SRSC: SDN-based Routing Scheme for CCN

Elian Aubry, Thomas Silverston, Isabelle Chrisment

SRSC: SDN-based Routing Scheme for CCN

Elian Aubry[†]*, Thomas Silverston[‡] and Isabelle Chrisment[†]*
[†]Université de Lorraine, LORIA – CNRS UMR 7503, France
*Inria, Villers-lés-Nancy, F-54600, France
[‡]The University of Tokyo, JFLI – CNRS UMI 3527, Japan

***Abstract*—Content delivery such as P2P or video streaming generates the main part of the Internet traffic and Content Centric Network (CCN) appears as an appropriate architecture to satisfy the user needs. However, the lack of scalable routing scheme is one of the main obstacles that slows down a large deployment of CCN at an Internet-scale. In this paper we propose to use the Software-Defined Networking (SDN) paradigm to decouple data plane and control plane and present SRSC, a new routing scheme for CCN. Our solution is a clean-slate approach using only CCN messages and the SDN paradigm. We implemented our solution into the NS-3 simulator and perform simulations of our proposal. SRSC shows better performances than the flooding scheme used by default in CCN: it reduces the number of messages, while still improves CCN caching performances.**

I. Introduction

The Internet is today mostly used to access content and the Cisco Visual Networking Index (VNI) [1] predicts that video will count for 72% of the overall Internet traffic by 2019. However, the Internet is based on the IP protocol and a host-to-host communication paradigm and the content delivery was not part of its initial architecture. Recently, several Information-Centric Networking architectures (ICN) have been proposed as an alternative to the classical TCP/IP end-to-end communication model, to deliver content at the Internet-scale such as Content-Centric Networking (CCN) [2]. With the extensive use of services and applications, whose goals are to publish and/or retrieve content (video streaming services, P2P, etc.), CCN provides a novel approach based on the content itself and not on the location of hosts. Packets address names and not hosts and content is stored along the delivery path (*in-network caching*). Any nodes in the network can issue the content for further requests. To date, many studies have focused on *Caching* in CCN [3], [4], [5] and only a few investigates *Routing* in CCN. CCN does not properly have a routing scheme and it is still one of its major issue for future deployment, along with the need to change the entire network infrastructure. Indeed, CCN *Interests* messages are flooded throughout the network, overloading the network, wasting its resources and reducing its overall performances.

At the same time, Software-Defined Networking (SDN) [6] [7] allows decoupling the data plane (traffic forwarding) from the control plane (routing decision) in the network. To this end, a programmable entity (i.e.: the SDN Controller) manages the switches and routers, in such a way that those nodes become forwarding devices only. The routing decisions for all the switches and routers are therefore supported by the controller; it can process forwarding operations at flow level or packet level by pushing rules into hardware devices. A communication protocol (e.g.: OpenFlow [8]) has been defined to enable communication between forwarding devices and the SDN controller.

978-1-4799-7899-1/15/$31.00

Only a few studies have proposed to integrate a SDN controller into a CCN environment in order to provide routing schemes for this architecture [9], [10], [11]. However, these studies still rely on the TCP/IP stack and CCN is therefore used as an overlay at the application level. We argue that this overlay design is a deadlock as it will still refrain deploying the CCN architecture. Indeed, CCN has been designed for a Future Internet to overcome the limitations of the current one, and other studies acknowledged the need of a clean-slate approach [12], [11].

In this paper, we combine the SDN paradigm into the CCN architecture and propose SRSC, a SDN-based Routing Scheme for CCN. Our solution is a clean-slate approach and relies only on the CCN architecture and does not uses IP or SDN traditional communication protocols (e.g.: Openflow). We implemented our proposal into the NS-3 simulator [13]. Our results show that SRSC reduces drastically the network load, while still improving the performances of CCN.

The rest of the paper is organized as follows. Section II provides an overview of the CCN architecture and details the specification of the SRSC protocol. Section III presents our simulation experiments and results. Section IV presents the related work. Finally, Section V concludes the paper and presents future work.

II. SDN-based Routing Scheme

A. Content-Centric Networking overview

The CCN architecture relies mainly on two messages, *Interest* and *Data*. When a user wants to retrieve a content, he publishes an *Interest* message into the network with the content name. When a CCN node receives this *Interest* message, if it has the content in its *Content Store* (CS), it returns a *Data* message to the user through the same interface. Otherwise, the node looks into its *Forwarding Interest Base* (FIB), a table dedicated to do the matching between a content name and the network interface, and forwards the *Interest* message into the network. If there is no entry in its FIB, a node broadcasts the *Interest* message on all its interfaces and then *floods* the network. After receiving an *Interest* message, the CCN nodes update their *Pending Interest Table* (PIT), a table binding the *Interest* message and the network interface it has travelled through, to be able to forward the *Data* on the reverse path up to the requester. All the nodes receiving a *Data* message can store the content into their *Content Store* (CS).

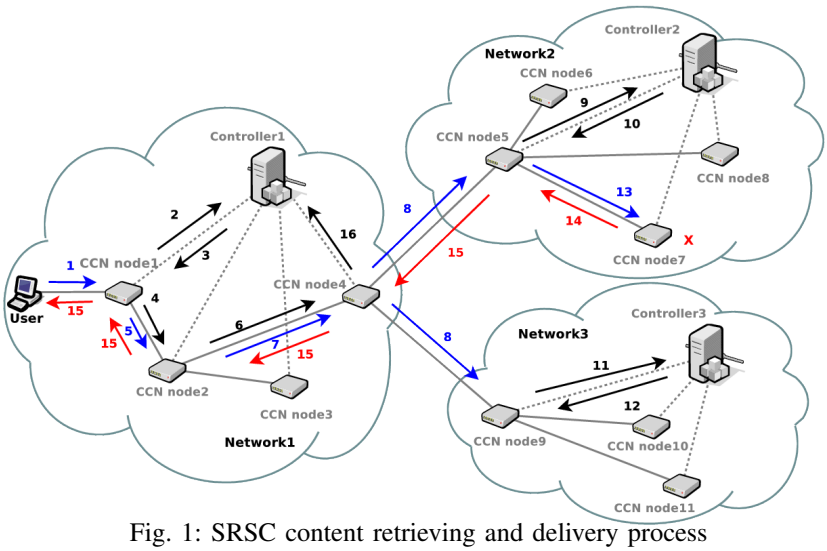

Fig. 1: SRSC content retrieving and delivery process

With CCN, the content is cached into nodes along the delivery path and any nodes can respond a *Data* message to an *Interest*. Thus, there is a high availability of content in the network and further requests will be answered by closer nodes, saving resources of the network. The cache decision process has been largely studied [14], [4], [3], [5] and has an important impact on the overall performances of CCN.

However, in this paper, we focus on the routing decision, i.e., on which node an *Interest* message should be transmitted in order to avoid flooding the entire network and wasting its resources.

B. SRSC overview

The SRSC scheme is depicted on Figure 1, and this figure will be use throughout the rest of the paper. In order to allow communication between nodes and controller without TCP/IP stack, SRSC uses only the CCN *Interest* and *Data* messages to provide a communication channel between nodes and controller. Thus, with SRSC, if a node has no entry in its FIB to forward the *Interest*, instead of broadcasting the message to all its interfaces, it forwards the *Interest* to the controller. The controller should therefore have the global knowledge of the entire network (nodes, topology, paths, content stored into caches) and can reply with a *forwarding rule* piggybacked into the *Data* message, transmitted from the controller to the node along the reverse path. Thus, with SRSC, there is no network flooding to get the content, and the controller provide information to nodes for routing the *Interest* message to the nearest requested content.

The workload of the controller is in correlation with the number of nodes and the size of the network it manages. Thus, a large-scale network should be split into several domains, and each domain has to be managed by a controller. For example, the Figure 1 shows three network domains, with a controller for each of them. Each controller has its own knowledge of the network it manages. A controller knows all the nodes in its network and there are *Border Nodes*, which are nodes connected to several domains (e.g.: node 4, 5 and 9 in Figure 1).

C. SRSC Specification

We present here SRSC, SDN-based Routing Scheme for CCN, and explain our choices by describing all the control messages used by the protocol. These messages are listed in the Table I. First, we detail the *bootstrapping step* allowing the controller to discover nodes, topology and to bind nodes with itself. Then, we describe the *forwarding step*, more particularly, how the controller can fill the nodes' FIB. Finally, we present the messages to announce CS updates (new content cached or removed) to allow the tracking of the content dissemination by the controller. The Figure 1 illustrates the process to discover a content for an user and shows how the content is transmitted in return and cached by nodes.

1) Bootstrapping: During this step, initiated by the controller, there are two phases called *controller announcement* and *neighbour discovery*. For the *controller announcement*, the controller sends a first *Interest* message in the network that will be spread over the network. This message will help to bind nodes to the controller and this phase is the only one flooding the entire network. This *Interest* uses the model 1 referenced in the Table I. ⟨**IdController**⟩ identifies uniquely the controller in the network (e.g.: hardware serial number). When a node receives this *Interest* message, if it is already linked to a controller it drops the message, otherwise it binds to the controller by keeping in memory the controller identity and forwards the *Interest* message to all its others interfaces.

Next, when a node binds to a controller, it starts the *neighbours discovery phase*. This phase allows nodes knowing their neighbours by sending *Interest* message on all faces (model 2 on Table I, where ⟨**IdNode**⟩ identifies the node). When a node receives this *Interest* (model 2), it responds with a *Data* message (using the same prefix as the *Interest* message) that contains its identity and its controller identity. Nodes within the same domain share the same controller ID.

At the end of this process, the nodes can send their local topology to the controller. They send an *Interest* message containing their neighbours to the controller with the model 3. In this message, the ⟨**Neighbours**⟩ attribute contains a list of couple ⟨**IdNode**⟩\⟨**IdController**⟩ representing all neighbours.

	step	function	message
1	Bootstraping	controller announce	/allNodes/controller/⟨IdController⟩
2	Bootstrapping	neighbours discovery	/neighbours/node/⟨IdNode⟩
3	Bootstrapping	topology discovery	/controller/⟨IdController⟩/⟨IdNode⟩/⟨Neighbours⟩
4	Bootstrapping	topology discovery	_⟨IdNeighbour1⟩\⟨IdControllerNeighbour1⟩_⟨IdNeighbour2⟩\⟨IdControllerNeighbour2⟩_
5	Forwarding	rules asking	/controller/⟨IdController⟩/path/node/⟨IdNode⟩/data/⟨Data⟩
6	Forwarding	path establishing	/node/⟨IdNextNode1⟩/install/⟨IdNextNode2⟩/.../⟨IdNode2⟩/data/⟨Data⟩
7	Cache announcement	adding cache	/controller/⟨IdController⟩/incache/⟨IdNode⟩/data/⟨Data⟩
8	Cache announcement	deletion cache	/controller/⟨IdController⟩/deletecache/⟨IdNode⟩/data/⟨Data⟩

TABLE I: Control messages' prefixes used to create the communication channel between nodes and controllers

For example, the couple **_3\1_** signifies that the node sending the *Interest* message is neighbour with the node 3, and that the node 3 is managed by the controller 1. At the reception of this *Interest* message, the controller updates its topology, identifies the border nodes, and responds with a *Data* message that confirms the binding. After this *bootstrapping step*, when all nodes of the network have sent their local knowledge, the controller knows the network topology, and the nodes know how to contact the controller to get forwarding rules.

The *bootstrapping step* is already set and not represented on Figure 1. There are three network domains managed by a distinct controller. A user is connected to the network 1 and a content X is stored at node 7 in network 2.

2) Forwarding: The forwarding step consists in creating a path for routing an *Interest* message to a CS, i.e., asking rules to the controller and pushing them on the path.

This forwarding step is explained with the use of the example on Figure 1. When a node receives an *Interest* message for a content (message 1 on Figure 1), if it has neither the content in its CS nor the rules in its FIB to forward the message, it queries its controller. For example, if a node receives an *Interest* message for the content ⟨**Data**⟩, it sends an *Interest* message to its controller with the model 5 (message 2). When receiving such message, the controller searches into its content table, a table binding the content name with nodes storing the content. If a managed node stores the content, the controller is aware of it. Upon the case, controller can therefore computes the shortest path between the requesting node and the node having content, or the border nodes (nodes that are also in other networks). In our example, the controller computes the paths between node 1 and the border nodes of network 1 (node 5 and node 9) because content X is not available in network 1. If several nodes have the content, it uses the shortest of the paths, and multipath can also be supported by our scheme. The controller responds with a *Data* message containing the entire path in the form of a node identifiers list : (⟨**IdNextNode1**⟩/⟨**IdNextNode2**⟩/.../⟨**IdNode2**⟩) (message 3).

At the reception of the *Data* message containing the path, the node fills its FIB with a rule that means : "for the prefix ⟨**Data**⟩, forward to the first node in the path list sent by the controller". Then, it sends a new *Interest* (model 6) to push forwarding rules along the path (message 4). Next, it sends the original *Interest* to the content ⟨**Data**⟩ on the path (message 5).

When the following node (node 2 in our example) receives the first *Interest* message that contains the path (message 4), it adds rule in its FIB to forward message with prefix ⟨**Data**⟩. When the rule is added, it sends a new *Interest* message to push rules on the path, constructed from the previous message by extracting path informations. This message is sent on the appropriate face defined by the rule previously added. It has also the model 5 (message 6). Then, it receives the *Interest* message ⟨**Data**⟩ (message 5) and forwards it on the path (message 7). This step is repeated by each node on the path until reach the neighbours of the node that has the content or the border node.

In our example, when the node 5 receives the *Interest* message, it queries its controller to find the content X (message 9). The controller 2 knows that the content X is stored in the node 7 and calculates the path between the node 5 and node 7. Controller 2 sends the forwarding rules to the node 5 (message 10). When the node 5 receives the control message, it forwards the *Interest* message for content X to the node 7 (message 13).

At the same time, node 9 receives the *Interest* message, it sends a control message to find the content X to its controller (message 11). Controller 3 does not have the content X in its table, so it looks to find a border node. It has only one border node (node 4) so it responds to node 9 a message that contains rules to forward the *Interest* message up to node 4 (message 12). When node 9 receives this control message, it sees that its PIT contains already an entry to the *Interest* message for content X associated with node 4, so it does nothing.

Now that the *Interest* message has reached the destination, we focus on the Data message to the user. Node 7 has the content X in its content store, so it may respond to the *Interest* message for content X with a *Data* message which contains X, forwarded along the reverse path (message 14). Nodes 5, 4, 2 and 1 have in their PIT the rules to send the *Data* message along the reverse path until the user (messages 15).

3) Cache announcement: In order for the controller to know which content is stored in its network, nodes should indicates explicitly to the controller when a content is added or removed.

When node ⟨**IdNode**⟩ stores in its cache a content ⟨**Data**⟩ as node 4 in our example, it sends an *Interest* message to its controller with the model 6 (message 16). When node ⟨**IdNode**⟩ removes a content called ⟨**Data**⟩ from its cache, it sends an *Interest* message to its controller with the model 7. When a controller receives one of these messages, it updates its content table and can to route further *Interest* message up to the closest Content Store.

III. Simulation Experiments

A. Simulation Environment

We have implemented SRSC into the NS-3 simulator [13] using ndnSIM [15], a NS-3 module that supports Named-Data Networking (NDN) communication model.

The routing scheme implemented allows processing messages with our protocol SRSC by doing *Interest* prefix matching to identify controls messages. In order to evaluate SRSC, we conducted several experiments using the Abilene topology with 11 nodes and 10Mbps links and the Geant topology with 41 nodes and 10Mbps links. We place one controller in each of these two networks. Each node has a cache size capacity of 3 elements and uses the LFU caching strategy commonly used with CCN. When a node receives a *Data* message, it stores the content and if its cache is full, it replaces the least frequently used one. We use a catalogue of 200 contents, randomly distributed into the network. A consumer is randomly placed and send 100 requests per second using the Zipf-Mandelbrot distribution law to decide the requested content. The popularity parameter of the Zipf-Mandelbrots function is set to $\alpha = 0.7$. As example, the catalogue of the PirateBay is modelled with $\alpha = 0.75$ [16].

We performed simulations with SRSC and compared with the flooding strategy used by default in CCN. For the first experiment, we evaluate the bootstrapping step duration time of SRSC. This evaluation allows to verify the speed of the bootstrap and its rapid convergence . We also evaluated the CCN performances by measuring the *Cache Hit Ratio* at each nodes, as well as the number of messages sent per second in the network. The *Cache Hit Ratio* represents the number of *Interest* messages satisfied by a content available in cache, divided by the total number of *Interest* messages sent into the network.

B. Results of Experiments

Results of the first experiments are depicted Figure 2 and show the number of messages sent during simulation time. We can show four lines, that represent four different configurations. The firsts configurations represent the Abilene topology managed by only one (blue line) and two controllers (turquoise line). The two second configurations represent the Geant topology with one and two controllers (green line and red line). Results show that the bootstrapping duration time is relatively quickly and seems to take the same time with 11 nodes (Abilene) and 41 nodes (Geant) and, with one controller or two controllers. Second result shows that the number of messages sent during the bootstrapping process is greater with a bigger topology, but also with a bigger number of controllers. This result was expected because each node must exchange informations with its neighbours, so the number of messages increases proportionally to the number of nodes.

Simulation results of the second experiments are depicted Figure 3 and are presented in four parts. The two graphs on the left represent experiments with the Abilene topology, while the two on the right are for Geant topology. *Cache Hit Ratio* is on the top plots, while number of *Interest* and *Data* messages sent per second are the bottom plots.

For the *Cache Hit Ratio*, blue line represents the SRSC results and green line the flooding results. About the number of messages sent, blue and green lines represent the SRSC *Interest* and *Data* messages sent, the red and turquoise lines correspond to the flooding *Interest* and *Data* messages. On this Figure, we can observe that the *Cache Hit Ratio* of SRSC is better than flooding but slightly improved. This is because

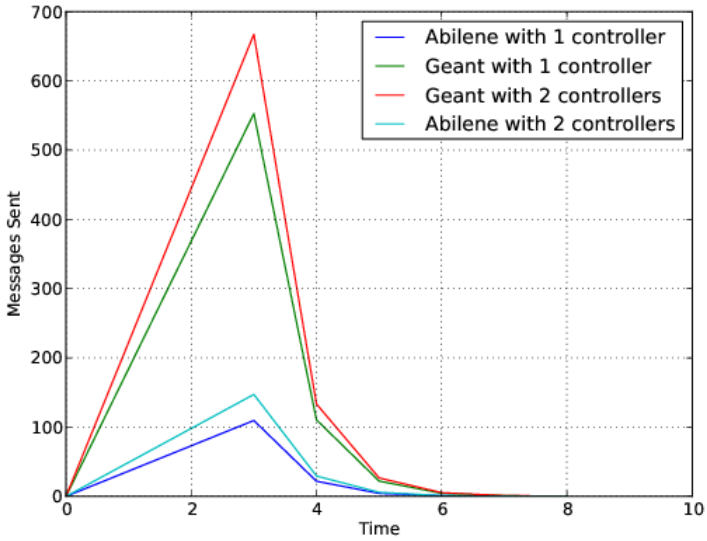

Fig. 2: Bootstrapping step process evaluation time in terms of the number of messages sent

all the control messages and bootstrapping messages are took into account in this ratio, and these control messages decrease the *Cache Hit Ratio* because only the controller can satisfy it. About the number of messages sent, we can see that the number of *Interest* messages per second is divided by ten with SRSC. The number of *Data* message remains approximatively the same in the two solutions.These results show that SRSC can reduce the overload caused by the flooding in CCN networks.

To summarize, our solution based on SDN within CCN allows providing a routing plane to CCN without relying on IP-level or overlay. This routing plane allows to reduce the network overload by reducing the number of messages sent by nodes. Another advantage consists in improving the cache hit ratio of the nodes, the highlight of CCN.

IV. Related work

The possibilities brought by SDN seem to be the key to provide the flexibility and routing plan necessary in CCN. Several studies about SDN integration within Information Centric Networks have already been conducted. In [11], Openflow is extended by adding new specific operations (content-related, routing related and security related operations) to enable ICN nodes to interact with controllers. Nguyen et al. [9] propose an architecture where an abstraction layer is used between CCN nodes and OpenFlow switches to retrieve content name from a CCN packet, hashes this name and encapsulates it into an IP datagram. Chanda et al. [17] also propose to modify the SDN architecture by adding a proxy server, which extracts content information from HTTP traffic and forwards information to controller. As these proposals combine SDN into CCN, they still rely on IP networks and this may limit the native deployment of the CCN architecture.

CCN routing protocols have already been proposed, such as OSPFN [12], which relies on IP network and uses privates IP addresses or named interfaces to create routing paths in network. Hoque et al. propose NLSR [18], an implementation

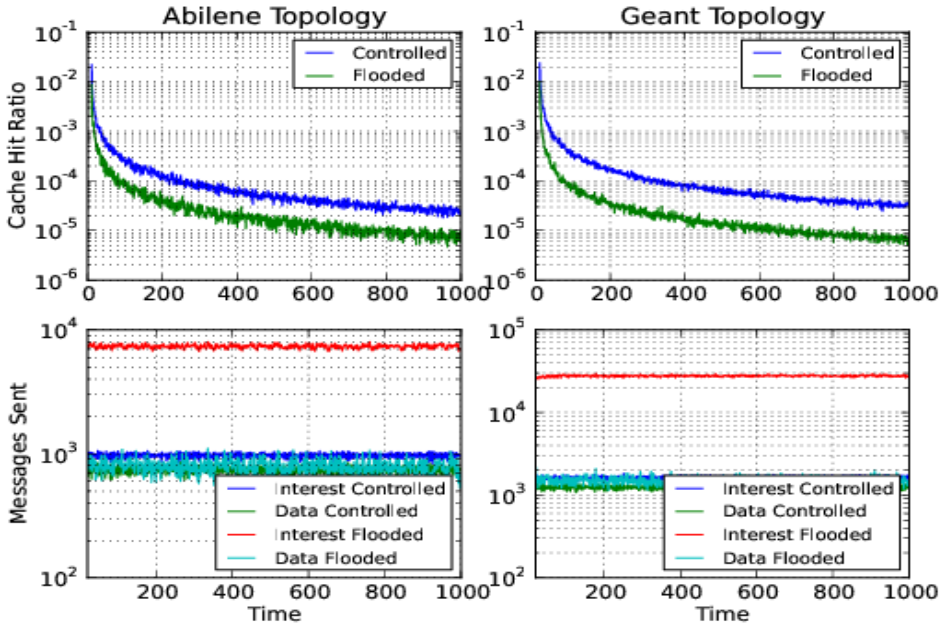

Fig. 3: Evaluation of the *Cache Hit Ratio* and the number of messages sent for SRSC and the flooding strategy in two topologies

IP-free of OSPFN that uses *Interest* and *Data* message to exchange information between nodes. Furthermore, each node stores all paths in its database, while our solution proposes an unique centralized controller to establish path between two nodes on the fly. Moreover, existing proposals for routing in CCN assume routing algorithm designed for single-instance destination, while CCN needs multiple instance destination because content can be stored at several locations.

V. Conclusion

In this paper, we proposed SRSC, a new routing scheme for CCN based on the SDN paradigm. Our proposition is a clean-slate approach and relies only on the CCN architecture, without the TCP/IP stack. SRSC has been implemented into NS-3 and evaluated through simulation experiments. Our results show that SRSC outperforms the flooding scheme used by CCN by reducing the number of messages and still improving the CCN caching performances.

As an IP-based solution may refrain the deployment of CCN, our solution is an important step towards the deployment of native CCN networks. Moreover, we proposed SDN-based functionalities directly implemented into the CCN architecture in order to facilitate interactions between nodes and controllers.

As future work, we want to compare SRSC with NLSR, the other one full CCN routing scheme. Wewill also use the programmability of the controller to enhance other functionalities of CCN, such as security by adding firewall rules, access control or load-balancing traffic.

References

[1] CISCO. Cisco vni. http://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. *ACM CoNEXT*, 2009.

[3] W. K. Chai, D. He, I. Psaras, and G. Pavlou. Cache "less for more" in information-centric networks. In *IFIP Networking*, 2012.

[4] C. Bernardini, T. Silverston, and O. Festor. SACS: Socially-Aware Caching Strategy for Content Centric Networks. *IFIP Networking*, 2014.

[5] J. Ren, W. Qi, C. Westphal, J. Wang, K. Lu, S. Liu, and S. Wang. Magic: a distributed max-gain in-network caching strategy in information-centric networks. In *IEEE INFOCOM*, 2014.

[6] Open Networking Foundation. Software-Defined Networking: The New Norm for Networks. White paper, Open Networking Foundation, Palo Alto, CA, USA, April 2012.

[7] M. Mendonca, B. A. Nunes, X. Nguyen, K. Obraczka, and T. Turletti. A Survey of Software-Defined Networking: Past, Present, and Future of Programmable Networks. *IEEE Surveys Communications and Tutorials*, 16(3):1617 – 1634, 2014.

[8] N. Mckeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner. Openflow: Enabling innovation in campus networks. *ACM SIGCOMM CCR*, 2008.

[9] X. Nguyen, D. Saucez, and T. Turletti. Providing CCN functionalities over OpenFlow switches. Research report, INRIA, August 2013.

[10] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf. Enabling information centric networking in ip networks using sdn. In *IEEE SDN4FNS*, 2013.

[11] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri. Information centric networking over sdn and openflow: Architectural aspects and experiments on the ofelia testbed. *Comput. Netw.*, 57(16):3207–3221, November 2013.

[12] C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. Technical report, Named Data Networking, 2012.

[13] NS-3 Consortium. projet ns-3. https://www.nsnam.org/.

[14] D. Saucez and T. Turletti. Efficient caching in Content-Centric Networks using OpenFlow. *IEEE INFOCOM*, 2013.

[15] A. Afanasyev, I. Moiseenko, and L. Zhang. ndnSIM: NDN simulator for NS-3. Technical Report NDN-0005, NDN, October 2012.

[16] C. Fricker, P. Robert, J. Roberts, and N. Sbihi. Impact of traffic mix on caching performance in a content-centric network. In *IEEE INFOCOM*, 2012.

[17] A. Chanda and C. Westphal. A content management layer for software-defined information centric networks. ACM ICN, 2013.

[18] A K M M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. Nlsr: Named-data link state routing protocol. ACM ICN, 2013.