A Framework for Experimenting ICN over SDN Solutions using Physical and Virtual Testbeds

G. Siracusano, S. Salsano, P.L. Ventre, A. Detti, O. Rashed, N. Blefari-Melazzi

University of Rome "Tor Vergata" / CNIT, Rome (Italy)

Keywords:

Information Centric Networking; Software Defined Networking; Testbed; OpenFlow; Open Source; Emulation.

ABSTRACT

Information Centric Networking (ICN) is a paradigm in which the network layer provides users with access to content by names, instead of providing communication channels between hosts. The ICN paradigm promises to offer a set of advantages with respect to existing (IP) networks for the support of the large majority of current traffic. In this paper, we consider the deployment of ICN by exploiting the Software Defined Networking (SDN) architecture. SDN is characterized by a logically centralized control plane and a well-defined separation between data and control planes. An SDN-enabled network facilitates the introduction of ICN functionality, without requiring a complex transition strategy and the re-deployment of new ICN capable hardware. More in details, in this paper we provide: i) a solution to support ICN by exploiting SDN, extending a previous work of ours; ii) design and implement an open reference environment to deploy and test the ICN over SDN solutions over local and distributed testbeds; iii) design and implementation of a set of Caching policies that leverage on the ICN over SDN approach; iv) performance evaluation of key aspects of the ICN over SDN architecture and of the designed caching policies. All the source code and the monitoring suite are publicly available. To the best of our knowledge, there are no other similar solutions available in Open Source, nor similar emulation platforms, including also a comprehensive set of monitoring tools.

1. Introduction

Information Centric Networking (ICN) is a paradigm emerged to overcome some intrinsic limitations of the IP protocol [1][2]. In ICN, the network provides users with access to content by names, instead of providing communication channels between hosts. The idea is to provide "access to named data" as the fundamental network service. This means that all content (e.g. a document, a picture) is given a name; then, users request for the named content, the network forwards the requests toward the "closest" copy of such a content, which is delivered to the requesting user. With ICN, the communication network becomes aware of the name of the content that it provides and the routing decisions are made based on the content name. As a result, ICN [3]: i) improves network efficiency; ii) naturally supports mobility of users and servers and multicast communications; iii) eases the operation of fragmented networks, or sets of devices disconnected from the rest of the network; iv) offers simpler application programming interfaces; v) provides a content-oriented security and access control model.

The capabilities of ICN are particularly valuable as we move to an increasingly mobile connected world, where information, end-points and people are continually connecting to a different point, requiring in-built mobility support from the network. The Internet's coupling of the IP address for both identifying a device (and related content) and for determining where it is topologically located in the network resulted in conflicting goals. On one hand, for routing to be efficient, the address must be assigned topologically; on the other hand in order to manage collections of devices, without the need for renumbering in response to topological change or mobility events, the address must not be explicitly tied to the topology [4]. ICN offers a clean solution, by logically separating network locators from identifiers, not only of devices but also of content and potentially of users and functions.

Despite the widespread attention that ICN has received from researchers in the past decade, both in terms of papers and research projects (see the section 9 on related work), the area is still facing significant research and innovation challenges, including innovative applications, interplay with cloud and virtualization concepts, name to location resolution, routing/forwarding table scalability.

One of the open issues is the deployment of an ICN infrastructure in the current networks, based on the IP protocol, as it may require the replacement or update of existing running equipment. In this regard, we believe that Software Defined Networking (SDN) [5][6][7][8] can be an important enabler of ICN, as it promises to facilitate the continuous evolution of networking architectures. The SDN architecture is characterized by a logically centralized control plane and a well-defined separation between data and control planes. Forwarding devices execute packet forwarding actions following rules installed by an SDN Controller. The logical interface between the Controller and the forwarding devices is called *southbound* interface. An SDN-enabled network could facilitate the introduction of ICN functionality, without requiring a complex transition strategy and the re-deployment of new ICN capable hardware.

In [9] we introduced an ICN over SDN network architecture to deploy ICN in IP networks. Here we build on and extend that work, providing the following main novel contributions:

1. Enhancements in the design of the ICN over SDN forwarding mechanisms to support mesh topology and to scale with the size of the topology.
2. Design and implementation of an open reference environment to deploy and test the ICN over SDN infrastructure and the related network service over local and distributed testbeds.
3. Design and implementation of a set of Caching policies that leverage on the ICN over SDN approach.
4. Performance evaluation of key aspects of the ICN over SDN architecture and of the designed caching policies.

The paper is organized as follows. Sections 2 and 3 provide background information respectively on the basic ICN solution called CONET [13] and on our ICN over SDN architecture [9]. Section 4 presents the detailed description of the forwarding

mechanism implemented in our Proof of Concept (ICNoSDN v1 PoC), which were not described in [9]. This is needed to understand the motivations and operation advances of the new ICNoSDN v2 PoC, which is described in Section 5. The novel contributions include automatic topology handling that allows managing large numbers of nodes, a flow forwarding mechanism that works in arbitrary mesh topology; an improved architecture of the Controller software that simplifies the introduction of new mechanisms. In section 6 we define a set of specific caching policies implemented as Controller logic. In section 7, we illustrate the emulation platform including the novel monitoring GUI, fully supported and tested in the OpenFlow v1.0 testbed provided by the OFELIA project [10] and in the Mininet emulation tool [11]. The framework and the tools are Open Source and available at [12]. The described environment can be used to test new mechanisms in the ICN over SDN architectural scenario. Section 7.4 in particular provides the needed indications and references for reusing the ICNoSDN v2 PoC. In order to ease the initial setup of the solutions, we also packaged everything in a ready-to-go virtual machine. To the best of our knowledge, there is no such effort readily available in Open Source, nor such an emulation platform with a comprehensive set of monitoring tools. Section 8 provides the performance evaluation of the caching policies implemented in the ICN over SDN architecture. The related works are discussed in section 9; finally, section 10 draws the conclusions.

2. The ICN approach: CONET

CONET [13][14] is based on the concepts introduced in Content Centric Networking/Named Data Networking (CCN/NDN) architectures [2][15]. It extends the approach proposed by CCN/NDN in several aspects, including integration with IP, routing scalability, transport mechanisms, inter-domain routing. For the reader's convenience, the basics of the CONET solution are reported hereafter, please refer to [13][14] for a detailed description. The "terminals" are called ICN Clients and ICN Servers in analogy with a client/server architecture or a publish/subscribe scheme. ICN Clients request content using CONET protocols as transport solution, leveraging the unique name of the content to be received, while the ICN Servers are the originators/providers of the content. In general, a terminal can act as both ICN Client and ICN Server if needed. As for the naming schemas, there is a full support of the approaches proposed by CCN/NDN; names could be human-readable or self-certifying.

The Forward-by-name operation, performed by ICN Nodes, consists in a name-based lookup table and on a prefix matching algorithm. The association between name prefixes and next hop is performed by using a table called FIB (Forwarding Information Base), this table must be accessed at line speed. Moreover, another table, called RIB (Routing Information Base) is used to exchange routing information with other nodes and it does not need to be accessed at line speed. The RIB and FIB could have tens of billions of entries in order to include all the possible content names, making it infeasible to implement both in router hardware. In CONET, the FIB is used as a cache of currently needed routes, while the full routing table is managed by a centralized routing logic (Name Routing System). The Name Routing System has some similarity with the Domain Name System, as it provides the resolution of a content name into a next-hop, while the DNS provides the resolution of a domain name into an IP address.

The content requests are called *interests* and the related packets are called *interest packets*. The *interest* packets are forwarded over the ICN network, taking into account the requested content-name for the "routing" of the request. The request travels in the network until it reaches a node that contains the content. This node receives the *interest* packet and replies with the *data* packets that are sent back towards the requesting node. The latter ones follow back the path towards the requester. Intermediate nodes can store the content, performing transparent "in-network" caching, following the CCN approach. In order to fit the transfer units of under-CONET technologies (e.g. Ethernet), CONET handles the segmentation of content using two levels: at the first level the content is segmented into chunks, at the second level chunks are segmented into smaller data units (called Carrier-Packets). The transfer of Carrier Packets is regulated by a receiver-driven transport protocol based on the well-known TCP congestion control mechanism [16]. This approach avoids the fragmentation at IP level, which is faced by chunks greater than 1500 bytes in the earliest implementations of CCN. Moreover, it helps in the SDN solution as the CONET carrier packets can be properly managed by SDN capable switches thanks to the information contained in the headers, while IP fragments resulting from the fragmentation of a chunk would lose all the needed information.

3. ICN over SDN architecture and high level view the Proof of Concept

The architectural concepts for the deployment of the CONET architecture over a Software Defined Network are introduced in [9]. Here we briefly recall them. Following the SDN approach, an OpenFlow-based ICN architecture is considered, where the intelligence of the ICN is de-coupled from the forwarding (of *interest* and *data* packets) and caching functions. As shown in Figure 1, this architecture is composed of two different planes: i) a data plane containing ICN Servers, ICN Clients and ICN Nodes; ii) a control plane that includes the Name Routing System (composed by NRS Nodes), SDN controllers and an Orchestrator node. The two planes communicate through an extended OpenFlow interface, used by the Controllers/NRS nodes to control one or more ICN Nodes. In the control plane, the Controllers/NRS nodes offer also a *northbound* API towards the Orchestrator node that orchestrates the overall behavior of a domain. Note that the role of NRS nodes in CONET is fully aligned with the SDN approach of using a controller to drive the forwarding behavior of switches/routers and to enforce an explicit separation between a data forwarding plane and a control plane.

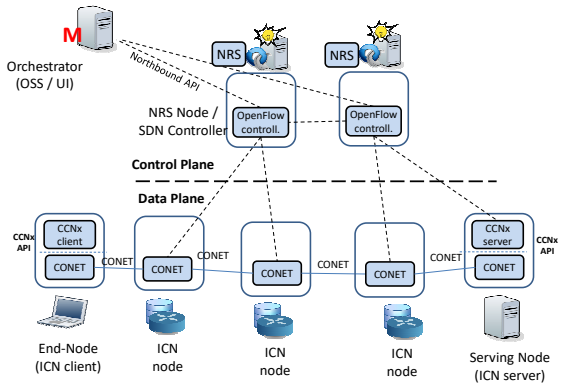

Figure 1 - Architecture for ICN over SDN based on CONET

The reference scenario for our testbed implementation is shown in Figure 2. We consider an OpenFlow based domain, in which ICN border nodes act as "Inter Working Elements" (IWE) with external domains. Such nodes translate the information contained in the Carrier-Packets header to something that can be processed by the OpenFlow-capable equipment. In particular, this adaptation corresponds to pushing (then popping) a tag to the Carrier-Packets

header. In the scenario depicted in Figure 2, there are no ICN Clients and Servers directly connected to the OpenFlow domain. In this case, the ICN "ingress/egress" nodes act as relay for the ICN Clients/Servers and execute the IWE functions on the ICN packets before forwarding them. More in general, it is also possible that ICN Clients/Servers integrate the IWE functions and are directly connected to the OF-capable nodes in the OpenFlow domain. The functionality of an ICN node is split between an OpenFlow capable switch and an external Cache Server paired with the switch, which act as ICN in-network cache. The NRS nodes, realized as set of SDN Controllers, instruct the forwarding nodes in order to realize the forwarding-by-name and the in-network caching operation.

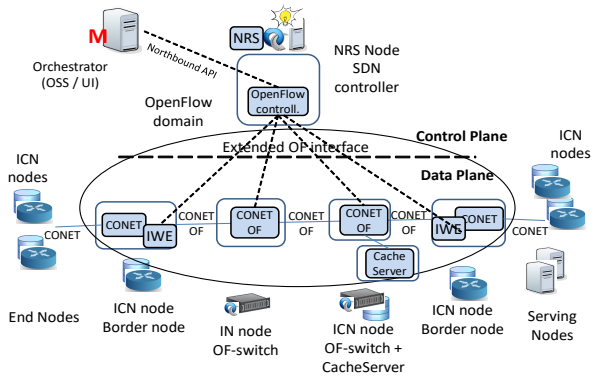

Figure 2 - ICN over SDN testbed scenario

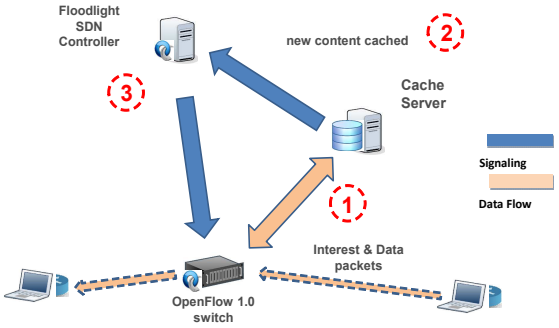

Figure 3 - ICNoSDN PoC operations (high level view)

The high-level view of the ICNoSDN PoC operations is shown in Figure 3. The ICN Clients (border nodes) send the *interest* requests that are initially served by ICN Servers, which send back the *data* towards the ICN clients. While the *data* are forwarded on the reverse path, the SDN controller can instruct the OF switches to forward a copy of the *data* towards Cache Servers distributed in the network (Figure 3, interface 1). Once a Cache Server completes a "chunk" of data, it informs the Controller about the possibility to serve new contents with its cache (Figure 3, interface 2). At this point the Controller proactively "pushes" rules in the OF switches (Figure 3, interface 3), instructing them to redirect further requests of these contents towards the Cache Server, instead of the ICN Servers. We call this behavior as *Tag Based Forwarding* (TBF), as it is based on the tags inserted by the border node of the OpenFlow domain. The forwarding of regular (e.g. non-ICN) IP traffic is supported; we distinguish the regular IP traffic from the ICN traffic using a disjoint set of addresses, so the Controller and the Switches can recognize them. Clearly, this is meant to work in a single-provider scenario, in which the provider can assign a subset of the private IP address space to support the ICN services. From the point of view of the regular IP traffic, the OpenFlow domain is seen as a layer 2 network. It is a specific design goal of the ICNoSDN testbed to support in parallel the operation of the ICNoSDN and of the regular IP traffic, in order to show the flexibility of the SDN based approach.

In the next section, we present in details the operations and forwarding mechanism of the first version of our Proof of Concept, which were not described in [9]. This is also needed to understand the advances of the new version 2, described in Section 5.

4. Proof of Concept version 1 operation details

A detailed view of the ICNoSDN v1 PoC operations is given in Figure 4 (where the logical steps are numbered from A0 to A13). The ICNoSDN v1 PoC uses a static approach to configure the Controller with the information about the experimental topology. For each experimental topology, we need to prepare two configuration files. The first configuration file includes the list of IP addresses of all CONET hosts in the network, separated in ICN Clients and Servers (A0 – Client/Server Config). If a host needs to play both roles, it is assigned two IP addresses. The second configuration file lists the DPID and MAC address of all the switches equipped with a cache (A0 – Cache Server Config) and the port to which a Cache Server is connected. The MAC address of the switch is needed because the Data packets redirected towards the Cache Server cannot keep the ICN server MAC address. This configuration procedure is error prone and does not facilitate experiments with different arbitrary topologies.

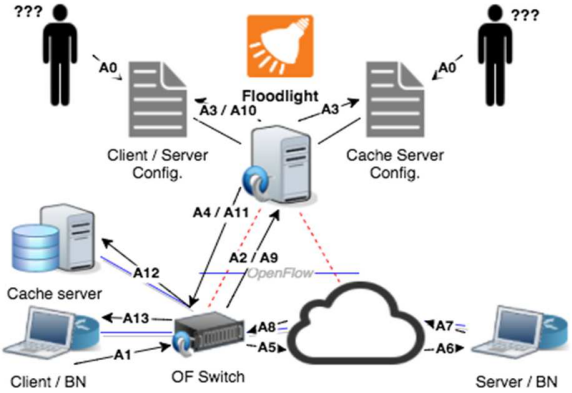

Figure 4 –ICNoSDN v1 PoC: detailed operations

When the client transmits an *interest* request for the first time (A1), the OF switch forwards the *interest* packet to the Controller (A2) through an OF Packet-In message. Taking into account the information contained in the configuration file, the Controller is able to identify that the request is coming from an ICN client and to retrieve the information needed to setup the forwarding rules for *data* packets the switch (A3). Then it "pushes" the forwarding rules for the Client and the duplication rule towards the Cache Server in the proper switches (A4) (rules 1.1 and 1.2 in Figure 5). The request is forwarded in the network (A5) and it can be received by the server (A6). When the ICN server sends the response (*data* packet) in the network (A7-8), the first packet is forwarded to the OF controller (A9). Leveraging on the information contained in the configuration file, the Controller is able to properly identify the Server (A10) and install the forwarding rule for it in the switches (A11) (rule 2.1 in Figure 5). Finally, the response packet is forwarded both to the Cache Server (A12) and to the ICN Client (A13). Finally, once the Controller is informed by the Cache Server about the completion of a chunk, it "pushes" the rule 3.1 of Figure 5 to the switch. Another request for the same content will be served by the Cache Server.

// **(1) Deliver of Data packets**
// For each (client, server) couple, on the switches that have received the Interest packets and are connected to a Cache Server

IF IP Proto is CONET **&&** IP/MAC destination is an ICN Cli
THEN IF IP Src is an ICN Cache Server
(1.1) Forward the packet only towards the destination with Priority 200
OTHERWISE IF IP Src is an ICN Server
(1.2) Forward the packet both the destination and the associated Cache Server with Priority 200

// **(2) Delivery of Interest packets**
// For each (client, server) couple, on the switches that have received a packet from a server

IF IP Proto is CONET **&&** IP/MAC destination is an ICN Ser
THEN IF IP Src is an ICN Client
(2.1) Forward the packet on the port towards the ICN Server with priority 201

// **(3) Delivery of the interests packets to the cache server**
// For each (client, server, tag) triple, on the switches connected to a Cache Server that has the content stored

IF IP Proto is CONET **&&** IP/MAC destination is an ICN Ser
THEN IF IP Src is an ICN Cli **&&** the packet contains **X** tag
(3.1) Forward the packet on the port toward the Cache Server with priority 350

Figure 5 - TBF forwarding Rules

For the forwarding of regular IP traffic, the classical MAC learning approach is used (with a procedure driven by the controller, implemented in the *Learning Switch* of Floodlight). If the MAC destination address is unknown, the packet is sent on all ports but the receiving one. For the ICN traffic, the Learning Switch module is extended to support the installation of the rules described in Figure 5. This approach inherits the limitation of the *Learning Switch* forwarding logic and it is able to work only in loop-free topologies. The scalability of this solution is very poor. In fact, let C, S, T be respectively the number of ICN clients, ICN servers and different content chunks (Tags). The number of rules of type (1) and (2) in Figure 5 is proportional to C·S, while the number of rules of type (3) is proportional to C·S·T.
In order to transport CONET Carrier Packets inside an OpenFlow domain, we use the solution #1 in Figure 8 in which the ICN ID is carried inside the IP Option field and the TAG is carried using the source and destination ports of the UDP header.

5. Proof of Concept version 2

The second version of the PoC, referred to as ICNoSDN v2 PoC aims at realizing an open source framework for the testing of different caching policies inside an ICN over SDN network. It can run on arbitrary topologies, therefore it introduces an automatic topology handling mechanism (section 5.1) and a new more general flow forwarding system (section 5.2), capable to deal with mesh topologies. Dealing with larger topologies in the experimental OFELIA testbed also meant dealing with different types of network equipment; thus we had the chance to test and improve the compatibility of the packet format on different switches (section 5.3). We introduced a modular and extendible controller software architecture (section 5.4), a key element to support different caching policies. In particular, a set of implemented caching policies are described in section 6.

5.1 Automatic topology handling

In the ICNoSDN v2 we drastically simplify the configuration procedure of the SDN controller (see Figure 6, in which the steps are numbered from B0 to B14). Rather than listing all the single IP addresses of the node, we reduce the configuration information to four ranges of IP addresses, respectively identifying: 1) the set of ICN Clients; 2) the set of ICN Servers; 3) the set Cache Servers; 4) the Local Region range (used for TBFF policy, see section 6.1). Using this information, the Controller can identify the different types of entities when receiving "packet-In" messages as needed to react appropriately. Thanks to the use of IP address ranges, we do not need specific forwarding rules for each (client, server) couple as in Figure 5. A single rule can cover all possible clients and server addresses. The scalability of the system is much improved: the number of rules of type (1) and (2) in Figure 5 is respectively proportional to the number of servers S and of client C (like in the regular MAC learning approach). The number of rules of type (3) is proportional to the number of different tags T, while in the ICNoSDN v1 it was proportional to C·S·T. In order to further simply the configuration procedure, we added some automatic discovery procedures for gathering information related to the Cache Servers. With this approach, the same static configuration information is reused for different arbitrary topologies (in the limit of the IPv4 address ranges that have been specified for the different node types) achieving scalability from the management point of view. While we used IPv4 in our testbed, this idea of associating a "semantic" to ranges of IP addresses can be actually implemented in real networks with IPv6, thanks to the huge available address space. In facts, ICN solutions have been proposed that even map the content name in the IPv6 address, see for example hICN [69] (while in our proposed solution we only need to map the class of node in the IPv4 address).
The automatic discovery procedure collects the addresses of Cache Servers and of the switches to which they are connected and will be referred to as *CS scouting procedure*. It is needed because the OpenFlow rules, used to forward traffic towards Cache Servers, have to be installed in proactive way in the switches, before that the Cache servers send any data. The CS scouting procedure avoids the needs of a static configuration file. At their start up and whenever the TCP connection with the Controller goes down, the Cache Servers send a *Scouting Packet* in the data plane (B1). This packet arrives at the first OpenFlow Switch, i.e. the attachment point of the cache server, and then goes to the Controller through a Packet-In message (B2). The Controller receives the Packet-In message and if it is a *Scouting Packet* stores the configuration info for the Cache Server, i.e. the connected switch (the one that sends the Packet-In), input port, the MAC address of the OpenFlow switch and Cache Server addresses (IP and MAC). After performing the scouting procedure, the Cache Server establishes a TCP connection with the Controller. At this point, the operations follow the same procedure of the v1 PoC. The Cache Server operations in ICNoSDN v2 PoC also include a *keep alive* procedure, based on sending periodic *hello messages* towards the Controller. After the Cache Server has established the TCP connection to the Controller, it sends periodic *hello messages* on the connection. This procedure allows detecting any communication problem. If the connection goes down, the Cache Server enters in emergency mode and re-activates the periodic Scouting Procedure until a new pairing can be realized with the Controller.
The procedures to configure the ICN Clients and Servers in the Controller are simpler because we can install the forwarding rules in reactive way and not in a proactive way as needed for the Cache Servers. In fact, the Controller relies on the traffic generated from

the ICN Clients and Servers in order to infer their attachment ports, the IP and the MAC addresses that are needed to install the proper rules for the forwarding of the *data* packets. The first time that a switch sees a packet from an ICN Client or Server, it sends a Packet-In towards the Controller. Then, the Controller with a comparison between the IP address and the IP address ranges stored in the configuration file checks if the host is an ICN Client or Server, finally it does the necessary processing for the packet and "learns" the information needed by our module.

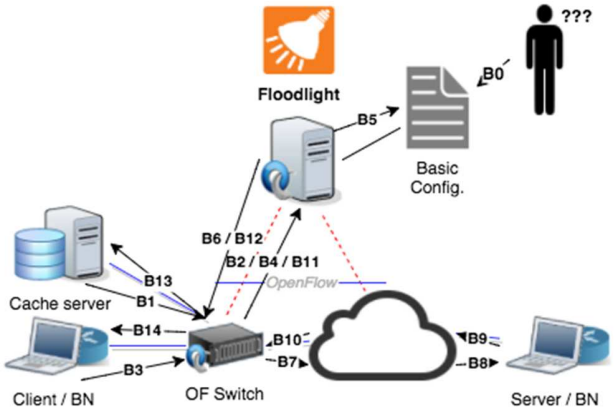

Figure 6 – Automatic handling of node types in ICNoSDN v2 PoC

5.2 Forwarding mechanism based on reverse path forwarding (RPF) check

In traditional layer-2 switched networks (i.e. not based on OpenFlow), topologies with redundant links and loops can be used thanks to the Spanning Tree protocol (SPT) [17]. Using SPT, the switches build the spanning tree that connects all the switches, disabling the ports that are not necessary and can create loops. The SPT approach is not efficient in OpenFlow based networks, where it is preferable to keep all links enabled. In fact, the SDN controller can allocate the flows to the links. The most efficient solution is to keep all links active and let the controller choose which links to use. For example, the Floodlight controller activates by default the *Forwarding* module that evaluates the Shortest Path towards a destination using Dijkstra's algorithm (no automatic learning, the Controller uses the knowledge of the whole topology). The routing of broadcast packet is also decided centrally by building a tree for broadcasts.

The v1 PoC implements a straightforward extension of the *MAC Learning Switch* forwarding logic and it is only able to support loop-free topologies. One property of the MAC learning approach interesting for our purpose is that response packets always follow back the path of the request packets. In our case this is needed, as in our ICN scenario, the *data* packets must follow the same path of the *interest* packets. If this does not happen, the in-network caching mechanism can become much less effective: a content chunk can be cached in a Cache Server during the forwarding of the *data*, but if the *interest* packets follow a different route it will never be requested to the Cache Server. Therefore, we are interested in designing a solution with symmetric routing as provided by MAC learning. On the other hand, we want to support arbitrary mesh topologies and leave all the links active so the SDN controller can choose all links.

To support arbitrary topologies with redundant links and loops, we have extended the *Learning Switch* solution implementing the RPF (Reverse Path Forwarding). The RPF check is a simple solution used for example in multicast protocol like PIM – DM [18], while a variant of this algorithm is used in DVMRP [19]. It consists in a check done on all incoming packets that needs to be broadcast (including the unicast packets towards a destination that is not in the forwarding table). The RPF check discards all packets that the switches receive on ports that are not on the shortest path towards the source (when there are multiple shortest paths, one of them is arbitrarily chosen). Our solution is simpler than the algorithms realized in PIM-DM and DVMRP, as we do not have any "pruning" or "re-grafting" messages. We virtually build a spanning tree combining the Learning Switch approach with the RPF check. When a packet is received on a given port p_x and p_x is on the shortest path towards the source, the controller understands that the source is reachable through p_x and installs the appropriate rules. Subsequently, when the switch receives a packet for the source, it uses p_x to forward the packet. Moreover, one must consider that this choice in a scenario with asymmetric links represents the best solution, when compared with a solution that forwards the packet using a direct (no reverse) spanning tree. In fact using a direct tree the switch could learn that the source is reachable through a port that is not on the shortest path from its point of view.

Figure 7 shows the RPF check in action on a topology that presents some loops. In particular, we show an example of a controlled broadcast communication started from the red node. The "SPT links" (continuous line) identify the branches of the spanning tree. The dashed lines represent the links that are not part of the Spanning Tress. Similarly, the arrows with the continuous lines show the packets that are accepted as broadcast packets and copied on all interfaces (except the incoming one). The arrows with the dashed line show the broadcast packets that are discarded by the RPF check.

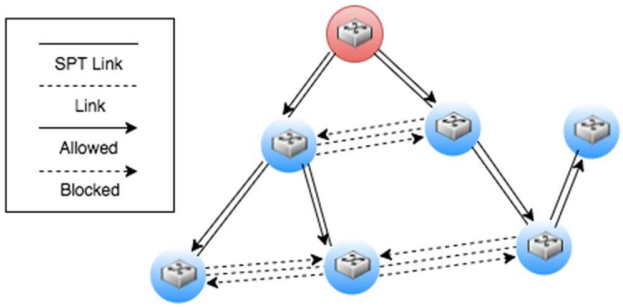

Figure 7 – RPF check in action

The RPF check has been implemented in the Control Plane as a simple check performed on the Packet-In, the Controller discards all the packets that the switches receive on ports that are not on the shortest path towards the source.

The combination of Learning Switch and RPF check represents a unified solution used for both ICN traffic and regular IP, but the ICN packets (recognized by their IP source and destination addressed) are further processed in order to setup the rules for the duplication of *data* towards the Cache Server.

5.3 Encoding of CONET packets

We designed two different solutions to transport CONET Carrier Packets inside an OpenFlow domain (Figure 8). In the first solution (#1 in Figure 8) the ICN ID is carried inside the IP Option field, the TAG is carried using the source and destination ports of the UDP header and the CONET payload is carried inside the remaining part of the UDP header (length and checksum) and inside the UDP payload. In the second solution (#2 in Figure 8) the TAG is carried in the UDP header, as in the first solution, but the ICN ID has been shifted from the IP header to the UDP payload. Both solutions leverage the fields of existing protocols in order to

carry CONET information, as needed to be processed by legacy OpenFlow switches. In both cases, the TAG carries the ICN information (a function of the ICN name) in a section of the packet that can be matched with the OpenFlow rules. The IP header Protocol field is set to UDP so the packets are seen by network elements (including OF switches as UDP packets). The ICN traffic can be distinguished from "regular" UDP traffic by the IP address. In the simple experiment reported in [9] we implemented and used the first solution based on the IP option. We have then extended the experiments on the OFELIA testbed deploying larger topologies across multiple OFELIA islands, as will be detailed in section 7.1. In these *multi-island* experiments, we encountered issues, as some ICN Clients systematically could not establish any communication with an ICN Server. This happened when the ICN Server was located behind an Open vSwitch instead of a hardware switch. We discovered that the switch had problems in forwarding the packets with the IP Option. Therefore, we implemented and used the second solution (#2 in Figure 8), avoiding potential issues with IP Options. It has to consider that this fix was also a key enabler for the deployment and the experimentations over Mininet environment.

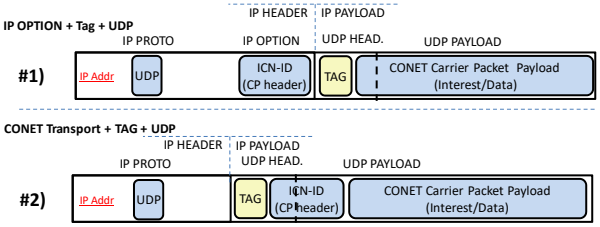

Figure 8 - CONET packet over SDN

5.4 Software Architecture of SDN Controller modules

The SDN Controller implementation is based on Floodlight 0.9 [23]. The v1 PoC described in [9] is based on a rough implementation, with the introduction of a new monolithic Floodlight module, managing all the aspects of the ICNoSDN control plane. The software implementation for v2 PoC improves the modularity of the control plane, easing the development of new application logics and caching policies. Figure 9 shows the new software architecture. The ConetModule class acts as an abstraction layer between the switches and the other parts of the module, and offers methods that permit to easily push *tag based rules* on the OpenFlow equipment (i.e. using the Southbound API). The ConetListener class manages the incoming packets from the switches. If they belong to ICN traffic, the ConetListener dispatches them to the appropriate Handler. In the class hierarchy of the Handler we have encapsulated the application logic and the caching policies. The Handler super class provides the simple Layer 2 Forwarding of the packets, while the caching policies are implemented in the subclasses, for example the MultiCSHandler realizes the Tag Based Forwarding. This architecture is directly inspired from the Netfilter framework [24], new improvements can be introduced easily and their impact on the remaining code is minimized. As regards the northbound API, we introduce a set of classes that extend the REST API of the Floodlight Controller and provide many interesting functionalities, the most important are: i) activation of a specific policy; ii) customization of the policies; iii) accounting of cached items. These classes have not been reported in Figure 9 in order to provide a more clear representation of the core architecture of the ICNoSDN v2 implementation.

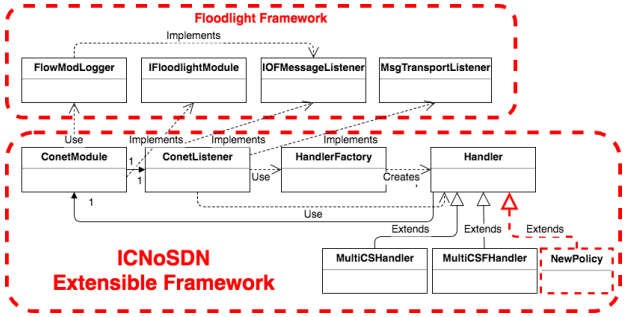

Figure 9 - Software architecture of the SDN Controller

6. Caching policies

The Tag Based Forwarding (TBF) mechanism is the basic caching policy implemented in the ICNoSDN architecture. It simply tries to cache every content that crosses an ICN capable node. The limitation of the storage capacity and of the other network resources (e.g. size of the OF switch tables) is not taken into account. More complex caching policies are needed to maximize the cache effectiveness taking into account the resource constrains. The ICNoSDN v2 PoC offers an environment to deploy and test different solutions. We have designed and implemented four caching policies described hereafter, that are available in the PoC distribution. The policies have been implemented by changing the logic in the SDN controller, i.e. providing additional extensions to the Handler class (see Figure 9). Note that the same policies could be implemented without SDN, by putting the control logic in the ICN nodes. The advantage of using the SDN infrastructure is the flexibility and the facility of implementing new policies operating in the logically centralized SDN control infrastructure.

6.1 TBFF

TBFF stands for TBF-Filter. It is based on "filtering", i.e. not caching a subset of the contents. The idea is that from the perspective of an ICN node, the contents can be divided in *local* and *remote* ones. This requires the partitioning of ICN Servers and ICN nodes in different *regions*. A content that is originated from an ICN Server in the same region of the node is *local*; a content that is originated from an ICN Server in a different region is *remote*. Local contents are not cached, so that cache resources are saved for remote contents. When caching resources are limited, TBFF ensures that they are used to reduce the inter-region traffic. Caching of local content is less effective, because they cross a smaller numbers of links, and the benefit of being served from the Cache Server are smaller than in the case of content coming from a remote Server and crossing the inter-region links. In TBFF, the Controller only installs rules that duplicate contents coming from remote servers, while the local contents are simply forwarded towards the clients. This behavior is realized configuring (with the Controller configuration file, or using the appropriate Rest API) the *local region range*, i.e. the subset of the IP addresses that belongs to the same region of a particular switch. The algorithms for the partitioning of a domain into regions are beyond the scope of this works. In our experiments on the ICNoSDN v2 PoC we have manually partitioned the network.

In Figure 11, we show the TBFF mechanism in action. The arrows with label XA (X=1..3) is a number) represent the flow of contents (*data* packets) from a local server, while those with label XB (X=1..5) represent the flow of contents from a remote server. *Data* packets coming from the remote server are duplicated towards the

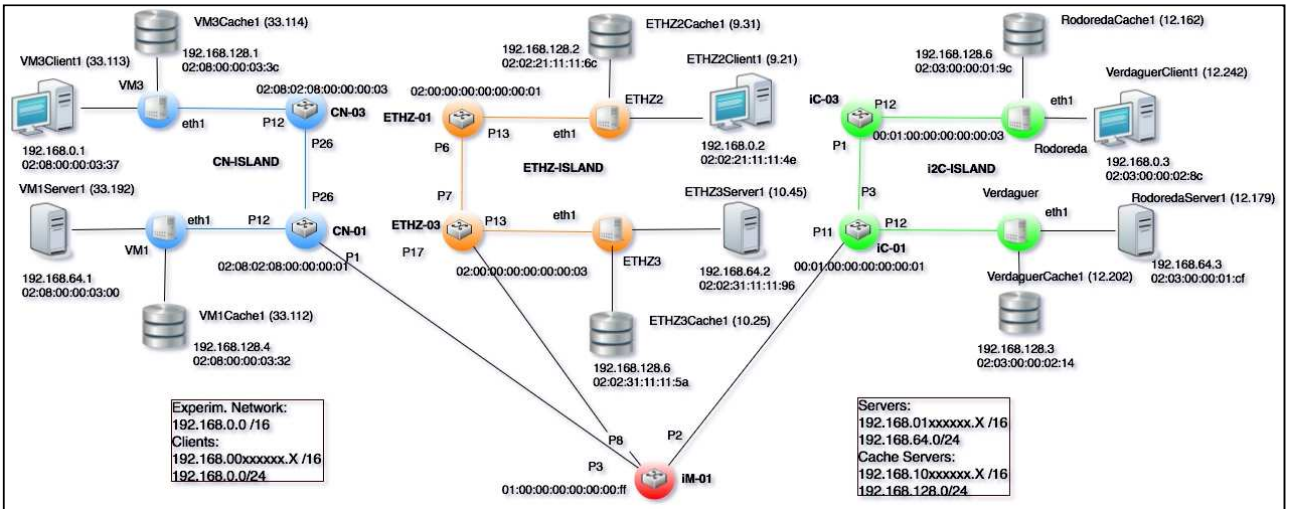

Figure 10 - OFELIA testbed scenario

Cache Server by the OF switch, while *data* packets coming from the local server are only sent towards the Client. Section 8 reports some experimental results of the TBFF implementation.

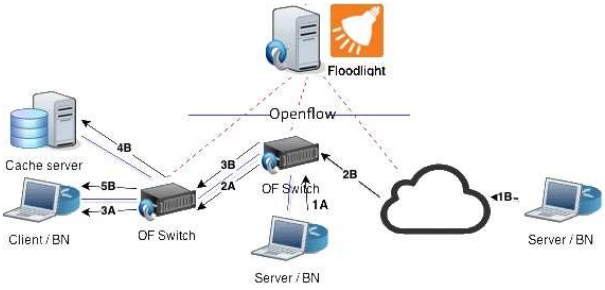

Figure 11 - TBFF in action

6.2 DTBF and DTBFF

DTBF and DTBFF respectively stands for Dynamic TBF and Dynamic TBFF. These mechanisms introduce a timeout for the rules that replicate the *data* packets towards the Cache Servers. With DTBF and DTBFF, each rule has an associated soft-timeout of duration T_o [s]: if a rule has not been matched for T_o seconds it is deleted (making room for new requests). This policy, taking advantage of what has been realized previously, is implemented with simple modifications that have minimal impact on the work already done. The DTBF and DTBFF Caching Policies have been validated and we show their benefits with dynamic patterns of requests (see section 8). The timeout T_o can be set using the configuration file or by exploiting a specific REST API that we have implemented in order to control this behavior in real time during the experiment.

6.3 FIX(p)

FIX(p) is a caching policy proposed and discussed in [20][21]. The idea is that each chunk of the content received by an ICN node has a probability p of getting indexed and cached. It is simple to be implemented as it does not need the cooperation among the in-network caches. On the other hand, as it is stated in [22] an explicit cache coordination policy would likely violate the CCN line speed constraint. In the original implementation ([20][21]) the receiving node implements the policy and decides whether to cache a chunk or not. We have implemented FIX(p) leveraging the SDN Control Plane. In particular we have added the FIX(p) logic in the Controller, when it processes the messages coming from the Cache Server. On receipt of a *cached* or *refreshed* message the Controller activates the FIX(p) logic: it chooses with probability p whether to consider the message and configure the rules in the flow table (according to the active caching policy) or discard the message. In our context this simple caching policy can be combined with the basic TBF mechanism or joined with other caching policies (TBFF, DTBF…). In the latter case, FIX(p) acts as an implicit cache coordination policy. For example if we have M Cache Servers inside a local TBFF domain, all the M Cache Servers in the request path will cache the requested content. The total number of objects that can be cached in the local domain corresponds to the capacity of the largest Cache Server in the domain. If all Cache Servers can contain N objects, it can happen that the total number of different objects is N, while the maximum aggregated capacity is M x N. The combination of FIX(p) and TBFF mitigates this issue improving the total number of different cached object in the local TBFF domain.

6.4 Additional caching policies

The proposed framework can be leveraged to realize additional caching policies, based on the existing ones or based on different mechanisms. For example, the popularity of the contents is not included as an ingredient in the implemented policies. A module that estimates the popularity of the contents should be implemented, and then this information could be used by the algorithms that decide which content is to be cached in a given Cache Server. The SDN based approach may allow the algorithms to be based on a centralized view of the system, improving their effectiveness.

7. Emulation tools

The ICNoSDN v2 PoC can be run over different testbeds, maintaining the same Execution Enviroment. In Figure 12, we can see the building blocks that make up the tools needed to run an experiment using our framework. An experimenter that want to use our framework should only choose the suitable deployment for the desired type of testbed, he/she will be able to execute the experiment using the same Monitoring GUI and the same Execution Environment. The ability to perform experiments using different testbeds is a considerable extension to the v1 PoC. In particular, as the access to the OFELIA distributed testbed could be difficult, it is valuable to have the option to use the local emulation approach provided by Mininet, lowering the “entry

barrier" to perform meaningful experiments. It is possible to execute a given experiment first on a local setup and then on a distributed testbed with minimal changes, saving a lot of configuration effort. Hereafter, we report the details of the deployment on the different testbeds and of the monitoring GUI.

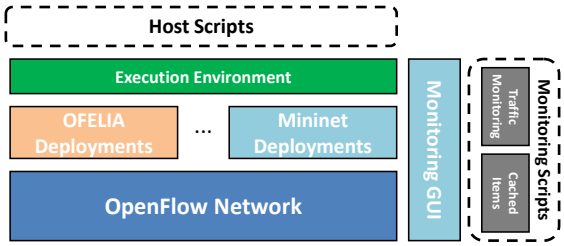

Figure 12 - Emulation building blocks

7.1 ICNoSDN deployment on OFELIA

OFELIA is a European research project that created a large distributed testbed for OpenFlow based Software Defined Networks [25]. The OFELIA testbed provides network virtualization and allows the experimenters to operate in parallel on the same physical resources. The OFELIA infrastructure consists of OpenFlow enabled islands. These islands create a federation all over Europe that allows the experimenters to access virtual machines and their interconnecting switches through the "OFELIA Control Framework" (OCF). The OCF is derived from the Expedient tool developed at Stanford University in the eGENI project [26]. Using the OCF an experimenter can create its own "slice" i.e. virtual topology over the set of physical OpenFlow switches, can instantiate its own OpenFlow controller(s) and a set of "user" Virtual Machines that may play the role of traffic sources/sinks. In our previous work [9] the experiments were realized on small OFELIA slice composed of three VMs and two OpenFlow switches, in a *single island* experiment. In the phase 2 of the OFELIA project the island interconnection has been introduced. This allows to increase the scale of the experiments, combining resources offered by different islands, and to test scenarios with higher latencies. Deploying and running a *multi island* experiment requires to connect through OpenVPN, via the central hub at IBBT in Ghent. Then using the OCF GUI (similar to the graphical user interfaces provided by other *Infrastructure as a Services* environments) it is possible to create and configure the experimental slice. A slice consists of a number of end points (Xen-based Virtual Machines) and OpenFlow access to a set of switches that connect the end points. This equipment is shared among the other experimenters, using the FlowVisor tool [27]. The experimenters have to choose the links between the end points and the switch ports in order to obtain the desired virtual topology (typically containing a subset of physical link). Then the experimenters have to deploy the SDN controller in one of the created VMs, as well as to deploy the needed software in the other VMs. The available facilities are completely independent of the software used, the only requirements is the compatibility with the OpenFlow 1.0.

The OFELIA deployment can be defined as semi-automatized; in fact, some steps in the deployment process cannot be fully dynamic. The creation of a slice in the OFELIA testbed requires the permission of the island administrator; this step must be performed manually. An important missing feature is the possibility to save/copy/restore the VMs images. This would enable the user to reduce the effort needed to setup new slices and new experiments in a considerable way. VMs could be even shared between experimenters, to replicate / enhance experiments performed by other researchers. In the current state, only "empty" VMs can be instantiated and the experimenters need to perform all the VM setup/configuration steps. In order to improve the deployment of ICNoSDN framework on the OFELIA testbed the v2 PoC introduces the automatic topology handling (see section 5.1) and provides a set of scripts that help in the setup/execution/analysis of the experiment. The topology in Figure 10 describes the v2 PoC deployment, with four federated islands: Trento, Barcelona, Zurich and Ghent (which provides inter-island connectivity among the other three islands). Users can easily run experiments with topologies containing a subset or all nodes of the topology in Figure 10. Note that to add a new host the user only needs to configure a new VM and provide the hosts setup scripts and configurations files. On the other hand, adding a new switch or link in the experiment topology requires the authorization of the administrators of the involved islands.

7.2 Mininet deployments

The main reasons for supporting local testbeds with an emulation environment are: i) distributed testbeds are accessible only for a limited number of users and their usage can limited in time; ii) an Experimenter do not have full control over the events and it is difficult to trace all the problems that can occur during the experiments executions; iii) the creation of slice in a distributed testbed often requires the admin permission so the experimenter needs to wait for the grant which makes the process slow. Extending the ICNoSDN PoC to support local emulation environment makes the development/deployment process easier and faster. Mininet [11] is one of the best development/testing tool for OpenFlow based SDN. It allows emulating a network composed of OpenFlow capable nodes on a single host. Based on Linux virtual network features (network namespaces and virtual Ethernet pairs) it provides an efficient way to emulate network topologies with reasonable fidelity. Obviously, the experimenter has the full control of what happens on the links and nodes of the emulated topologies, making the test and debugging of services easier. The Mininet tool does not provide a fully virtualized environment for the virtual nodes. On one hand, this allows saving resources of the host running Mininet; on the other hand, this requires some attention in the porting of the ICNoSDN Execution Environment. We had to properly configure our software tools (CONET ICN software and monitoring tools), creating a logically separated execution environment (folders, configurations files and scripts) for every Mininet node. We wrote a python script that extends the Mininet functionality: taking the topology name as an argument, it automatically prepares and configures the local testbed. In the Mininet based emulation environment, the ICNoSDN v2 PoC offers the possibility to experiment different (user defined) topologies. The deployment scripts are topology-independent; an experimenter can input his/her topology. A catalogue of built-in topologies is included in the ICNoSDN v2 PoC. Each of these topologies includes a number of ICN Clients, ICN Servers, Cache Servers, OF capable switches and a Floodlight Controller. The python deployment script also enables to run the SNMP daemon for collecting traffic measurements and the Multi Router Traffic Grapher (MRTG [28]) application to draw the graphs of the measurements. By using the Mininet API, we first spawn the SNMP daemon in all our hosts, and then we properly configure MRTG in order to collect and generate the monitoring data. Another key feature of the Mininet deployment scripts is that they compile automatically all the needed software tools before the execution of the experiment.

7.3 Monitoring GUI

A key element for the execution of our experiment is the Orchestrator node (see Figure 1). This node provides the overall control of the experiment, it offers a GUI to the experimenter, and collects the results obtained from the monitoring of the ICN nodes interfaces. The Orchestrator node gathers information both from the Controller node (using its northbound interface) and directly from the ICN nodes (using the SNMP protocol). The northbound interface of the SDN controller is an extended version of the REST API provided by the Floodlight controller; using this interface the Orchestrator node can select the caching strategy and can receive information about the number of cached objects and the status of the switches flow tables. SNMP agents running on the ICN nodes and Cache servers provide the statistics about the traffic rate measured on the network interfaces. The SNMP manager that collects such statistics runs on the Orchestrator node. The traffic statistics are processed using the MRTG tool [28], which stores them using the RRDtool [29]. The extended northbound interface of the SDN controller exposes the statistics related to the number of cached objects. These statistics are collected by a python script that saves the data using the RRDtool as well. The RRDtool provides also a convenient mechanism to produce graphic representation of all statistics that are presented on the web GUI of the Orchestrator node.

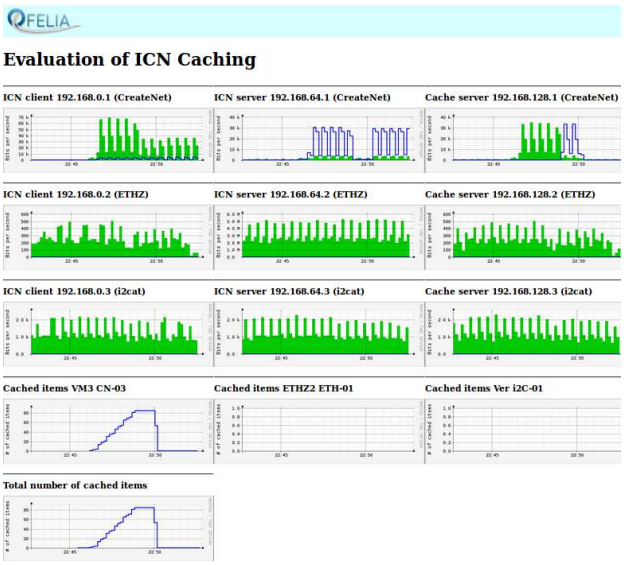

Figure 13 - Performance monitoring using the web GUI

Figure 13 shows an example of the performance monitoring GUI for an experiment on ICN caching. The first three rows show the amount of crossing traffic for ICN Clients, ICN Servers and Cache Servers. These rows represents respectively the Trento, Zurich and Barcelona islands (see Figure 10). In each graph, the blue line shows the outgoing traffic and the green solid line shows the incoming traffic. The graphs in the last two rows show the number of cached items (for each of the three cache servers and then the total number).

7.4 Software framework

In addition to the software modules in the SDN Controller and in the Orchestrator node, the ICNoSDN v2 PoC includes several other components. The ICN software that runs on ICN Servers and clients is a modified version of the CCNx suite [31]. We patched the ccnd daemon using the CONET CCNx patch available in [32]. This patch adds the support for CONET Carrier Packets and transport protocol, leaving untouched the API offered by the CCNx daemon to the others tools of the CCNx suite. An experimenter can directly use the other components of the CCNx suite (e.g. ccncatchunks or ccnrepo), or he/she can configure and use the CCNx software using the v2 PoC host scripts. In our framework, we support both plain OF switches and OF switches paired with Cache Servers. The latter ones act as *in-network* caching nodes that can be controlled using our extended southbound interface. In addition to these tools, we provide also a set of scripts (Host and Measurements scripts) that helps the experimenters during the setup and execution of the experiments. As described in the previous sections we offer two different testbed deployments methods, one for the OFELIA testbed and another one based on a local Mininet emulator. For the OFELIA testbed, a step-by-step tutorial is provided at [12]. For the Mininet based emulation, a "ready-to-go" Virtual Machine with the full ICNoSDN v2 PoC installed is available for download [12].

8. Performance Evaluation

A performance evaluation of key aspects of our framework is reported in this section. We show the potential of the SDN paradigm to implement the caching operation inside an ICN network and we provide a performance analysis of the different caching strategies we implemented in our ICNoSDN v2 PoC. The topology represented in Figure 10 has been used in the experiments. Note that ICN clients can communicate with "local" Cache Servers and ICN Servers (i.e. in the same island) or with "remote" Cache Servers and ICN Servers (i.e. in a different island). This configuration is rather typical; therefore, the experiments results will be of general validity and not restricted to the specific topology. Table 1 reports the definition of the variables governing our tests, which will be used in the following sections.

Notation	Definition
N	Number of contents
N_A	Number of active contents
M	Number of chunks composing a content
S	Size of a content, equal to M * chunk size
N_r	Number of requests (N * M)
MaxItems	Max number of items handled by an OF switch
T_c	Requests cycle [s]
T_o	Expiration time of the flow entries [s]
T_a	K * T_c [s]
C_r	Cached requests
p	Caching probability

Table 1 – Definition of the variables

In each experiment, a synthetic ICN traffic (*interest* and *data* packets) is generated using a set of ICN Client and Server applications. This synthetic ICN traffic is then processed by the ICN/SDN network under test and we are able to collect the measurements of interest using our monitoring tools.

The emulation tools described in section 7 allows running the experiments both on the Ofelia testbed and on a local Mininet deployment. From the experiments run on the Ofelia testbed we were able to verify the functionality of our implemented solution over real hardware switches (see the issues described in section 5.3) and we evaluated the flow size limitations of the hardware switches (see section 8.2). The experiments reported in this section have been performed on the Ofelia testbed, but the ones in section 8.4, which have been performed on Mninnet. Anyway, we checked that the results on both testbeds are the same if we set a limitation on the switch flow table size in the local Mininet experiments corresponding to the one in the hardware switches.

8.1 IP forwarding and static in-network caching

The purpose of this experiment is to verify the functionality of the TBF (Tag Based Forwarding) in-network caching mechanism by measuring the traffic loads on ICN servers and ICN Cache server and comparing then with the case in which no TBF caching is used.

In this experiment, we considered a "static" input traffic pattern. It foresees a continuous repetition of a set of content requests (cycling requests), which is only characterized by the number (N) of contents in the set. A given ICN Client (we recall that an ICN Client in the testbed represents a potentially large set of end users) requests a set of N different content objects, each one composed by M chunks (the size S of each object will be M * chunk size). Note that our basic unit of storage is the chunk. Therefore, each client will periodically generates a number of requests $N_r = N*M$ for different content chunks. We denote as *cycle time* T_c the duration of the period [s] in which the ICN client requests the N_r objects. The chunks of the requested objects are stored in the Cache Server and the following requests for the same contents will be forwarded to it instead of the origin server. We define T_o the expiration time [s] of the flows in the switches. If we assume that T_o is longer than the cycle time T_c, the entries in the switch flow table will not expire in our experiment.

We used as baseline for our evaluation a traditional forwarding (NO-TBF), which does not use in-network caching and only uses MAC learning forwarding with RPF check (as described in section 5.2). In this first evaluation scenario, we simply assume that the cache resources can cache all the set of requested contents and the number of chunks is not greater than the number of flow entries the switches are able to manage. Each client requests contents to a remote origin server with a requests cycle T_c of about 110 seconds, the communication mapping is the following: CLI-CNET to SER-ETHZ, CLI-ETHZ to SER-i2C and CLI-i2C to SER-CNET. If No-TBF is used, the *interest* requests and the *data* keep flowing on the inter-islands links. Figure 14 shows the traffic captured on the interface of SER-ETHZ (top) and on the interface of the cache server CAC-CNET (bottom) near the client CLI-CNET. The ICN Server interface is loaded, while the Cache server is unloaded (note that the y-scale of the graph is dynamically adapted, for the ICN Server (top) it is set to 50kb/s while for Cache Server (bottom) the scale is set to 1 kb/s.

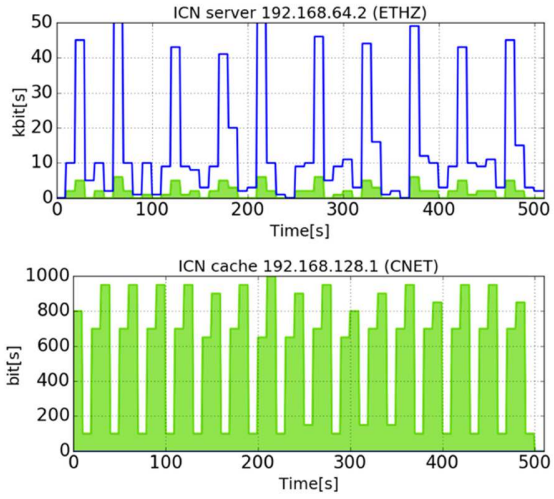

Figure 14 - No-TBF

In Figure 15, we show the effects of introducing the simple strategy TBF (Tag Based Forwarding) strategy for in-network caching. The same T_c of about 110 seconds is used in the clients. Initially, the set of N different contents are requested and provided by the SER-ETHZ. During the data transfer, the packets are copied to the Cache server, that caches the content and notifies the controller (the number of cached items increases). After T_c, the ICN Clients start requesting again the same set of content: the load of the ICN Server and on the inter-island links goes to zero, while the CAC-CNET starts serving requests and the traffic on the Cache Server increases.

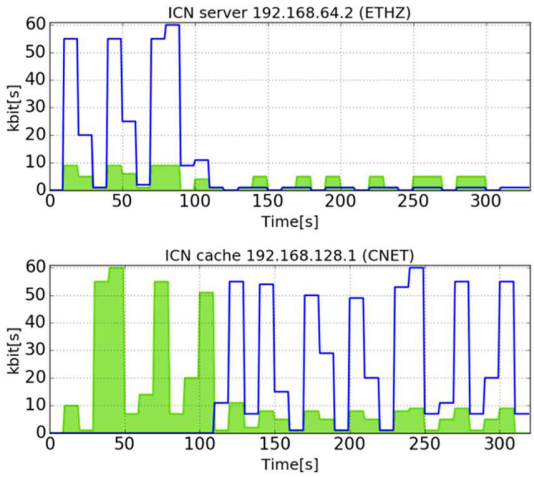

Figure 15 - TBF caching policy

8.2 Policy based in-network caching

In the previous experiment, we have assumed that the resources of the Cache Servers (storage capacity) and of the OF switches (table entries) do not constitute a constraint for our caching strategy. However, this is not a realistic assumption because the resources are limited. In particular, during our experiments on the OFELIA testbed, we have encountered resource limitations in the OF switches: once the entries of the OF tables are exhausted it is not possible to use the proposed caching mechanism for new contents. Therefore, we have analyzed the dimensions of the OF tables of the different switches deployed in our testbeds. We executed experiments where an ICN Client requested a set of big sized contents, each one made of a very large number of chunks (in our solution we need a different flow entry for each chunk). The objective was to saturate the OF tables, to receive the OF_TABLE_FULL error from the switch and to register the maximum number of entries in the table. Table 2 reports our findings: the limits of the available OF tables are 1518 entries. Actually, the number of rules that can be inserted is lower because the ingress ports are "wildcarded", and this is equivalent to insert one rule for each port. The rightmost column of Table 2 reports the maximum number of items that can be redirected to the cache server for the switches used in our testbed (NEC AX-3640-24T2XW-L). Note that in these conditions, the bottleneck will not be the storage capacity in the Cache Server, but the number of items that can be redirected by configuring the OF switch tables.

Switch	Entries	Ports	Max_Items
CN-03	1518	2	755
i2C-01	1518	3	503
ETH-01	1518	2	755

Table 2 - Max number of available flow table entries and items that can be handled

Taking into account the fact that the resources are constrained, in the second experiment we enhance the static in-network caching solution by using the TBFF strategy defined in section 6.1. The TBFF strategy selectively caches content on a Cache server to maximize the gain obtained by caching. This avoids the caching of contents originated from the local server. The purpose of this second experiment is to verify the functionality of the TBFF caching mechanism by measuring the traffic load on the local and remote ICN servers.

We assume that an ICN Client located in CreateNet island is periodically making a set of requests to three different ICN servers, one located in CreateNet island and the other two in the remote islands (i2Cat and ETHZ). We use the same "static" pattern of requests described for the first experiment, but now the number of requested item N_r is more than 1K exceeding Max_Items. We performed a run of the experiment with the static TBF caching policy and we observed that in the steady state the items exceeding Max_Items were equally split among the three ICN servers and that the requests for these items were causing traffic load on the ICN servers (results not shown for space reasons). On the other hand, when using the TBFF policy, the CAC-CNET Cache Server does not cache all the contents coming from the local ICN server SER-CNET. The number of items requested from the remote ICN servers is less than Max_Items, therefore all "remote" items will be cached in the CAC-CNET Cache Server. In Figure 16, the steady state of the experiment with TBFF is reported, after the first cycle (of duration T_c) of the requests has been performed. The ICN client continues to request the same set of contents. The local ICN Server in CreateNet (top left in Figure 16) is still loaded, as the contents that it provides has not been cached. On the other hand, the load on the remote ICN Servers in ETHZ and i2Cat island has been reduced to zero (the two bottom diagrams in Figure 16, in which there is only 1-2 kb/s background traffic, note again the different scale of the graphs). We verified that the number of cached items C_r in the cache is around 700, below the limit imposed by the maximum number of entries in the OpenFlow switch flow table, showing the effectiveness of the TBFF strategy.

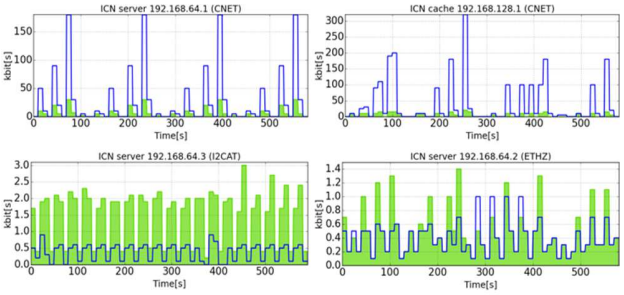

Figure 16 - Advanced TBFF caching policy

8.3 Dynamic in-network caching

D-TBF and D-TBFF (defined in section 6.2) introduce a dynamic entry expiration mechanism that makes room to new entries when the old ones are not used. The purpose of the third experiment is to validate the dynamic mechanism and show how it can accommodate a set of different requests larger than the capacity of the OF tables.

We defined a "dynamic" request pattern in which the "active set" of content requests changes over time. N_A is the number of different contents in the active set. The client periodically (with cycle time T_c) requests a set of N_a contents. At a given time $T_a = K * Tc$ (i.e. after K cycles of requests) it will start requesting a different set of N_a contents. such that total number of different requests N_r > Max_items. In the experiment, the table entries resources are not enough to deal with all the N_r contents, but are capable to deal with the contents in the "active" set of requests (N_a). Therefore, we need to make room to new contents and remove the entries related to old unused contents. Using D-TBF or D-TBFF, after the time-out time T_o, the entries for the first set of content will expire, making room for the new content requests.

The ICN Client in CreateNet island starts performing a set of requests towards the ICN Server in ETHZ island (at time t = 0) for Nr <= Max_items, these requests are performed periodically with a period T_c of about 70 seconds. Figure 17 reports the results of this third experiment. The requests are initially served by the remote ICN Server (SER-ETHZ); meanwhile they are cached by the local Cache Server as it can be seen from the increasing number of cached items. The ICN Client repeats the same set of requests for K=3, so after the first T_c (first and third red vertical red lines) the requests are served by the Cache Server and the ICN Server load goes to zero. Then at time T_a (second and fourth vertical red lines), the ICN client starts requesting a different set of requests, i.e. the active set of requests is changing. The new contents are requested to the remote ICN Server SER-ETHZ, as they are not in the CAC-CNET Cache Server. The Cache Server starts caching the new contents and the cached item number increases. After a short while, the entries related to the old contents start to expire, reducing the number of cached items. After a transient phase, the number of cached items is the same as before (because in our request pattern the second active set has an identical size to the first active set). The requests are now forwarded to the local Cache Server and the load on the remote ICN Server goes again to zero as desired.

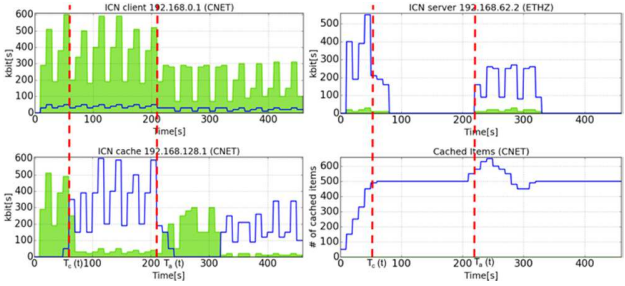

Figure 17 - D-TBF caching policies

8.4 Validation of the FIX(p) strategy

The FIX(p) mechanism (section 6.3) introduces a probability in the decision whether to cache or not a content. This policy provides a simple "implicit" cache coordination mechanism, which actually does not require any explicit cooperation among Cache Server. One of the most evident advantage is the improved cache diversity in the network; other benefits of the cache coordination have been widely discussed in [66]. The goal of the experiment presented in this section is to validate the FIX(p) policy.

FIX(p) can be combined with the basic TBF caching mechanisms or with other caching policies implemented in the ICNoSDN v2 PoC. In the experiment described hereafter, it is combined with the basic TBF mechanism. For this experiment, we used a subset of the topology depicted in Figure 10, only including the i2C and ETHZ islands. The ICN Client1 (i2C island) requests contents uploaded on ICN Server2 (ETHZ island).

We execute the experiment with four different values of the caching probability p in FIX(p) {1, 0.8, 0.6, 0.4}, ranging from 1 to 0.4 in steps of 0.2. When p is equal to 1 it corresponds exactly to the basic TBF case (this means that each content will be always

cached). Figure 18 reports the number of cached items in the iC3 Cache Server as function of the time. The number of cached items grows from zero and continuously increases until it reaches the maximum threshold (in this case 380 items, which is the number of different chunks requested in the experiment). When we repeat the experiment decreasing the caching probability, showing the influence of the probability on the caching strategy: the grow rate of the cached items decreases proportionally. With p=1, we are able to cache the maximum number of cached items in 5 minutes. Looking at Figure 18, we observe that after 5 minutes the fraction of the items that has been cached directly depends on p. For example, comparing p=0.8 and p= 0.4, we can clearly show that after 5 minutes the number of the items for p=0.4 is exactly the half of the items for p=0.8.

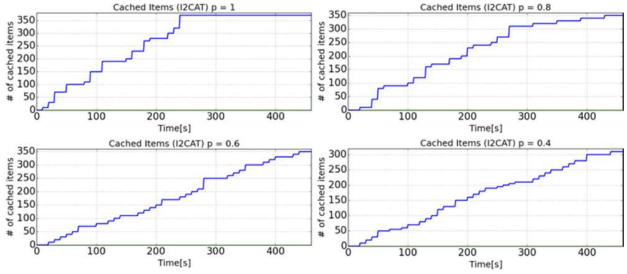

Figure 18 – FIX(p) cached items vs time for different p

8.5 Discussion on the scalability limitations

The most important outcome of the tests on the OFELIA testbed is that the scalability of the solution is limited by the number of entries in the flow table. This number is very low in the specific testbed environment (see Table 2) for a number of reasons: i) the switch used in the testbed are the first generation OpenFlow switches; ii) only a fraction of the flow table of a physical switch is assigned to a testbed user; iii) the virtualization mechanism transforms a wildcard rule "all ports" in a set of rules, one for each port. In a modern and not virtualized switch, we can expect a number of entries in the order of tens of thousands.

Nevertheless, there can be scenarios in which we want to support hundreds of thousands of different active chunks in a domain, cached in a specific Cache Server. In such cases, we may need to enhance the scalability of the proposed Tag based solutions (assuming that the Cache server storage space will not become a bottleneck). A first solution could be to consider chunks with variable size. In fact, if the size of chunk is kept constant, very large files will generate a very large number of chunks; therefore a large number of different forwarding rules will be needed. An improvement could be to have a chuck size that increases if the file is longer. Another mitigation to this problem is a proper use of the caching policies, which can reduce the average number of chunks cached in Cache servers and consequently the number of Tag based forwarding rules. In this respect, the Tag semantic could be extended to transport additional information useful to assist the decision of the Caching policies.

9. Related work

The basic ideas about the ICN paradigm have been proposed in [1] [2], which in turn find their root in [30]. Several recent projects have been dealing with ICN research ([15][31][33][34][35][36], to list a few). Surveys on ICN concepts and research issues can be found in [37] and [38]. Our previous work on ICN namely on CONET framework can be found in [13] and [14]. As regards SDN, the seminal paper on the paradigm can be found at [8], surveys on SDN research issues can be found in [5][6] and [7].

ICN and SDN have evolved separately; a relatively recent research trend is considering their combination. Our early work related and supporting ICN functionalities in SDN can be found in [39][40]. Hereafter we provide an overview on research related to the combination of ICN and SDN.

The proposed architecture in [43] shares some common points with our approach especially in the operations performed at the edge of the ICN domain. The edge node acts as the domain Controller and is responsible to achieve internal or edge delivery.

The recent work in [44] proposes an ICN architecture based on SDN concept, aligned with the approach we have proposed in [9].

The authors in [42] exploit the SDN concept for their specific ICN solution [33][41], which is rather different from our ICN approach [13] based on the CCN/NDN architectures [2][15] .

The architecture proposed in [45][46] focuses on the extension of SDN/OpenFlow, particularly in storage and content primitives. Content routing is supported by mapping content to a TCP flow and using an SDN approach to route the flow. The mapping is performed in the edge proxies, which map HTTP requests into flows. The main difference with our approach proposed is that the clients and server just use legacy HTTP without using any ICN protocol, as we do.

In [47], the "tagging" approach is proposed for routing within a data center. The approach of having a centralized routing engine per autonomous system has also been considered for regular IP routing. For example, [48] first proposed the so-called Routing Control Platforms, while the BGP "Route reflector" solution [49] can be seen as a form of centralization of RIBs. Recently, solutions to centralize the RIB avoiding to run BGP at the edges of an autonomous system and exploiting a SDN backbone have been proposed in [50] and [51]. As for the inter-domain name-based routing, the use of a BGP-like approach was already proposed in [52]. Recently, it has been proposed to extend BGP to disseminate advertisement information about content objects in the form of URIs [53].

In [54], the authors provides a detailed view and discuss about the availability of OpenFlow based technologies. Moreover, they present research initiatives regarding the use of SDN for future internet research and experimentation.

In [55], the authors provide an experimental environment based on virtual networks called Future Internet Testbed with Security (FITS). FITS allows the creation of multiple virtual networks in parallel, based on virtualization tools Xen and OpenFlow. The experimenting platform divides physical network in virtual networks, each containing its own protocol stack, routing rules and management. They used FITS to experiment Content-Centric Network and compare the CCNx with the conventional TCP/IP protocol stack.

In [56], the authors present a survey on the existing ICN software tools, with a cross comparison between ICN simulators.

In [57], the authors present a survey on caching approach in ICN, identifying two different types of caching (In-network and Off-path), and then describing different caching deployment scenarios and caching algorithms. Icarus [58] is a discrete event simulator specific for ICN caching simulation. This simulator, designed to be extensible and scalable, provides an easy interface to implement and test new caching policies. This extendibility approach is similar to ours, but Icarus is a simulator, while we have presented an emulator.

In [59] and [60], the authors propose a wrapper which pairs a switch interface to a CCNx interface, decodes and hashes content

names in CCN messages into fields that an OpenFlow switch can process (e.g. IP address and port number). Wrapper also enables the OpenFlow switch to forward and monitor *interest* packets using content names. In contrast to our work, they did not provide any platform and real testbed scenarios. The implementation of this work is very simple, it includes a node for generation content, a node for requesting the content and an OpenFlow switch, no GUI for monitoring and experiment evaluation is provided.

In [67], the authors propose a mechanism to deploy ICN in a network domain using SDN paradigm. Briefly, the controller sets up the path rules for transferring an *interest* packet and its returning *data* packet. Differently by us, the scenario assumes that any ICN request is served by a cache node internal to the network domain. Conversely, we also consider the cases in which ICN packets traverse the SDN network domain since the related content is not (yet) available in the internal cache. In [68], the authors present an ICN architecture that uses a SDN approach for routing and caching programmability. While our architecture uses OpenFlow, the architecture in [68] uses the Protocol-Oblivious Forwarding protocol for which the switches can apply powerful match conditions over customizable and variable length fields of the packet header.

The more general aspects of caching in ICN has been studied quite extensively in the literature, hereafter we provide a selection of some references. In [61], random selection centrality-driven caching schemes are used to investigate the possibility of caching less in order to achieve higher performance gain and determine the effectiveness of the schemes. In [62], the authors propose a centralized architecture for ICN without using OpenFlow protocol. The proposed architecture combines Content-Centric Networking and Network Coding (NC). They have considered multi-level caches operating under several different coding situations. It is assumed that the LRU algorithm runs in all caches and the LCD is incorporated in the intermediate nodes. In [20], the authors present a simulation study of CCN to emphasize on caching performance. They have considered several aspects including sizing of cache catalog, popularity, caching replacement and decision policies, topology and single vs multi-path routing strategies. The result of their work shows that the cache catalog and popularity has the most important factor that affects the cache-hit performance. In [63], the authors try to investigate different combinations of naming and digital signature schemes and analysis their impacts on cache-hit probability. In [64], the authors investigate the impact of traffic mix on the caching performance of a tow level caching hierarchy. They have identified four main types of content (Web, file sharing, user generated content and video on demand). Their result shows that the caching performance increases if VoD content is cached towards the edge of the network to leave core with large caches for other type of content. In [65], the authors focus on the performance evaluation of the ICN paradigm and develop an analytical model of bandwidth and storage sharing. Moreover, they provide also a closed form characterization of the average content delivery time.

10. Conclusions

In this paper, we have presented a solution to support ICN by exploiting SDN, implemented by means of Open Source components and offered as a platform for further experiments and developments. We have provided an in depth description of the configuration and forwarding mechanism needed to support the ICN based transfer of content from ICN servers to ICN clients, exploiting in-network caching. The solution supports arbitrary topologies and, in general, it scales linearly with the number of different active chunks that are cached in an in-network cache.

We have implemented a set of basic caching policies using the SDN-based approach and have provided their performance analysis thanks to a set of testbed management and monitoring tools that we have developed.

From a methodological point of view, we have shown that it is possible to deploy the different policies by changing the logic residing in controller modules. Considering that the SDN concepts are currently shaping the evolution of telecommunication networks, we believe that they will play a fundamental role for ICN. In facts, the ICN over SDN approach facilitates the introduction of ICN in production networks and fosters the development of innovative caching policies and mechanisms.

We have provided a reference environment to deploy and test the proposed ICN over SDN infrastructure over local and distributed testbeds. Additional caching policies can be added and further experiments are facilitated by the provided tools. We have given a strong emphasis on replicability and extendibility of our research. To this purpose, the capability to emulate the solution using the widespread Mininet emulator represents an important contribution.

11. Acknowledgments

This work has been partially supported by the EU research projects OFELIA and H2020 EU-JP ICN2020.

12. References

[1] T. Koponen, M. Chawla, B.G. Chun, et al.: “A data-oriented (and beyond) network architecture”, ACM SIGCOMM 2007

[2] V. Jacobson, D. K. Smetters, J. D. Thornton et al., ”Networking named content”, ACM CoNEXT 2009

[3] N. Blefari Melazzi, L. Chiariglione: “The potential of Information Centric Networking in two illustrative use scenarios: mobile video delivery and network management in disaster situations”, IEEE Multimedia Communications Technical Committee E-letter, Vol. 8, N. 4, July 2013.

[4] David Meyer and Darrel Lewis. The locator/id separation protocol (LISP). In RFC 6830, 2013

[5] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, S. Uhlig, “Software-defined networking: A comprehensive survey”, Proceedings of the IEEE, 2015

[6] B. Nunes, M. Mendonca, X. N. Nguyen, K. Obraczka, T. Turletti, “A survey of software-defined networking: Past, present, and future of programmable networks”, IEEE Communications Surveys & Tutorials, 2014

[7] T.D. Nadeau, K. Gray, “Software Defined Networks”, 2013, Published by O’Reilly Media

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, “OpenFlow: Enabling innovation in campus networks”, ACM Communications Review, 2008

[9] S. Salsano, N. Blefari-Melazzi, A.Detti, G.Morabito, L. Veltri: “Information centric networking over SDN and OpenFlow: Architectural aspects and experiments on the OFELIA testbed”, Computer Networks, Elsevier 2013

[10] OFELIA project: http://www.fp7-OFELIA.eu

[11] Mininet project home page, http://mininet.org/

[12] CONET (COntent NETworking) and ICN over SDN Home Page http://netgroup.uniroma2.it/CoNet/

[13] A. Detti, N. Blefari-Melazzi, S. Salsano, and M. Pomposini. “CONET: A Content-Centric Inter-Networking Architecture”, ACM Sigcomm workshop ICN 2011.

[14] A. Detti, M. Pomposini, N. Blefari-Melazzi, S. Salsano, “Supporting the Web with an Information Centric Network that Routes by Name”, Computer Networks, vol. 56, Issue 17.

[15] Named Data Networking (NDN), http://named-data.net/

[16] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, N. Blefari-Melazzi, “Transport-layer issues in Information Centric Networks”, ACM SIGCOMM Workshop ICN-2012.
[17] IEEE 802.1D - http://standards.ieee.org/getieee802/download/802.1D-2004.pdf
[18] Protocol Independent Multicast – Dense Mode - http://www.rfc-editor.org/rfc/rfc3973.txt
[19] Distance Vector Multicast Routing Protocol - http://tools.ietf.org/html/draft-ietf-idmr-dvmrp-v3-11
[20] D. Rossi G. Rossini, “Caching performance of content centric networks under multi-path routing (and more)”, Technical report, Telecom ParisTech, 2011
[21] S. Arianfar and P. Nikander, “Packet-level Caching for Information centric Networking,” in ACM SIGCOMM, ReArch Workshop, 2010.
[22] S. Wang, B. Jun, Z. G. Li, X. Yang, J. P. Wu, “Caching in information-centric networking”, AsiaFI Future Internet Technology Workshop, 2011
[23] Floodlight Home page, http://www.projectfloodlight.org
[24] Netfilter Project Home Page, http://www.netfilter.org/
[25] Marc Sune et al. “Design and implementation of the OFELIA FP7 facility: the European OpenFlow testbed”, Computer Networks, Elsevier 2014
[26] Enterprise GENI (eGENI) project: http://groups.geni.net/geni/wiki/EnterpriseGeni
[27] R. Sherwood, G. Gibb, K. K. Yap, G. Appenzeller, M. Casado, N. McKeown, G. Parulkar, “FlowVisor: A network virtualization layer”. OpenFlow Switch Consortium, Tech. Rep, 2009.
[28] MRTG home page, http://oss.oetiker.ch/mrtg/doc/mrtg.en.html
[29] RRDtool home page, http://oss.oetiker.ch/rrdtool/prog/rrdpython.en.html
[30] D. Cheriton, M. Gritter, “TRIAD: A new next-generation Internet architecture”, Technical report, CSD, Stanford University, 2000
[31] CCNx project web site: http://www.ccnx.org
[32] Conet patch for ccnx https://github.com/StefanoSalsano/alien-ofelia-conet-ccnx
[33] PURSUIT project, http://www.fp7-pursuit.eu
[34] SAIL project, http://www.sail-project.eu/
[35] CONVERGENCE project, http://www.ict-convergence.eu
[36] COMET project http://www.comet-project.org
[37] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, B. Ohlman, “A survey of information-centric networking”, IEEE Communications Magazine, 2012.
[38] G. Xylomenos, et al. “A survey of information-centric networking research”, IEEE Communications Surveys & Tutorials, 2014.
[39] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, A. Detti, “Supporting Information-Centric Functionality in Software Defined Networks”, SDN’12: Workshop on Software Defined Networks, co-located with IEEE ICC, June 10-15 2012, Ottawa, Canada
[40] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, L. Veltri, “An OpenFlow-based Testbed for Information Centric Networking”, Future Network & Mobile Summit 2012, 4-6 July 2012, Berlin, Germany.
[41] D. Trossen, G. Parisis. “Designing and Realizing an Information-Centric Internet”, IEEE Communications Magazine, July 2012
[42] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, L. Tassiulas, “Pursuing a Software Defined Information-centric Network”. In Software Defined Networking (EWSDN), 2012 European Workshop on (pp. 103-108). IEEE.
[43] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, S. Shenker, “Software-defined internet architecture: decoupling architecture from infrastructure”, 11th ACM Workshop on Hot Topics in Networks (HotNets-XI), October 29-30, 2012, Redmond, WA
[44] S. Eum, M. Jibiki, M. Murata, H. Asaeda and N. Nishinaga, "A design of an ICN architecture within the framework of SDN," *2015 Seventh International Conference on Ubiquitous and Future Networks*, Sapporo, 2015
[45] A. Chanda, C. Westphal, “Content as a network primitive”, arXiv preprint arXiv:1212.3341 (2012).
[46] A. Chanda, C. Westphal, “ContentFlow: Mapping Content to Flows in Software Defined Networks”, arXiv preprint arXiv:1302.1493 (2013).
[47] B. J.Ko, V. Pappas, R.Raghavendra, Y. Song, R. B. Dilmaghani, K.-won Lee, D. Verma,“An information-centric architecture for data center networks”. 2nd ICN workshop on Information-centric networking (ICN '12),Helsinki, Finland
[48] M. Caesar, D. Caldwell, N. Feamster, J. Rexford, A. Shaikh, and J. van der Merwe, “Design and implementation of a routing control platform”, in NSDI’05, 2005
[49] T. Bates, E. Chen, R. Chandra “BGP Route Reflection:An Alternative to Full Mesh Internal BGP (IBGP)”, IETF RFC 4456, April 2006
[50] C. E. Rothenberg, M. R. Nascimento, M. R. Salvador, C. N. A. Corrêa, S. C. de Lucena, and R. Raszuk, “Revisiting routing control platforms with the eyes and muscles of software-defined networking”, HotSDN'12 Workshop, Aug. 2012, Helsinki, Finland
[51] V.Kotronis, X. Dimitropoulos, B. Ager,“Outsourcing the routing control logic: better internet routing based on SDN principles”, 11th ACM Workshop on Hot Topics in Networks (HotNets-XI), 2012, New York, USA
[52] M. Gritter, D. Cheriton, “An Architecture for Content Routing Support in the Internet”, Proc. Usenix USITS, March 2001
[53] A. Narayanan, Ed., S. Previdi, B. Field, “BGP advertisements for content URIs”, draft-narayanan-icnrg-bgp-uri-00, Work in progress, July 28, 2012
[54] F. de Oliveira Silva, et al “Enabling Future Internet Architecture Research and Experimentation by Using Software Defined Networking”, European Workshop on Software Defined Networking, EWSDN 2012.
[55] P. Guimaraes, V. Henrique, et al. “Experimenting Content-Centric Networks in the Future Internet Testbed Environment”, Communications Workshops (ICC), 2013 IEEE International Conference on. IEEE, 2013.
[56] M. Tortelli, D. Rossi, G. Boggia, L.A. Grieco. “ICN software tools: Survey and cross-comparison”, Simulation Modelling Practice and Theory, April 2016.
[57] Ibrahim Abdullahi, et al., “Survey on caching approaches in Information Centric Networking“, Journal of Network and Computer Applications, October 2015.
[58] L. Saino, I. Psaras and G. Pavlou. “Icarus: a Caching Simulatorfor Information Centric Networking (ICN)”, Proceedings of the 7th International ICST Conference on Simulation Tools and Techniques, 2014.
[59] X. N. Nguven, D. Saucez, T. Turletti, "Efficient caching in Content-Centric Networks using OpenFlow", IEEE INFOCOM, 2013

[60] X. N. Nguyen, D. Saucez, T. Turletti, “Providing CCN functionalities over OpenFlow switches”, research report - https://hal.inria.fr/hal-00920554/document.

[61] W. K. Chai, D. He, L. Psaras, G. Pvlou, "Cache Less for More in Information-centric Networks (extended version)", Computer Communications, 2013

[62] Qin, Lidu, et al. "Exploring Cache Coding Scheme for Information-centric Networking." *Computational Science and Engineering (CSE), 2014 IEEE 17th International Conference on*. IEEE, 2014

[63] A. Detti, A. Caponi, G. Tropea, G. Bianchi, N. Blefari-Melazzi, “On the Interplay among Naming, Content Validity and Caching Information Centric Networks”, IEEE GLOBECOM 2013.

[64] C. Fricker, P. Robert, J. Roberts, N. Sbihi, “Impact of traffic mix on caching performance in a content-centric network”, IEEE INFOCOM NOMEN Workshop, 2012

[65] L. Muscariello, G. Carofiglio, and M. Gallo. “Bandwidth and storage sharing performance in information centric networking”. ACM Sigcomm workshop ICN 2011.

[66] Li, Jun, et al. "Popularity-driven coordinated caching in named data networking." Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems. ACM, 2012.

[67] M. Vahlenkamp, F. Schneider, D. Kutscher, J. Seedorf, ”Enabling information centric networking in IP networks using SDN” Proceedings of IEEE SDN for of Future Networks and Services (SDN4FNS), 2013

[68] Charpinel, Sergio, et al. “SDCCN: A Novel Software Defined Content-Centric Networking Approach.” 2016 IEEE 30th International Conference on Advanced Information Networking and Applications (AINA).

[69] “Mobile Video Delivery with Hybrid ICN”, CISCO white paper, available at: http://www.cisco.com/c/dam/en/us/solutions/collateral/service-provider/ultra-services-platform/mwc17-hicn-video-wp.pdf