for testing (like in any modern general-purpose processor) threads can be distributed on the physical processing cores with different approaches that result in substantially different performance outcomes. Figure 5 describes the layout of the CPU cores that equip our servers. Two independent NUMA sockets are present in our machines, each one with a dedicated layer-3 (L3) cache. Each socket has 8 independent cores, with a dedicated L2 cache. Each core supports the execution of two threads with hyperthreading, each one with a dedicated L1 cache. We expect that placing independent threads on cores that do not share cache maximizes their performance, as forwarding is a memory intensive task[2]. In general, we expect that overall performance will be maximized with a configuration that allows exploiting the maximum possible processing power while minimizing cache interference. However, stating what this configuration is, especially with the more complex (from a computational point of view, not from the development one) FastClick modular implementation is difficult, as the performance experiments will highlight.

Data throughput results are presented in figure 6 for both the stand-alone and the modular implementations, with increasing number of threads and three different configurations. The first configuration, "Hyperthreading," refers to the case in which threads can run on the same cores. The second configuration, "Single Socket," refers to the case in which threads run on different cores in the same socket, while the last configuration, "Dual Socket," refers to the case in which threads run on different cores in different sockets. It is worth noting that when the number of threads exceeds the number of cores, (*i.e.*, 16 in our machines) threads will run on both sockets and share some cores.

The starting point is the very first column, obtained running only one forwarder thread, which shows that the achievable throughput with one single thread is roughly one fourth of the maximum we can achieve, confirming that CPU processing power strongly influences the performance at 10 Gbit/s wire-speed. The following group of columns shows that there is a huge performance gap between two

[2]Note that, as required by DPDK, each forwarding thread is pinned to a physical core at initialization time, thus avoiding thread migration (which can be especially harmful in data-intensive applications).

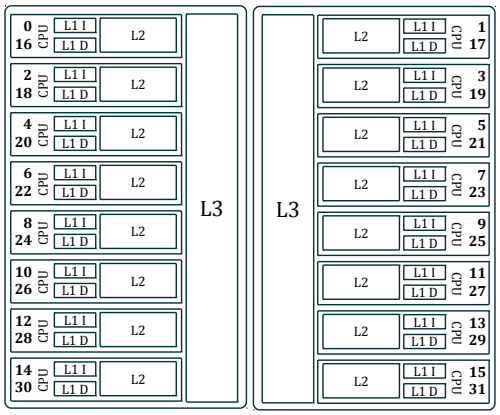

Figure 5: Cores placement and cache hierarchy on the test servers.

threads running with hyperthreading or on independent cores. Threads on independent cores yield approximately double the performance of a single core, while the improvement with hyperthreading is negligible. This suggests that L2 cache plays a critical role in the router's performance. In the third group of columns the throughput approaches what results to be the maximum forwarding capacity (about 10 Mpps) of our servers. The placement uses distinct sockets, and the importance of L3 cache becomes more visible. Four threads on independent sockets reach the maximum throughput, while on the same socket there is a performance decrease of approximately 10%. With six threads on both cores we achieve the maximum throughput, while hyperthreaded threads achieve a performance of about 25% less than the maximum. With eight or ten threads the performance approaches the maximum in all configurations.

Interestingly enough, further increasing the number of threads leads to a small performance degradation due to the increased concurrency of threads on all the caches. In the case of thirty-two threads, the throughput decreases even further, simply because the router is also competing for resources with the operating system.

Looking at the performance of the modular implementation, we can observe a significant difference with respect to its stand-alone counterpart. The first column, in which a single thread is running, shows that the increased flexibility offered by Click introduces some performance degradation due to more complex or less efficient computations. The following columns show that with a sufficient amount of additional resources, the modular version also approaches 10 Mpps, very close to the results of the stand-alone one. Increasing the number of threads beyond 16 results in a steep throughput drop due to the contention on caches introduced by Click, which cannot be completely controlled in this experiment as we instead do in the standalone implementation. Performance degradation with the modular version of *Augustus* can be attributed to Click's modularity that imposes additional overhead in module to module communication. However this problem can be overcome using well known techniques such as click-devirtualize tool and packet batching proposed by FastClick (not used in our evaluations).

Results obtained in the previous set of examples suggests that L2/L3 cache misses represents the main reason for the

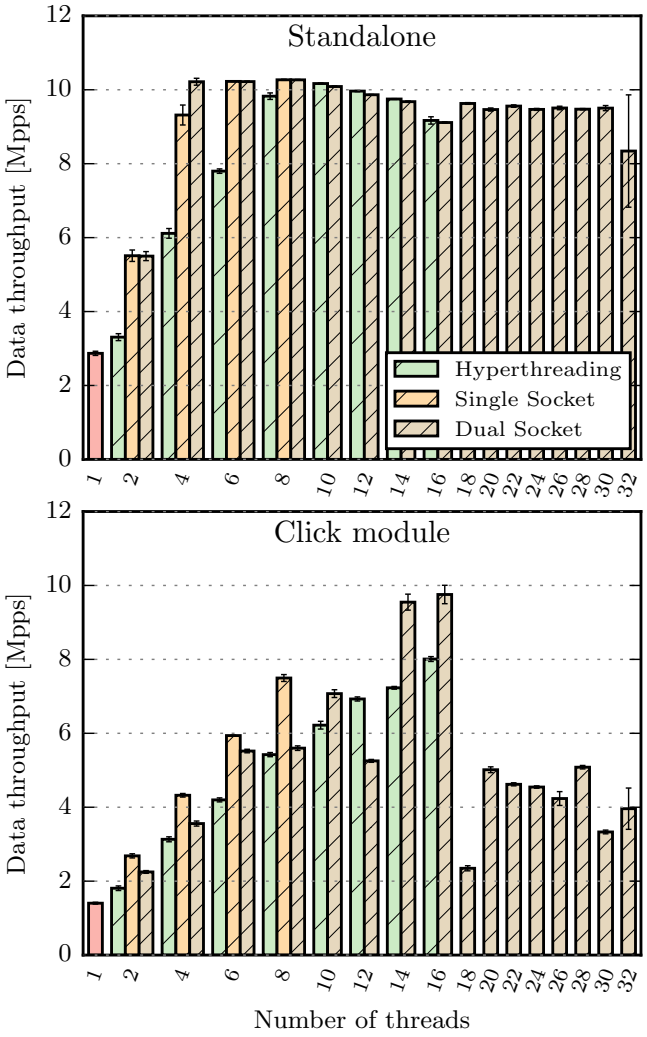

Figure 6: Measured throughput for both versions with varying threads number and placement.

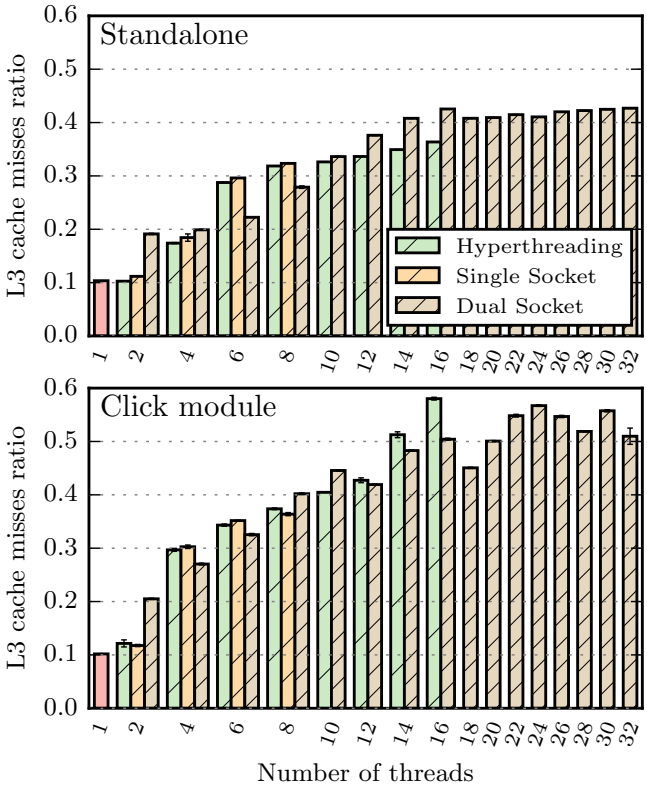

Figure 7: Cache miss ratio for both versions with varying threads number and placement.

observed throughput reduction with the increasing number of threads. To verify our hypothesis we measured L3 cache misses ratio reported in figure 7. The highest throughput performance is obtained with configurations that maintain the competition on caches reasonably low, yet allowing enough computational power. Interestingly, but not surprisingly, the ability of strictly pinning routing threads to a dedicated core greatly enhances performance, and allows for optimized computational power allocations. The modular FastClick implementation instead leads to higher cache competition mainly due to module to module communication overhead. In that case RAM access becomes the bottleneck of the system, leading to loss of performance when too many computational resources are assigned to routing functions.

4.3 FIB size scaling

The size of the FIB is extremely important because it is the interconnection between the control and data planes. The FIB is populated with prefixes by the control plane function that computes the routing table, and it is read by the forwarding engine every time a new interest is not found in the CS or in the PIT. Since ICN control plane is not available, and in any case there would not be an ICN network allowing to measure actually observed prefixes, we fill the FIB with constant minimum size (57 bytes), randomly generated prefixes. The Interest generator produces interests at random in the same prefix space, as there cannot be unmatched prefixes for a forwarding base. As we mentioned already, the worst-case analysis implies following the red and blue paths in figure 1, so that CS and PIT are always analyzed without success and the FIB is accessed. The PIT size is approximately defined by the product between throughput and maximum "latency" (or RTT) admitted for pending interests. We set this latency to 1 s, which is large enough for an operational network: if an interest is not satisfied within 1 s it is reasonable to discard it. Exploring CS size influence is most interesting when coupled with a traffic model, while in a worst case analysis it is never hit. Clearly its size, which we set to 8154 entries, does influence the performance, but it does not seem of much interest when no hits are possible.

The FIB size instead refers to the actual size used to store forwarding information. Hence its size is fundamental to understand the scaling properties of the proposed architecture and implementations. We explored scaling the FIB hash table up to almost 2^{26} buckets (about 4 GB of DRAM, larger values are not achievable with the hardware we use): given the current design with buckets accommodating up to 7 entries in the same cache line, this would support about 130 millions entries keeping the probability of any bucket exceeding the one cache-line threshold below 10^{-3}. The FIB

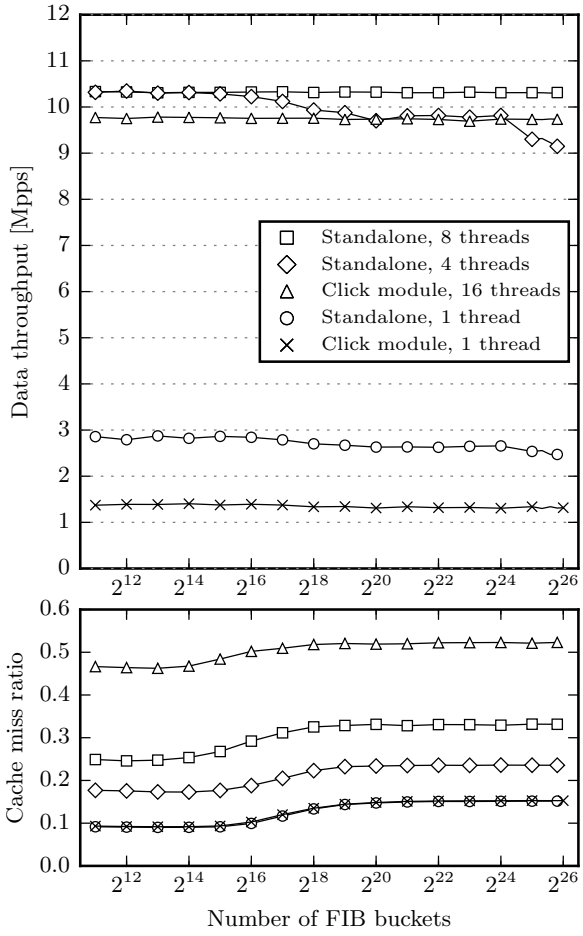

Figure 8: Routing performance and level-3 cache miss ratio as a function of the FIB size.

query runs longest-prefix first in the hash table, so the whole bucket space is always explored, preventing cache-related advantage due to either a small addressing space in the interest flow or correlation in interest packets.

The upper plot in figure 8 summarizes the results, comparing the performance of the single-threaded case and the best-performing multi-threaded case in both implementations. The figure clearly shows that, for all the versions, the overall performance is only marginally affected by the increased FIB size. It also shows that the click version appears to be less sensitive to the increased size. Only the standalone implementation with four threads (dual socket) experienced a measurable throughput degradation, though not very large and also with a trend difficult to explain.

The lower plot in figure 8 shows the L3 cache miss ratio for the same test cases, and indeed explains them. All curves remain constant until about 2^{15} buckets (2 Mbytes) and grow slightly to become again steady again at 2^{19} buckets (32 Mbytes). These two values are related to the size of L2 and L3 caches, 2 and 20 Mbytes respectively. Cache misses refer to every process running and not only to FIB misses. The 4-threads curve does not show a different behavior compared to the others, making it difficult to explain the throughput reduction observed in the upper plot. However, as already observed in [7, 8] the interaction between processes and the cache management is very difficult to predict. One can conjecture that with more threads accessing the FIB, then the cache management tries to keep it more aggressively in L3 cache with eight and sixteen threads than it does with 4. Clearly single thread cases are constrained by other factors. One last observation relates to the large impact of the FIB in cache misses: moving from one to sixteen threads the overall L3 miss ratio grows by a factor of four.

5. DISCUSSION AND CONCLUSION

The Content Centric Networking paradigm is expected to provide a breakthrough in terms of user experience compared to traditional IP networking. Open source free implementations can also boost the emergence of novel networking paradigms as community networks [14]. This paper contributes *Augustus*, a software architecture for a CCN router working at wire-speed in a 10 Gbit/s network and 10 Mpps,and explores its performance providing insight and guidance for further CCN software roueter development.

Specifically, *Augustus*:

- Handles more than 10 millions data packets per second and supports a FIB with up to 2^{26} entries, and it is able to saturate the 10 Gbit/s link with Ethernet payloads as small as 87 bytes;
- Runs both as a stand-alone system, achieving the best performance, or as a set of elements in the Click modular router framework. In the latter case the performance is only slightly penalized.
- Is open source and can be used in software based networks (e.g., leveraging NFV technologies) for fast and incremental ICN deployment.

During the *Augustus* design and implementation we have learned a set of useful "hints" for high-performance CCN software router design and deployment:

- Manual configuration for best performance.
  Augustus provides line rate packet processing, it requires manual configuration to carefully tune its performance. It would therefore be critical to decouple its performance from the hardware configuration, for instance, by adding automated detection of the architecture and consequent smart thread distribution.
- Abstraction hides critical low level properties.
  Software programming for general-purpose hardware involves additional abstraction layers with respect to low level hardware programming, including an operating system and (in the case of Click) the router framework itself. Every layer adds abstraction: on the one hand this eases the development and extensibility, on the other hand each layer makes it harder to track and tune some low-level properties that are crucial for achieving high-performance.
- Complex zero-copy in modular framework.
  In the FastClick implementation it was crucial to guarantee that no packet data copies would be performed throughout the elements. Click provides a very simple API for moving packets across elements that will behave differently depending on whether data copy is

considered needed. In order to guarantee zero copy we had to deeply investigate the API implementation and adapt our code accordingly. Further, We discovered room for optimization in the high-level copying procedure, which resulted in a patch contributed back to FastClick.

A final remark on frame size is due at this very last point. Packet routing and forwarding capability is independent from the packet size, thus 10 Mpps routing capability can sustain about 120 Gbit/s throughput with 1500 bytes Ethernet frame, but much more with large jumbo frames, which can be quite standard in a data-centric architecture, where the data unit is the answer to an interest and not an IP packet. Of course, with very large frames the amount of data would start posing stricter requirements to the I/O system and other bottlenecks may arise; still, the forwarding engine could use the same technology and performance as the one we presented.

Acknowledgements

This work has been partially funded by EIT Digital through the Information-aware data plane for programmable networks project - Activity 15212, and by Technological Research Institute SystemX, within the project "Network Architectures".

6. REFERENCES

[1] S. Arianfar, P. Nikander, and J. Ott. On content-centric router design and implications. In *ACM Workshop Re-Architecting the Internet (ReArch)*, 2010.

[2] T. Barbette, C. Soldani, and L. Mathy. Fast userspace packet processing. In *11th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, 2015.

[3] H. Dai, B. Liu, Y. Chen, and Y. Wang. On pending interest table in named data networking. In *8th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, 2012.

[4] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl. Network of information (NetInf) – an information-centric networking architecture. *Computer Communications*, 36(7):721–735, 2013.

[5] M. Gallo, D. Perino, Z. B. Houidi, and L. Muscariello. Content-centric networking packet header format. Internet-Draft draft-ccn-packet-header-00, Nov. 2014.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *5th ACM Int. Conf. on Emerging Networking Experiments and Technologies (CoNEXT)*, 2009.

[7] A. Kandalintsev and R. Lo Cigno. A Behavioral First Order CPU Performance Model for Clouds' Management. In *4th IEEE Int. Congress on Ultra Modern Telecommunications and Control Systems (ICUMT)*, 2012.

[8] A. Kandalintsev, R. Lo Cigno, D. Kliazovich, and P. Bouvry. Profiling Cloud Applications with Hardware Performance Counters. In *KIISE/IEEE 28th Int. Conf. on Information Networking*, 2014.

[9] J. Kim, K. Jang, K. Lee, S. Ma, J. Shim, and S. Moon. NBA (network balancing act): A high-performance packet processing framework for heterogeneous processors. In *10th ACM European Conf. on Computer Systems (EuroSys)*, 2015.

[10] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek. The Click modular router. *ACM Trans. on Computer Systems*, 18(3):263–297, Aug. 2000.

[11] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *ACM SIGCOMM Computer Communication Review*, 37(4):181–192, Oct. 2007.

[12] H. Lim, D. Han, D. G. Andersen, and M. Kaminsky. MICA: A holistic approach to fast in-memory key-value storage. In *11th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2014.

[13] R. B. Mansilha, L. Saino, M. P. Barcellos, M. Gallo, E. Leonardi, D. Perino, and D. Rossi. Hierarchical content stores in high-speed ICN routers: Emulation and prototype implementation. In *2nd Int. Conf. on Information-Centric Networking (ICN)*, 2015.

[14] L. Maccari, and R. Lo Cigno A week in the life of three large Wireless Community Networks *Elsevier Ad Hoc Networks*, 24(PartB):175âĂŞ-190, Jan. 2015.

[15] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici. ClickOS and the art of network function virtualization. In *11th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, 2014.

[16] M. Mosko. A content-centric networking forwarding design for a network processor. In *IEEE Int. Conf. on Communications (ICC)*, 2015.

[17] D. Perino and M. Varvello. A reality check for content centric networking. In *ACM SIGCOMM workshop on Information-centric networking*, 2011.

[18] D. Perino, M. Varvello, L. Linguaglossa, R. Laufer, and R. Boislaigue. Caesar: A content router for high-speed forwarding on content names. In *10th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, 2014.

[19] W. So, T. Chung, H. Yuan, D. Oran, and M. Stapp. Toward terabyte-scale caching with SSD in a named data networking router. In *10th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, 2014.

[20] W. So, A. Narayanan, and D. Oran. Named data networking on a router: Fast and DoS-resistant forwarding with hash tables. In *9th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, 2013.

[21] W. Sun and R. Ricci. Fast and flexible: parallel packet processing with gpus and click. In *9th ACM/IEEE Symp. on Architectures for Networking and Communications Systems (ANCS)*, 2013.

[22] M. Varvello, D. Perino, and L. Linguaglossa. On the design and implementation of a wire-speed pending interest table. In *IEEE Workshop on Emerging Design Choices in Name-Oriented Networking (INFOCOM WKSHPS)*, 2013

[23] W. You, B. Mathieu, P. Truong, G. Simon, and J.-F. Peltier. DiPIT: a Distributed Bloom-Filter based PIT Table for CCN Nodes. In IEEE, editor, *21th Int. Conf. on Computer Communication Networks (ICCCN)*, 2012.

[24] H. Yuan and P. Crowley. Scalable pending interest table design: From principles to practice. In *IEEE Conference on Computer Communications (IEEE)*, 2014.