A Cluster-based Scalable Router for Information Centric Networks

Andrea Detti, Lorenzo Bracciale, Pierpaolo Loreti, Giulio Rossi, Nicola Blefari Melazzi

Electronic Engineering Dept. University of Rome "Tor Vergata", Italy

Abstract

To support content-oriented services, the routers in Information Centric Networks (ICN) have to provide packet processing functions that are more complex with respect to IP standards, making harder to attain high forwarding rates. The ICN community is working to overcome this issue, designing new software and hardware routers, for setting higher the throughput bar.

In this work, we propose to distribute the workload of a router over several physical machines, resulting in a faster, cluster-based, scalable ICN router (CSR), able to operate with any current software/hardware ICN solution. We also present the specific challenges that the ICN paradigm poses to the design of a cluster router, and show how our proposed CSR deals with them.

The overall forwarding rate can be increased or decreased by suitably dimensioning the number of cluster resources (horizontal scaling) or the performance of individual cluster resources (vertical scaling). When deployed in a cloud environment, the amount of cluster resources can follow the traffic demand, so implementing an elastic ICN router. To assess the feasibility of our approach, we developed a real CSR, based on a new kernel-based ICN load-balancer. The paper includes an extensive set of measurements meant to assess the capabilities and performance of the proposed solution.

Key words: Information Centric Networks, Router, Cluster, Cloud, Scalability, Elastic Services

1. Introduction

Information Centric Networking is a communication paradigm that provides content-oriented functionality and services within the network and at the network layer, rather than relying on end-systems only [1]. Content routing, caching, multicast, mobility, data-centric security, etc. are examples of functionality that Information Centric Networks (ICNs) natively offer through their routers and that, on the contrary, IP networks delegate to hosts, middle-boxes or overlay infrastructures.

Providing content-oriented services at the network layer improves the network efficiency but increases the complexity of packet processing in routers, thus making a challenge the achievement of high forwarding rates. Indeed, an ICN router is a stateful forwarder, carrying out a sequence of complex prefix match operations based on variable length names.

To improve the forwarding rate of ICN routers, researchers focused on software and hardware optimizations [2] [3] [4], such as the recent integration of CCNx (a specific implementation of ICN) with the Vector Packet Processing (VPP) technology [5]. In this paper we explore a complementary approach: rather than optimizing the single ICN router, we distribute its processing burden on more machines. We devise an ICN *Cluster-based Scalable Router* (CSR), externally exposed as a single node, but internally composed of a group of ICN routers connected to a load-balancer (fig. 2). We will show in the paper that the resulting forwarding rate is close to the sum of the internal-router ones.

The CSR architecture is agnostic as the specific technology used by internal-routers, with the only obvious constraint of using the same packet format. The whole cluster logic runs on the load-balancer and un-modified ICN routers can be used within it.

The CSR is able to scale with respect to the forwarding rate, either *horizontally* by adding more internal-routers or *vertically* by improving the forwarding rate of individual internal-routers, e.g., using a faster forwarding implementation or much more powerful hardware.

The CSR components can use real or virtual hardware, e.g. virtual machines (VMs) offered by a cloud. In this case, the number of internal-routers/VMs can be dynamically extended or shrunk to follow traffic demand without service interruption, making it possible to provide elastic ICN routers as-a-service.

Clustered systems are used for several applications, such as databases, server farms, distributed processing, etc., but to the best of our knowledge this is the first paper that explores them for designing scalable ICN routers. And although at a first glance the cluster design may appear as very simple, we found out, while carrying out a practical implementation, that the stateful and complex processing of an ICN router poses several new challenges, which we are going to discuss and face in the rest of the paper. The main contribution of this work are:

- the innovative proposal of a cluster-based scalable ICN router architecture, improving ICN forwarding performance, while preserving in-network caching and multicast functionality and related advantages.
- an open-source Linux implementation, based on NDN (Named Data Networking, a specific ICN solution [6])
- an experimental campaign aimed at showing the capability and performance of the cluster-based scalable router in static and dynamic (elastic) configurations.

2. Related Work

2.1. Information Centric Networks

An ICN is a communication architecture providing users with data items rather than end-to-end communication pipes. The network addresses of an ICN are hierarchical names (e.g. `video/foo/s1`) that do not identify an end host but a data item.

A data item and its unique name form the so called *named object*. A named object is actually a small data unit (e.g. 4kB long) and may contain an entire

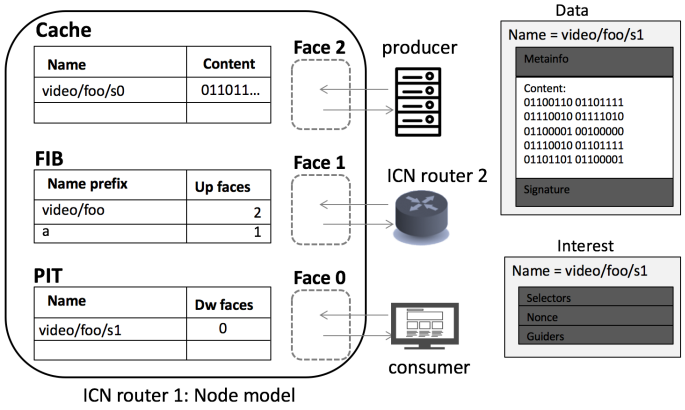

Figure 1: ICN forwarding engine model and packets

content (e.g. a document, a video, etc.), or a chunk of it. The names used for addressing the chunks of a same content have a common prefix (e.g. `video/foo`) followed by a sequence number identifier (e.g. `s1, s2, s3, etc.`).

An ICN is formed by nodes that can be logically classified as consumers, producers and routers. Consumers pull named objects provided by producers, possibly going through intermediate routers.

Any node uses the forwarding engine shown in figure 1 and is connected to other nodes through channels, called *faces*, which can be based on different transport technologies such as Ethernet, TCP/IP sockets, etc.

Data units exchanged in ICN are called *Interest packets* and *Data packets*. To download a named object, a consumer issues an Interest packet, which contains the object name, and that is forwarded towards the producer. The forwarding process is called routing-by-name since the output face is selected through a name-based prefix matching based on a *Forwarding Information Base* (FIB) containing name prefixes (such as `video/foo` and `a` in figure 1) and the corresponding output faces (or upstream faces). The FIB is usually configured by routing protocols, which advertise name prefixes rather than IP subnetworks [7]. During the Interest forwarding process, the node temporarily keeps track of the forwarded Interest packets in a Pending Information Table (PIT), which

stores the name of the requested object and the identifier of the face from which the Interest came from (downstream face).

When an Interest reaches a node (producer or intermediate router) having the requested named object, the node sends back the object within a Data packet, whose header includes the object name. The Data packet is forwarded downstream to the consumer by consuming (i.e. deleting) the information previously left in the PITs, as bread crumbs.

Each forwarding engine can cache (in-network) the forwarded Data packet to serve subsequent requests of the same object. Usually the data freshness is loosely controlled by an expiry approach. Indeed any Data packet includes a metadata reporting the freshness period specified by the producer, which indicates how long the Data can be stored in the cache.

The forwarding engine also supports *multicast distribution*: if a node receives multiple Interests for the same object, the engine forwards the first packet and discards the subsequent ones appending the identifier of the arrival downstream faces in the PIT, if necessary. When a Data packet arrives, the node forwards a copy of it towards each related downstream face in the PIT.

The ICN security is built on the notion of *data-centric security*: the content itself is made secure, rather than the connections over which it travels. The ICN security framework provides each user with a private key and an ICN digital certificate, signed by a trust anchor, and uniquely identified by a name called *key-locator* [8]. Each Data packet is digitally signed by the content owner and includes the key-locator of the digital certificate to be used for signature verification. For access control purposes, Interest packets can be signed too.

An ICN uses *receiver-driven flow/congestion control*. To download a content formed by many chunks, the consumer sends a *flow* of Interest packets, one per chunks, and receives the related flow of Data packets. Flow and/or congestion control are implemented on the receiver side by limiting the number of in-flight Interest packets to a given amount (aka pipeline-size), which can be a constant value or a variable one, e.g. controlled by an AIMD scheme [9]. In an ICN we have end points that exchange Interest-Data sequences and the packet flow is

regulated by the receiver. Dually, in TCP/IP the end points exchange Segment-Ack sequences and the packet flow is regulated by the sender.

2.2. ICN forwarding performances

Since ICN routers have to perform more complex forwarding operations with respect to IP routers, many research works studied forwarding architectures and technical solutions to improve their performance.

For example, in [2], the authors studied the forwarding performance of various ICN prototypes (CCNx [10], CCN-lite [11] and NDN [6]) and presented CCN-par: an enhanced version of CCN-lite, which includes optimized packet queuing disciplines and substitutes all the critical data structures with more efficient hash tables. Conversely in [5], the author proposes the integration of CCNx with Vector Packet Processing (VPP) technology to improve the efficiency of the forwarders.

Other works focus on a specific issue: ICN routers have to work with variable length names, differently from IP routers that deal with the fixed length IP addresses; thus, the lookup in the tables (PIT and FIB) could become critical from the computational point of view. Moreover ICN routing tables are expected to be much larger than IP ones. In this context, in [3] Yuan et al. studied an optimized system lookup for names focusing on the optimization of three critical functions: i) exact string matching with fast updates, ii) longest prefix matching for variable-length and unbounded names, and iii) large-scale flow maintenance. In [12], So et al. designed an NDN [6] packet forwarder using a hash-based algorithm for name lookup and a PIT partitioning technique that can efficiently exploit multicore processors. A special attention has been paid to hash functions to ensure good lookup performance but also protection against DDOS attacks. A different approach has been pursued in [4], where authors present a hardware implementation of the PIT that reduces the overall processing time and in particular the I/O communications with the software control unit.

All these studies aim to design an optimized ICN router by improving the software component, the hardware component or both. The cloud-based router

architecture presented in this work is complementary to those solutions, since it builds on top of them, allowing ICN routers to scale horizontally by sharing the computational burden on multiple machines in an elastic way. As another advantage, the proposed approach is also useful to save energy, as under-used resources can be released, powering them off; indeed, as pointed out by [13], ICN routers power consumption is far from being negligible.

2.3. Cluster-based routers

The idea of a cluster-based router is anything but new; one of the first work in this field dates back to 2001 when in [14] authors studied how to improve the reliability and scalability of IP routers. Usually cluster-based routers are hierarchical machines arranged and connected to each other to offer different paths between the input and output ports. The cluster router concept has been more recently integrated into the NFV philosophy thanks to SDN technology and to the availability of software routers. In this context, IP software routers are commercial reality in NFV systems such as the Ericsson virtual router [15] or the Juniper NFV router [16]. In [17] authors used NFV and SDN to design an elastic IP router for a VPN and built a demonstrator based on Open Virtual Switch and Mininet.

To the best of our knowledge, this is the first work that brings these concepts to the ICN world, presenting an architecture and an implementation of an elastic ICN router, and facing the specific issues and requirements of ICN.

3. Cluster Design

The primary goal of this work is to design a cluster-based scalable router (CSR) that is externally seen as a single ICN router while internally distributing the load on a set of internal-routers by means of a load-balancer. We chose to exploit a fast load-balancer, which performs *simple* and *state-less* (from an ICN point of view) operations to distribute the traffic on *normal* ICN routers, employed as internal-routers. The main constraint of the proposed architecture

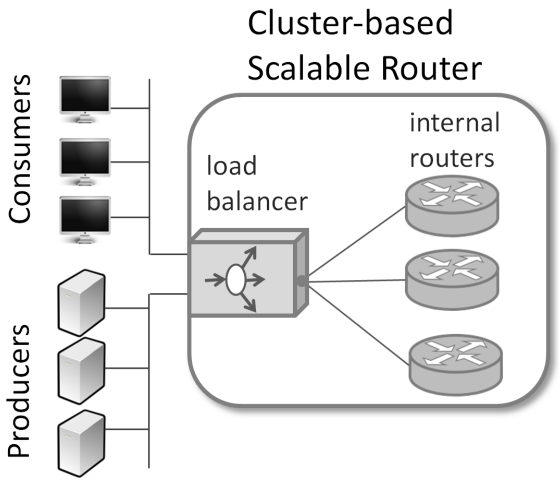

Figure 2: Cluster-Based Scalable Router

is to rely on connection-less faces: for convenience, in this paper we devised the CSR using connection-less UDP faces even tough the extension to Ethernet faces is feasible, as discussed later on.

Fig. 3 presents an high-level view of the system operations: when the load-balancer receives an external Interest (I) or Data (D) packet, it dispatches the packet to an internal-router. Internal-routers have the full FIB[1] and can carry out the actual ICN processing, eventually generating a new ICN/UDP/IP packet addressed to the next ICN node of the path. This packet traverses again the load-balancer before leaving the cluster. Being a possible single point of failure, the load-balancer can be deployed with an high availability configuration e.g. using VRRP protocol (RFC 5798). In what follows we present the logic of the cluster-based scalable router, discussing design principles and the operation flow.

[1]How to handle the routing plane is outside of the scope of this work. Anyway, a possible SDN-like approach is to use a centralized routing entity connected to external routing peers for building the full FIB. Then the full FIB is pushed to the internal-routers.

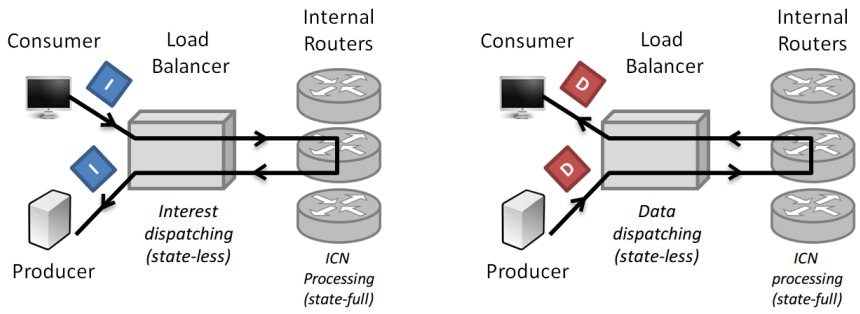

Figure 3: CSR operations

3.1. Design principles

Logic wholly located in the load-balancer - ICN routers are rapidly evolving both in terms of functionality and performance and their implementation is currently under significant development. Our cluster-based router is complementary to this evolution as it is able to accommodate current and future releases of ICN routers without **any modification to the forwarders source codes**. The whole cluster logic (fig. 4) runs on the load-balancer, which accepts incoming faces on a well-known UDP port, e.g. 6363 for an NDN cluster.

State-less and kernel-based load-balancer - To avoid the load-balancer being the router bottleneck, its logic must be very simple and fast, to sustain the aggregate throughput of possibly many ICN internal-routers. This led us to design a **state-less** load-balancer, which dispatches ICN packets by changing on-the-fly their UDP/IP headers (SNAT/DNAT blocks in fig. 4), without any decapsulation/encapsulation work, and never leaving the OS kernel space.

Cross-layer implicit-identifiers of internal-routers - The ICN forwarding model is state-full. Indeed, Data packets are sent back consuming the PIT entry built by the related preceding Interest. In a cluster framework, this implies that the load-balancer must dispatch a Data packet towards the same internal-router that previously served the related Interest packet, otherwise a PIT miss would

take place, disrupting the communication.

This need is in apparent contrast with the state-less nature of the load-balancer. At the arrival of a Data packet, a state-less balancer can not remember which is the internal-router that previously served the Interest, but at the same time has to necessarily send this packet to such internal-router.

To cope with this impasse, our idea is to insert in an outgoing Interest packet an identifier of the serving internal-router and to piggyback this identifier in the returning Data packet, thus enabling proper dispatching. Since we do not want to change neither ICN packets nor the ICN router software, we exploit as *implicit-identifier* the values of the UDP or Ethernet headers that are naturally piggybacked in the Data packets [1], namely: the UDP source port in case of UDP faces, or the source MAC address in case of Ethernet faces. Indeed, these values are memorized in the PIT of external ICN routers and come back in the returning Data packet within the UDP destination port or destination MAC address fields, enabling the correct dispatching to the right internal-router (PORT MAP block in fig. 4, for routers with UDP faces).

This solution does **not require any change in the ICN protocol/router**, but only a configuration of the router UDP port; it has **no impact on external nodes**, since incoming faces are bounded to the single well-known UDP port (6363) or MAC address of the load-balancer; and finally it **also works in case of not-perfect-matching** between Interest and Data names, e.g. those implemented in NDN by MinSuffix/MaxSuffix Interest Selectors, which are often used for content/version discovery purposes.

In the considered implementation of a router with UDP faces reported in fig. 4, we bound each internal-router to a specific UDP port, namely: the internal-router n. x uses the port $637x$ [2].

Hash-based dispatching - The load-balancer evenly distributes the incoming Interest packets to all internal-routers by using a hash function (NAME HASH

[2] In case of Ethernet faces, we do not need such a bounding, since each router already has a different MAC address

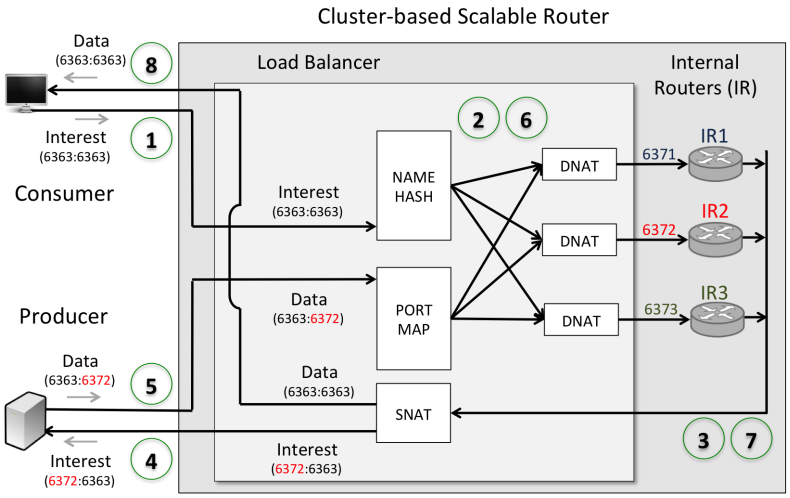

Figure 4: Functional block diagram of the CSR with UDP faces

block in fig. 4), whose input is the Interest name including all the generic name components but not the Nonce (if present). In so doing, the CSR operates in a **work-conserving** way since all internal-routers are involved during the download of different contents, but also during the download of the same content, as chunk names include a different sequence number.

Furthermore, since the same chunk is always served by the same internal-router, the **effectiveness of caching and Interest aggregation (multicast) is preserved**, excepting limited cases of Interest/Data exchanges involving not perfect matching, e.g. for initial version discovery.

We finally observe that our hash-based dispatching requires the full FIB on each internal-router and this may appear as a drawback with respect to approaches based on the partitioning of the name space. By using these alternative approaches, the FIB of an internal-router would contain only the subset of names of the partition assigned to that router. Nevertheless, in so doing a single download would exploit only a single router and, moreover, the internal-router handling popular names (e.g. YouTube) would be overloaded. With our hash-based dispatching all internal-routers are uniformly loaded, even during a single

download. Moreover, the FIB is usually implemented by means of Hash tables, thus the number of entries does not have a relevant impact on the complexity of the forwarding lookup.

Live horizontal scaling - The number of internal-routers can change, especially in the case of cloud deployments, so that the cluster can self-adapt its allotted resources to respond to the variation of traffic demand. The CSR supports live horizontal scaling: the addition or removal of an internal-router happens with **zero downtime**, i.e., without any service interruption and packet loss. The insertion of a new internal-router (scale-out) only requires the activation of the new router machine and then a simple hot reconfiguration of the NAME HASH and PORT MAP modules of the load-balancer. After the scaling operation, new Interest/Data packets will use the new set of internal-routers; Data packets that were in-flight at the scaling time will be properly served consuming the PIT entries of the old set of internal-routers, thanks to the usage of the UDP port as internal-router identifier.

Conversely, the removal of an internal-router happens in two stages: in a first stage, the NAME HASH must be reconfigured to restrict the set of internal-nodes, excluding the router label IR_x from receiving new Interest packets; then, in a second stage (typically after a max-network-RTT), router IR_x can be switched off. In this way any pending Data can be properly served before shutting down the involved machine.

3.2. Sample operation flow

Let us now discuss a simple exemplary Interest/Data exchange by commenting fig.4. The figure reports also source and destination UDP ports (s:d) used by Interest and Data packets. Without lack of generality, we assume that the CSR is located between a consumer and a producer, whose receiving UDP ports are the NDN default ones (6363).

The Consumer has a UDP face connected to the IP address and port (6363) of the load-balancer and sends an Interest through that face (step 1). The Interest is received by the load-balancer and traverses a hash function (NAME

HASH) that, on the base of the Interest name, selects the internal-router n.2 to serve the packet (step 2). Using a DNAT operation, the Interest is forwarded, by replacing the destination IP address and the destination UDP port with the ones of the selected internal-router (6372).

The internal-router n.2 forwards the Interest towards the producer within a new UDP/IP packet, whose source port is 6372 and destination port is 6363 (step 3). The UDP/IP packet traverses again the load-balancer that only has to change the source IP address with its own IP address (SNAT), before sending out the packet to the Producer (step 4).

When the Producer receives the Interest, it sends back the Data packet towards the load-balancer within a UDP/IP packet whose source port is 6363 and destination port is 6372 (step 5). The Data packet enters the load-balancer and is processed by the PORT MAP module which, on the basis of the destination UDP port, forwards the packet to the internal-router n.2, which previously served the related Interest packet (step 6).

The internal-router n.2 receives the Data, consumes the PIT entry left by the previous Interest and forwards the Data back to the consumer within a new UDP packet that traverses the load-balancer (step 7). The balancer changes the IP source address with its IP address (SNAT) and the source UDP port with the default one that in this case is 6363 (step 8). Finally the Data is properly received by the Consumer.

4. Performance assessment

In this section, we report a set of experiments to illustrate the capabilities of the proposed architecture, as well as the performance gain made possible by our solution.

4.1. Experimental setup

We ran our experiments in a controlled virtualized environment composed of Linux-based virtual machines (Ubuntu 16.04, kernel 4.4.0) and Linux bridges.

The hosting server has 16 cores (32 CPUs, hyper threading 2) and 64GB of RAM, and uses a KVM hypervisor with paravirtualized network drivers (Virtio) and hardware acceleration.

The testbed architecture is shown in figure 5. We have a set of consumers and producers on two different networks, which are interconnected by a cluster-based scalable router. The networks between the consumers and the CSR, the CSR and the producers, the load-balancer and the internal-routers are implemented by Linux bridges, each one providing a raw throughput close to 30 Gbit/s. All virtual machines run an ICN forwarder, namely NFD of the NDN's v0.6.0 platform [6], with the exception of the load-balancer running only our ICN kernel-based software[3].

Each producer has 1 virtual CPU, it uses the utility `ndnputchunks` to publish a content of 300MB, whose chunks have a size of 4400 bytes and is identified by a unique name prefix: producer #1 publishes a content named `a`, producer #2 a content named `b`, and so forth.

Each consumer has 2 virtual CPUs, its ICN FIB has a default entry ("/") pointing the load-balancer through a UDP face, and it uses the `ndncatchunks` utility with `pipelinesize=16` to retrieve contents. Each internal-router has 1 virtual CPU and its ICN FIB an entry for each content name prefix (a,b,...), properly pointing the related producer with an UDP face. The load-balancer has 1 virtual CPU and runs our software, which is composed of a set of plain `iptables` rules supported by a new **iptables NDN extension** carrying out matching and hashing (Jenkins one-at-a-time) operations on NDN packet fields. To limit as much as possible the interaction among processes runnining on different VMs, we pinned each virtual CPU on a different core, anyway for those

[3] A more realistic scenario provides that the CRS is deployed in a cloud, interconnecting related VMs by a virtual network offered by the cloud, and configuring the load balancer with necessary public (floating) IP addresses. All the considered ICN faces are supported by UDP sockets, thus consumers and producers can be located anywhere on Internet, and can directly interconnect the CSR router or could be served by other ICN routers, in turns connected to the CSR with UDP faces

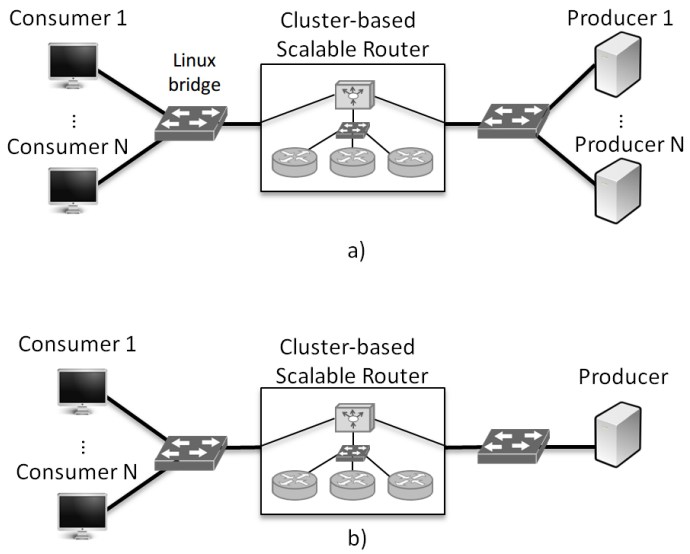

Figure 5: Network evaluation scenarios. Different consumers concurrently download a content provided by a) different producers b) a single producer.

tests with more virtual CPUs than real cores this was not possible and few interaction effects show up in the performance as discussed later on.

All the technical details relevant to the cluster configuration are presented in the technical annex [18], including also the open-source version of the iptables NDN extension.

4.2. Load balancing

In this section we show the load balancing performance, focusing on the throughput improvement and work-conserving behavior of the CSR. For the consumers and producers, the throughput is measured as the sum of their input and output traffics (Interest+Data), including IP/UDP headers. Internal-routers handle a double amount of consumer/producer traffic since any packet enters and exits. As a consequence, to make easier the comparison, the internal-router throughput is measured considering only traffic from/to consumers.

We considered the scenario depicted in figure 5(a) in which different consumers concurrently download contents provided by different producers. Specif-

ically, the consumer $\#x$ downloads the whole sequence of chunks of the content provided by the producer $\#x$.

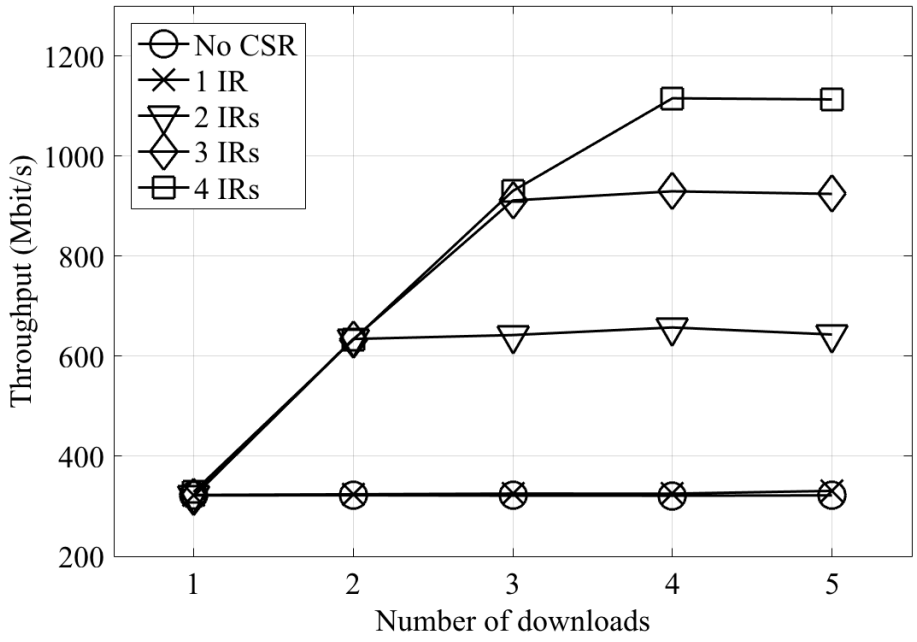

Figure 6: Aggregated consumer throughput vs. number of downloads, for different number of internal-routers (IR), without caching.

Figure 6 shows the aggregated throughput of all consumers varying the number of active downloads (N_{down}) and the number of available internal-routers (N_{IR}). ICN caching is disabled.

We begin by considering a "No CSR" case in which we replace the whole CSR of figure 5(a) with a plain NDN router. We achieve an overall consumer throughput in the order of 300 Mbit/s and we observe that this **throughput limit is due to the ICN forwarding complexity** that brings the CPU load of the router close to 100%. In case of a single download, also the CPUs of the consumer and of the producer are close to 100%, since all involved ICN forwarders support the same packet forwarding rate. Overall, we conclude that in

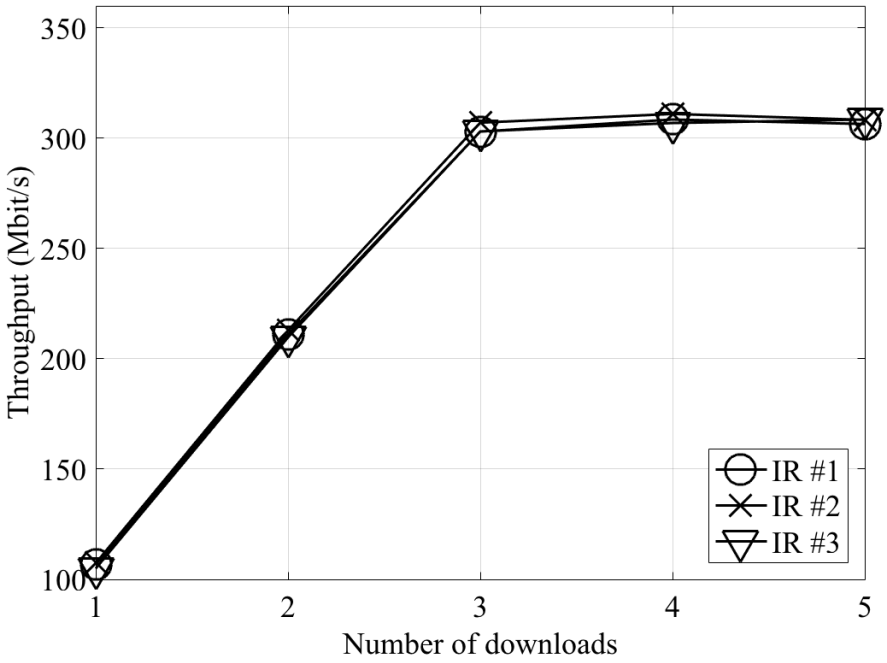

Figure 7: Aggregated consumer throughput handled by a single internal-routers (IR #x) in case of a CSR with three internal-routers vs. number of downloads, without caching.

our configuration there is an inherent *ICN forwarding rate* limitation to roughly 300 Mbit/s due to a CPU overload caused by the complexity of the ICN forwarding processing.

Let us now analyze the performance of the proposed cluster router for the different configurations (1 IR, 2 IRs, etc.). Since any ICN forwarder is CPU limited to 300 Mbit/s, it follows that the expected aggregated throughput of all consumers is close to $300 \cdot \min(N_{IR}, N_{down})$ Mbit/s. Indeed, when $N_{IR} > N_{down}$ the CPUs of end-point nodes limit the overall throughput; conversely when $N_{IR} < N_{down}$ the CPUs of internal-nodes limit the overall throughput. The load-balancer does not implement any ICN forwarder, but just a low complex ICN packet dispatcher, based on IP/UDP kernel processing, and thus its CPU is mostly idle compared to the one of other nodes.

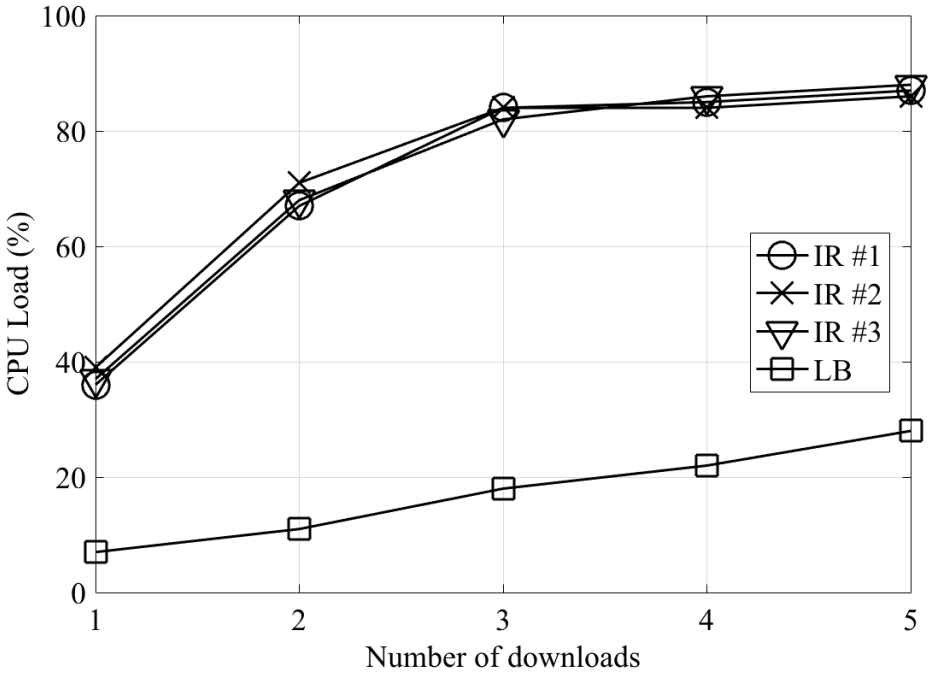

Figure 8: Internal-routers and load-balancer CPU load in case of a CSR with three internal-routers vs. number of downloads, without caching.

From figure 6, we firstly observe that for the extreme case of a CSR with a single internal-router (1 IR) preceded by the load-balancer, we obtain the same performances of the No CSR case, confirming the **transparent behavior of the load-balancer** on the system performance.

When we increase the number of internal-routers, we observe that **the throughput increases almost linearly with the number of internal-routers**, e.g., up to 1150 Mbps with 4 internal-routers[4]. Moreover, the maximum throughput is practically reached with a number of downloads (i.e. con-

[4]We observe that in cases where the number of involved virtual CPUs get close with the number of the cores of our server, processes interactions take place and the performance increase is a little bit sub-linear. For instance in case of 4 IRs and 4 downloads, we expect a throughput in the order of 1.2 Gbit/s but we actually got a lower value.

sumer/producer couples) greater or equal to the number of internal-routers.

Besides confirming the expected maximum throughput behavior, these results show also that having several internal-routers is not always necessary, since the bottleneck may be elsewhere. For instance, in case of two downloads, the throughput does not significantly change by increasing the number of internal-routers beyond 2 because, over this threshold, the bottleneck moves from the CSR CPUs to the consumer/producer CPUs. This further motivates the deployment of the CSR in a cloud environment, to dynamically adapt the cluster size also taking into account this phenomenon.

In figures 7 and 8 we consider 3 internal-routers and we vary the number of downloads, to show the load-balancing **work-conserving behavior**. Figure 7 shows the portion of the aggregated consumer throughput handled by each internal-router, while figure 8 shows the related CPU load in percentage. The traffic is fairly distributed on all available internal-routers, corresponding to a load equally balanced on all the computational resources available in the cluster. And this also takes pace in case of a single download because dispatching is on a chunk (Interest/Data) basis and a single download involves many chunks. At the same time, the computational burden of the load-balancer, even with five downloads, is sensibly less than that of any one of the internal routers, showing that our **load-balancer is not a processing bottleneck**.

4.3. Heterogeneous internal-nodes

In a real environment, system designers have to cope with heterogeneous physical machines, e.g. with different CPUs. In that scenarios the load-balancer should distribute the ICN network traffic to the different IRs proportionally to their ICN throughput capability, i.e. in a non-uniform way. Since the load-balancer packet dispatching module uses a hash function to select the serving IR, a non-uniform traffic distribution can be merely achieved though a non-uniform mapping of the hash space over the Internal nodes.

To assess the effectiveness of a non-uniform dispatching, we considered the scenario of figure 5(a) with three IRs, but now we impose a limit of 170 Mbit/s

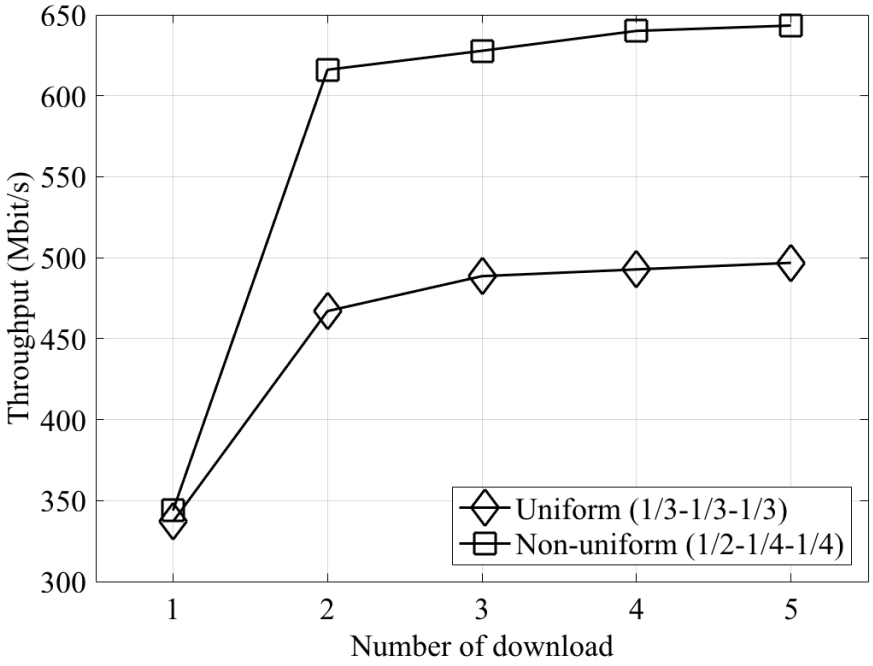

Figure 9: Aggregated consumer throughput for a uniform and non-uniform traffic distribution vs. number of downloads in case of a CSR with three internal-routers and limiting to 170 Mbit/s the network throughput of the LB-IR1 and LB-IR2 links

to the network throughput of the LB-IR1 and LB-IR2 links by using Linux Traffic Control tools, to emulate the presence of IRs with different ICN throughput capability. In so doing, IR1 and IR2 have nearly the half of the ICN throughput capacity of IR3. In figure 9 we reports the aggregated throughput of all consumers using a uniform and an non-uniform packet dispatching. In the non-uniform case, the hash space is partitioned so that IR3 handles 50% of the traffic, while IR1 and IR2 handles 25% each, thus matching the throughput heterogeneity. Results show that the non-uniform traffic dispatching allows to achieve an higher throughput. Indeed, due to the consumer flow control mechanism, in the uniform case IR1 and IR2 act as bottlenecks and IR3 is underutilized. On the contrary, as showed in figure 10, introducing a proper non-uniform traffic dis-

tribution, all IR can reach their maximum throughput capability, consequently increasing the total consumer throughput.

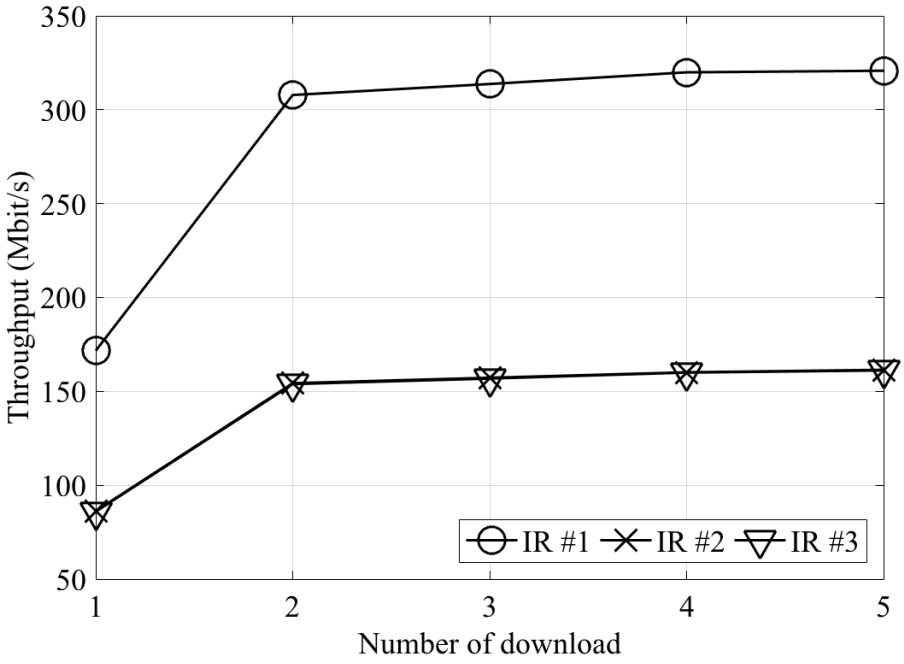

Figure 10: Aggregated consumer throughput handled by a single internal-routers (IR #x) in the non-uniform traffic distribution

4.4. Preserving ICN features

In this section we test if caching and multicast ICN features are preserved by the proposed CSR architecture.

For what concerns multicast, we consider the the scenario depicted in figure 5(b), in which 5 consumers concurrently download the *same* content provided by a single producer. In figure 11 we show the throughput measured on consumers, producer and internal-routers (towards consumers). As we can see, the producer traffic is equal to the one of a single consumer confirming that **the CSR preserves the ICN multicast functionality**. Furthermore, the sum of consumer throughputs is equal to the sum of internal-router throughput being

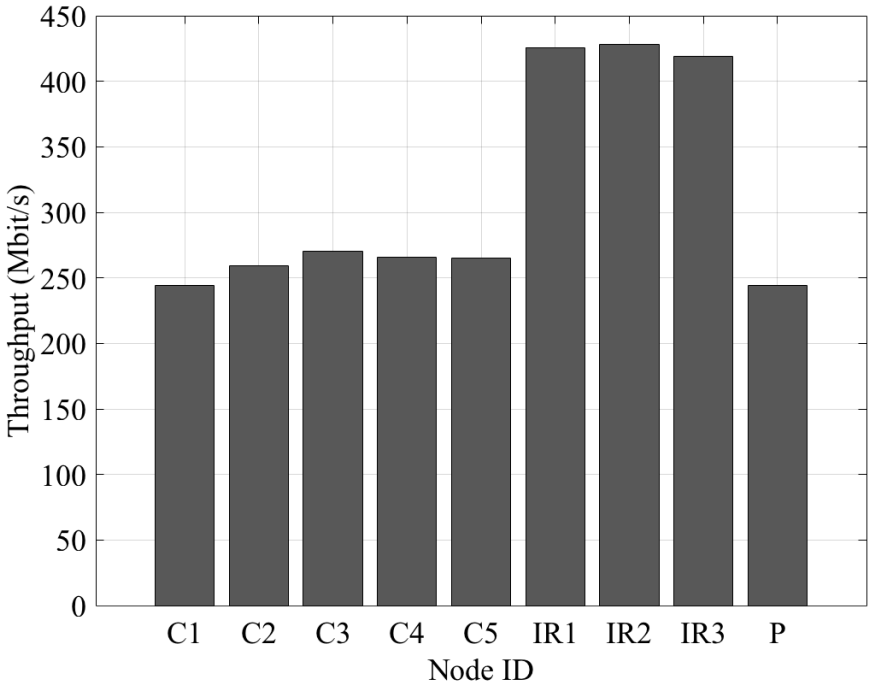

Figure 11: Throughput of 5 consumers (C), producer (P) and internal-nodes towards consumers (IRx) in case of a multicast test and a CSR with 3 internal-nodes

the CSR the multicast splitting point. We also point out that in this case the internal-routers achieve a higher throughput performance with respect to figure 7 because the multicast delivery decreases the number of forwarded packets between consumers and produces thus reducing the CPU load per packet.

To assess the caching functionality, we enable caching on the internal-nodes, and set up an experiment where a single consumer requests two times the same content. When the test starts, the caches of the internal-nodes are empty. In figure 12 we show how the throughput varies on the consumer and on the producer versus time. As we can see, content is transmitted only one time from the producer, since the second request is completely served by the caches of internal-routers, thus demonstrating that **the CSR preserves caching functionality**. Fig. 13 reports the cache size of all internal-routers. During the

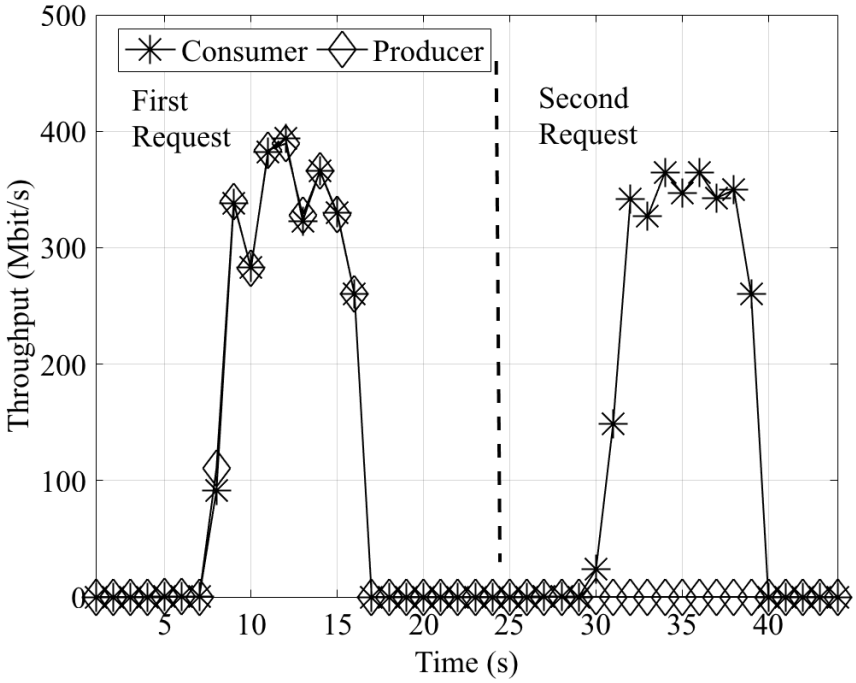

Figure 12: Throughput of a consumer and of a producer during the caching test and a CSR with 3 internal-nodes

first download, we observe a **fair caching occupation**, indeed during a single content download all the IRs are almost equally involved, because the load-balancer dispatching works on a chunk (i.e. Interest/Data) basis. It follows that, by caching received chunks (Data packet) , also the caches of internal-routers will be equally used. During the second download the cache occupation remains stable since all Data packets are served by the caches.

We finally observe that, the resulting distributed caching strategy has a drawback in case of non-perfect-matching: indeed the CSR may not return a valid cached data because not available on the cache of the internal-router selected for processing the Interest. However, this inefficiency should have a limited impact on real systems, since these situations of non-perfect-matching are rare with respect to perfect-matching cases and mostly occurring at the start

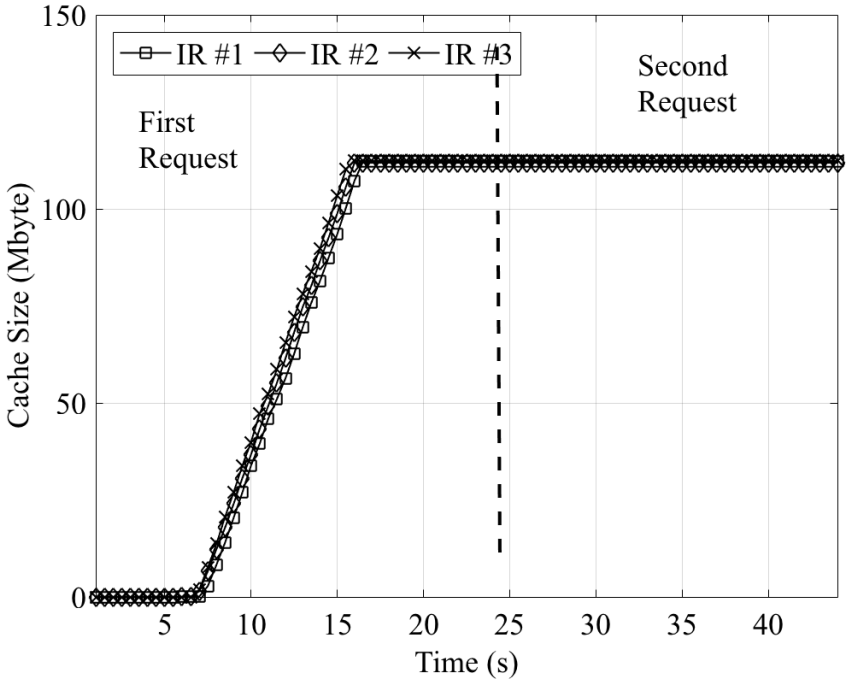

Figure 13: Cache size of internal-nodes during the caching test and a CSR with 3 internal-nodes

of a download for version discovery purposes.

4.5. Live horizontal scalability

In the following experiment we test the ability of the CSR to modify the number of internal-routers without any service interruption. To this aim, we request 5 consumers to execute long lasting downloads of different contents, and we vary the number of internal-routers during the data transfer (*live scaling*). Caching is disabled on all nodes.

Figure 14 shows the aggregated throughput as seen by all the consumers. In the left-hand part of the plot we perform a scale-out operation by varying the number of internal-routers from one to three, making a change every 20 seconds. In the right-hand part of the graph we perform the scale-in procedure by reducing the number of internal-routers from three back to one. We observe how

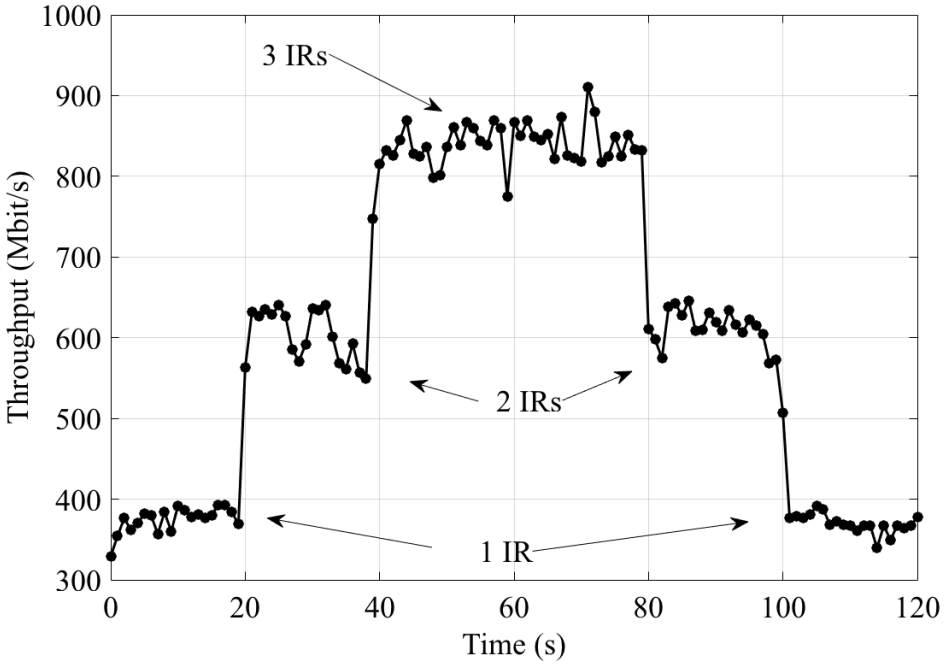

Figure 14: Consumer aggregated throughput dynamically varying the number of internal routers.

the throughput promptly follows the hot cluster re-configurations. Moreover, to verify the **absence of packet loss**, in these tests we used a configuration of `ndncatchunks` that does not allow retransmissions. Consequently, if there had been a packet loss then the communication would have been interrupted, but in our tests all transfers have been successfully completed.

5. Conclusion

In this work we presented a cluster-based scalable router (CSR), a novel architecture that allows ICN routers to scale horizontally and vertically, exploiting resources available on different machines to achieve better throughput performance, while preserving all ICN features. This approach is complementary with respect to solutions pursued in the literature, which focus on pushing

further the performance of a single ICN forwarder through software/hardware improvements. Besides a functional architecture, we presented a working and efficient implementation based on a Linux kernel module. We also conducted an extensive set of measurements devised to illustrate the capabilities of the presented solution, and to analyze the achieved performance. Our approach allows implementing elastic ICN routers able to self-adapt to a time varying traffic demand, acquiring and releasing resources, and guaranteeing a given performance standard. This is particularly important when the solution is used in a cloud environment, where virtual resources (e.g. NFV) can be provided on demand.

Acknowledgements

This work was partly funded by the EU-JP H2020 ICN2020 project.

References

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, R. L. Braynard, Networking named content, in: Proceedings of the 5th international conference on Emerging networking experiments and technologies, ACM, 2009.

[2] A. S. Khatouni, M. Mellia, L. Venturini, D. Perino, M. Gallo, Performance comparison and optimization of icn prototypes, in: Globecom Workshops (GC Wkshps), 2016 IEEE, IEEE, 2016, pp. 1–6.

[3] H. Yuan, T. Song, P. Crowley, Scalable ndn forwarding: Concepts, issues and principles, in: Computer Communications and Networks (ICCCN), 2012 21st International Conference on, IEEE, 2012, pp. 1–9.

[4] W. Yu, D. Pao, Hardware accelerator to speed up packet processing in ndn router, Computer Communications 91 (2016) 109–119.

[5] Cicn project, https://wiki.fd.io/view/Cicn, accessed: 2017-05-03 (2017).

[6] Ndn project, http://named-data.net/, accessed: 2017-05-03 (2017).

[7] A. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, L. Wang, Nlsr: named-data link state routing protocol, in: Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking, ACM, 2013.

[8] Y. Yu, A. Afanasyev, D. Clark, V. Jacobson, L. Zhang, et al., Schematizing trust in named data networking, in: Proceedings of the 2nd International Conference on Information-Centric Networking, ACM, 2015, pp. 177–186.

[9] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, N. Blefari-Melazzi, Transport-layer issues in information centric networks, in: Proceedings of the second edition of the ICN workshop on Information-centric networking, ACM, 2012, pp. 19–24.

[10] Ccnx project, http://blogs.parc.com/ccnx/, accessed: 2017-05-03 (2017).

[11] Ccn-lite project, http://ccn-lite.net/, accessed: 2017-05-03 (2017).

[12] W. So, A. Narayanan, D. Oran, Named data networking on a router: Fast and dos-resistant forwarding with hash tables, in: Architectures for Networking and Communications Systems, 2013, pp. 215–225. doi:10.1109/ANCS.2013.6665203.

[13] K. Ohsugi, J. Takemasa, Y. Koizumi, T. Hasegawa, I. Psaras, Power consumption model of ndn-based multicore software router based on detailed protocol analysis, IEEE Journal on Selected Areas in Communications 34 (5) (2016) 1631–1644. doi:10.1109/JSAC.2016.2520278.

[14] J. Ge, H. Qian, Cluster-based virtual router, in: Info-tech and Info-net, 2001. Proceedings. ICII 2001-Beijing. 2001 International Conferences on, Vol. 2, IEEE, 2001, pp. 102–109.

[15] Ericcson virtual router, https://www.ericsson.com/ourportfolio/products/virtual-router, accessed: 2017-04-20 (2017).

[16] Juniper nfv solution, https://www.juniper.net/uk/en/solutions/nfv/, accessed: 2017-04-20 (2017).

[17] S. Van Rossem, W. Tavernier, B. Sonkoly, D. Colle, J. Czentye, M. Pickavet, P. Demeester, Deploying elastic routing capability in an sdn/nfv-enabled environment, in: Network Function Virtualization and Software Defined Network (NFV-SDN), 2015 IEEE Conference on, IEEE, 2015, pp. 22–24.

[18] Ndn iptables match extension - technical annex and linux kernel module reference manual, `http://netgroup.uniroma2.it/Andrea_Detti/papers/conferences/CSRappendix.pdf` (2017).