NDNFlow: Software-defined named data networking

van Adrichem, NLM; Kuipers, FA

DOI
10.1109/NETSOFT.2015.7116131

Publication date
2015

Document Version
Accepted author manuscript

Published in
Proceedings of the 1st IEEE conference on network softwarization, IEEE NetSoft 2015

NDNFlow: Software-Defined Named Data Networking

Niels L. M. van Adrichem and Fernando A. Kuipers

Network Architectures and Services, Delft University of Technology
Mekelweg 4, 2628 CD Delft, The Netherlands
{N.L.M.vanAdrichem, F.A.Kuipers}@tudelft.nl

***Abstract*—In this paper, we introduce NDNFlow: an open-source software implementation of a Named Data Networking based forwarding scheme in OpenFlow-controlled Software-Defined Networks (SDNs). By setting up an application-specific communication channel and controller layer parallel to the application agnostic OpenFlow protocol, we obtain a mechanism to deploy specific optimizations into a network without requiring a full network upgrade or OpenFlow protocol change.**

Our open-source software implementation consists of both an NDN-specific controller module and an NDN client plug-in. NDNFlow allows OpenFlow networks with NDN capabilities to exploit the benefits of NDN, by enabling the use of intermediate caches, identifying flows of content and eventually performing traffic engineering based on these principles.

I. Introduction

Recently, Software-Defined Networking (SDN) has gained the interest of both research and industry. For research, SDN opens up the possibility to implement optimizations that previously were theoretical in nature due to implementation complexity. For industry, SDN delivers a way to dynamically monitor and control the network beyond the capabilities of self-organized distributed traffic engineering and failover mechanisms.

OpenFlow [1] is often considered to be the de facto standard to implement SDN. Other emerging future internet architectures, such as Information Centric Networking (ICN), introduce application-specific forwarding schemes. In particular, we believe that SDN and ICN can benefit from each other. SDNs can benefit from the power of caching from ICNs and in general need to be able to quickly adapt to new application-specific forwarding schemes, such as ICNs. ICNs, on the other hand, greatly benefit if they can be adopted with little effort by already existing SDNs. Furthermore, the benefits of SDN to IP, being greater management control and monitoring over the network, also apply to ICNs. Finally, ICNs benefit from SDNs as they can efficiently distribute content in partially upgraded networks, removing the necessity to upgrade the full network and thus easing the deployment and transition phase.

In this paper, we discuss our experiences in setting up an SDN for the application-specific forwarding mechanism of Named Data Networking (NDN) [2], a popular ICN implementation. Although this paper is dedicated to setting up an SDN-supported ICN, our experiences and decisions also apply to other forwarding mechanisms that may emerge.

In section II, we first discuss the initial principles of SDN and OpenFlow. In section III, we explain the functionality of the ICN implementation NDN. Section IV presents our two initial proposals toward application-specific SDNs and reasons why we think these approaches are infeasible for standardization. Section V proposes our mechanism in which we have divided the SDN in two layers: the regular OpenFlow layer based on traditional forwarding mechanisms, supplemented with an application-specific layer. Section VI presents the exact details of our implementation. Section VII presents experimental measurements performed on NDNFlow. Finally, section VIII concludes this paper.

II. Software-Defined Networking

In its initial form, Software-Defined Networking (SDN) concerns separating the control plane (decision functions) from the data plane (forwarding functions) in networks. This enables a more flexible form of networking in which abstract business rules in terms of robustness, security and QoS can be translated into a network configuration policy. In turn, the configuration policy can be configured in the networking devices using either an abstract configuration interface (such as OpenFlow [1], OpenFlow Config [3], OVSDB [**?**], ForCES [**?**] or NetConf [**?**]) or vendor-specific configuration parameters.

SDN allows applications to request QoS parameters on-the-fly from a network control agent. This enables scenarios in which applications can offer guaranteed quality by negotiating the service they need from the network and ultimately pay for that service for the time they need it.

Software-Defined Networking is often associated with the network configuration protocol OpenFlow [1]. OpenFlow is a vendor-independent protocol which can configure network nodes both in advance, and in a reactive fashion. OpenFlow-enabled switches connect to a single controller entity, which configures the switches based on their topological properties and predefined rules concerning routing, firewalling and QoS. Additionally, when a switch receives a packet for which it has no installed flows yet (i.e., it is a new connection not matching any predefined rules), it sends this packet to the OpenFlow controller. The OpenFlow controller performs access control and computes the appropriate path for the new data flow and configures all switches accordingly.

III. Named Data Networking

In this section, we summarize Named Data Networking (NDN) [2] and its implementation CCNx [4]. CCNx implements an Information Centric Network (ICN) by using a route-by-name principle. In contrast to identifying by source and destination IP addresses, NDN identifies Interest and ContentObject packets by one or more name components (for example, Bob could publish his holiday photos under the name /bob.eu/holidayPhotos). A user-client requests content by sending out an Interest containing a name describing the desired information. Intermediate nodes on the path from client to server forward the expressed Interest hop-by-hop to the generator responsible for the requested name. When the Interest reaches a node that has a cached copy satisfying the description of the Interest, the Interest is dropped and the data is delivered from cache. If not, the Interest travels the path to the content generator, which creates a ContentObject and delivers it to the client accordingly.

The resulting content is encapsulated in a ContentObject and forwarded along the exact reverse path back to the client. Functionally, each intermediate node administers the following three tables in its memory:

1) The ContentStore (CS), which contains cached copies of previously delivered content.
2) The Pending Interest Table (PIT), which stores recently forwarded Interests and their originating interfaces.
3) The Forwarding Information Base (FIB), containing forwarding rules based on the NDN names.

Each incoming Interest is compared to the content of the CS to determine if it can be fulfilled from cache immediately. If not, the packet name is compared to the PIT content to prevent forwarding of duplicate requests. Finally, the FIB is consulted to determine the forwarding actions. The Interest and its originating interface are added to the PIT, enabling the resulting ContentObject to travel to the requester by source-based routing. Where IP prefix matches on a fixed number of bits, NDN prefix matches on a variable number of subsequent name components. With each subsequent component, the Interest name adds a restrictive element to the possible set of valid ContentObjects. For example, an Interest named /bob.eu/holidayPhotos prefix matches a ContentObject for /bob.eu/holidayPhotos/2013, as the requester did not specify the exact holiday period or location.

IV. Related Work

In order to use SDN and OpenFlow to set up ICN networks, we have evaluated multiple techniques before coming to our final proposal in section V. In this section, we discuss previous initiatives and parts of our early work and argue why we think these are not feasible for standardization.

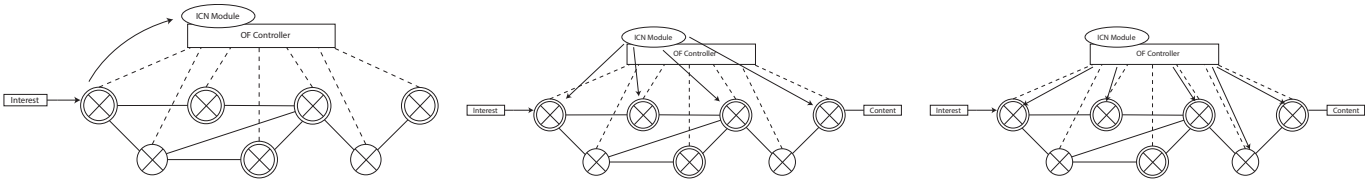
A straightforward way to implement ICN using SDNs is to implement ICN functionality into the Open vSwitch specification and enhance the OpenFlow protocol to support ICN names, as suggested in [5]. We, however, think the joint maturing of both ICN protocols and the OpenFlow protocol will increase the complexity of realizing a stable standardization of OpenFlow that supports both regular IP/Ethernet forwarding and ICN. As [6] shows, the concept of naming in NDN is, among others, still subject to further optimization to decrease routing table size and thereby increase the forwarding efficiency. Given that the standardization of OpenFlow for regular packet forwarding is already a complex task, chances that standardization will include application-specific forwarding schemes are small.

Even if standardization would include an ICN protocol, we foresee a rise in application-specific forwarding schemes in general to optimize the Internet for the most frequently used applications. One application-specific adaptation of OpenFlow, or any SDN paradigm for that matter, would exclude other application-specific forwarding schemes facing identical problems.

Both [7] and [8] propose to reuse IP's address and port fields to contain hashes of content names in order to allow OpenFlow switches to forward Interests to an OpenFlow controller that performs path calculation. Where [7] uses additional IP-options to indicate ICN packets, [8] remains agnostic to how ICN packets are distinguished from regular IP packets. We argue that this approach leads to an excessive increase in IP routing table complexity.

Similarly, we at first intended to wrap or encapsulate ICN streams in IP packets containing a reserved IP anycast address to allow fine-grained control beyond the scope of ICN-capable switches. We found this method to be less trivial than it appears. Where CCNx is already capable of performing UDP over IP encapsulation, it uses static ports for each connection, disabling a switch to differentiate between different flows. As the CCNx application is OpenFlow unaware by nature, changing it by generating different tuples of source IP address and the 4 bytes of the UDP source and destination ports implied a drastic change to the internal functions of the CCNx daemon. More generically, forcing developers to create port tuples in such a specific way in order to benefit from SDN functionality is in contrast with the open philosophy of SDN. Furthermore, OpenFlow switches may not send the complete incoming CCNx packet to the controller, they may apply buffering to recreate the original message when necessary, but instead might only forward the first part of the message. This implies possibly losing parts of the ICN name, information necessary for the ICN controller to perform path computation.

Finally, P4 [9] and POF [10] respectively implement a packet processor description language and forwarding architecture design to allow protocol-oblivious forwarding, work resulting in the ONF OF-PI proposal [11]. However, P4 implements static field sizes, rendering it unsuitable for use with the CCNx implementation, which carries a variable amount of variable-sized name components. Furthermore, while using $\{offset, length\}$ search keys as proposed in POF may work, we consider the complexity of rewriting all abstract comparisons and functions to bit-wise operators too tedious.



(a) Step 1, local NDN daemon receives an unassociated Interest and forwards this Interest to the ICN module of the OpenFlow Controller.

(b) Step 2, with the knowledge on topology and ICN capabilities, the ICN module computes a feasible path and configures the appropriate ICN-enabled switches accordingly.

(c) Step 3, the ICN module instructs the legacy OpenFlow controller to configure the necessary IP/Ethernet rules on all intermediate OpenFlow enabled switches.

Fig. 1: An overview of the steps necessary to configure an ICN flow over an OpenFlow-enabled network. All switches are OpenFlow capable, doubly-circled nodes additionally have ICN capabilities.

V. NDNFlow

Given that neither extending the OpenFlow protocol for application-specific forwarding schemes nor using application-specific IP broadcast addresses to distinguish ICN traffic from regular IP traffic is feasible, NDNFlow introduces a second application-specific layer to OpenFlow. NDNFlow implements a separate communication channel and controller module parallel to the already existing OpenFlow communication channel and process.

In this application-specific layer, all communication and path computation regarding the ICN are handled separately from the regular IP and Ethernet plane. By separating the ICN layer from the regular OpenFlow layer, we introduce SDN functionality independent of changes and restrictions in the standardization of the OpenFlow protocol. This prevents inter-dependencies on versions of protocols, easing the deployment and future maintenance. As shown in figure 1, switches that are ICN enabled set up a communication channel, parallel to their regular OpenFlow communication channel, to the ICN module of the OpenFlow controller. This ICN channel is then used to announce ICN capabilities, information availability and requests for unreserved flows. The controller's ICN module computes paths for ICN flows, and configures both the ICN capable and legacy IP and Ethernet switching fabric to allow ICN flows to pass through the network. Hence, we introduce a separate SDN control mechanism for the application-specific OSI Layers 5 to 7 (ICN), independent from OSI Layers 2 to 4, reusing the separation of layer responsibilities to maintain overall network manageability.

Where ICN-enabled switches receive ICN-specific flows directly, flows between ICN-enabled switches that are initially unreachable due to intermediate legacy IP and Ethernet switches are realized by setting up IP-encapsulated tunnels. The legacy switches are configured by the legacy OpenFlow controller to forward those tunnels accordingly. Hence, the configuration procedure consists of 4 steps shown in figure 1, where a doubly-circled node represents a switch capable of both ICN and OpenFlow.

Due to the fact that both the ICN-enabled switches and the ICN controller module are aware of the specifics of the ICN forwarding mechanism, they have equal understanding of an ICN flow and its details. Furthermore, their communication protocol can be extended to contain flow-specific parameters, such as the needed bandwidth and the expected duration of a flow, without changing the OpenFlow protocol.

VI. Implementation

Our software currently runs on general purpose x64 architecture servers running Ubuntu Server. On these servers, we have installed stock Open vSwitch 2.0.2 [12] to enable configuration of operation by the OpenFlow protocol. In addition, we enable switches with ICN capabilities by installing the CCNx daemon [4], the open-source implementation of NDN.

A. OpenFlow controller implementation

In order to implement our proposal, we have extended the POX (branch betta) controller [13] by designing an additional custom ICN module. We use the native POX Discovery module to perform topology discovery and learn a switch adjacency matrix. We reuse the OpenNetMon [14] forwarding module to perform legacy path computation and enable end-to-end IP forwarding. Additionally OpenNetMon may be used to perform fine-grained monitoring of flows. Finally, we implement a CCNx specific plug-in that is added to communicate with ICN-enabled switches and perform ICN-specific path computations. The implementation of NDNFlow is published open source and can be found at our GitHub web page [15].

B. CCNx daemon implementation

The CCNx daemon is extended by implementing an additional SDN plug-in, which sets up a connection to the POX ICN module, parallel to Open vSwitch's regular OpenFlow connection, and announces its ICN capabilities, capacity and information availability. The extension is realized similarly to our plug-in solving global NDN routing table complexity [6]. Whenever a CCNx daemon receives an Interest for which no flow or previously defined forwarding rule exists, it forwards this Interest to the POX ICN module. In turn, the POX ICN module looks up the appropriate location or exit-point of that Interest, calculates the appropriate path based on the topology information learned from the discovery module and announcements from CCNx-enabled switches and configures the intermediate NDN nodes accordingly. Finally, the Open vSwitch is configured by the controller as shown in figure 1.

C. Protocol Implementation

We chose to use the JavaScript Object Notation (JSON) to facilitate communication between the CCNx and SDN module due to its generic implementation and high support in different programming languages. Currently, we implement the following abstract messages to support our actions.

```
Announce{
        DPID : <DataPathID>,
        IP : <InetAddress>
}
```

The Announce message is used by nodes to propagate their ICN abilities. More precisely they state where they are connected in the OpenFlow network using the unique Datapath ID (DPID) of the switch, and how the ICN functions can be accessed by IP.

```
AvailableContent{
        Content : [
                <(ContentName) Name> : {
                        Cost : <Integer>,
                        Priority:<Integer>
                }
        ]
}
```

After authentication, the ICN-enabled switch propagates the information it has access to using the AvailableContent message. Each item can be stored locally or accessed elsewhere, for example via a network outside of the scope of the SDN, and additional costs can be added which are taken into account by routing discovery. Absolute backup replicas, which are only to be accessed when the primary replicas are unavailable, can be announced by increasing the value of the Priority field. Hence, robustness can easily be implemented by placing redundant copies of information across the network.

```
IncomingInterest{
        Name : <ContentName>
}
```

The IncomingInterest message is used by the CCNx module to request the controller what action to perform with unmatched incoming and following Interests.

```
InstallFlow{
        name : <ContentName>,
        action : <FaceType>,
        actionParams : [<params>]
}
```

After computing the appropriate actions, the controller issues an InstallFlow message to all the switches along the path to install the correct forwarding rules, reducing the original interest name to match the name prefix of a complete flow or segmented piece of information. The FaceType and action parameters can be used to configure flow-specific parameters. Among others, we use them to set up IP-encapsulated tunnels between ICN nodes that are separated by one or more ICN-incapable switches to enable flow exchange between them.

VII. Experimental Evaluation

In this section, we will first discuss our experimental setup and the used measurement techniques, followed by the conducted experiments and results.

A. Testbed environment

We have conducted our experiments on a testbed of physical, general-purpose, servers all having a 64-bit Quad-Core Intel Xeon CPU running at 3.00 GHz with 4.00 GB of main memory and 1 Gbps networking interfaces. OpenFlow switch functionality is realized using the Open vSwitch 2.0.2 software switch implementation, Named Data Networking functionality by installing CCNx 0.8.2, both running in parallel on Ubuntu Server 14.04.1 LTS with GNU/Linux kernel version 3.13.0-29-generic. The CCNx is connected to Open vSwitch via a socket to the internal bridge interface to realize connectivity, hence data is forwarded to and from CCNx through the OpenFlow LOCAL port.

Throughout, we use a 2-switch topology on which the discussed switching fabric and additional plug-ins from section VI are configured. A third server is configured as controller using the POX controller and modules discussed in section VI.

In order to measure the delay time between requesting and receiving content we use *ccnping* [16], an NDN alternative to the popular application *ping* that can be used to confirm connectivity and measure round-trip times in classical IP networks. Similar to *ping*, *ccnping* sends an Interest per interval to a given destination prefix concatenated with a random value. When sending, *ccnping* stores the timestamp of creation and computes the round-trip time (RTT) upon arrival of the appropriate ContentObject. The *ccnping* server and client are installed on the 2 switches and connect to the CCNx switch fabric using the application interface.

B. Experiments and results

Using the described testbed and tools, we have performed 4 types of experiments to evaluate the suitability and stability of NDNFlow. In our experiments, we differentiate between the proactive and reactive SDN approaches in which flows are respectively configured in advance, or on-the-fly. As the decision between proactive and reactive configuration can be made independent for both the CCNx and OVS specific forwarding fabric, we perform the following 4 experiments: (1) We determine a baseline by measuring RTTs using a statically configured CCNx over classic IP. (2) We determine the overhead of Open vSwitch and the NDNFlow CCNx-plug-in by measuring RTTs in a proactively configured CCNx and Open vSwitch network. Since the biggest difference between proactive and reactive configuration lies within the delay of setting up the flow (measurable by the delay of the first packet), we continue measuring the delay of the first packet of every new flow with a reactive configuration of CCNx in both a (3) proactive and (4) reactive configured OVS network.

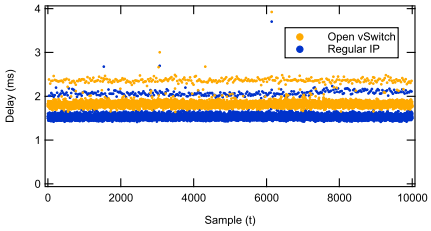


Fig. 2: Baseline measurements using CCNx over regular IP and OVS.

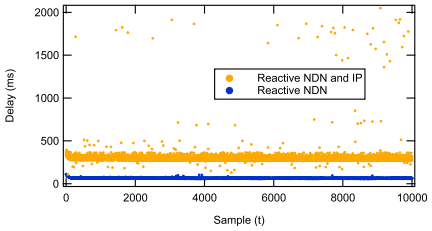


Fig. 3: Measured delay of CCNx path setup in pro- and reactive configuration.

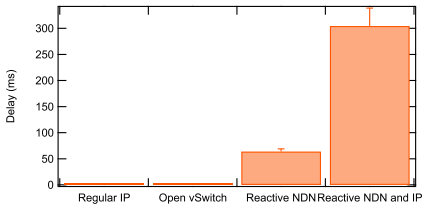


Fig. 4: Delay averages and 95% confidence interval.

While experiment (3) shows the delay invoked by computing and configuring the path of the content flow, (4) shows the delay invoked by additionally configuring the legacy IP part of the OpenFlow network. We measured $10,000$ samples for each configuration.

Figure 2 shows the results for (1) and (2), while figure 3 shows the results for (3) and (4). Figure 4 shows the relative averages and 95% confidence interval, giving: (1) a baseline of 1.534 ± 0.101 ms for CCNx over IP networks, (2) 1.834 ± 0.115 ms for a fully proactive configuration of CCNx and OVS, and (3, 4) 64.006 ± 5.028 and 304.630 ± 34.191 ms for a reactive NDNFlow configuration in a proactive and reactive OVS configuration, respectively, to determine the additional costs of configuring the content flows in CCNx and OVS.

The measured values show that, on average, OVS adds an additional delay of 0.300 ms, while configuring a new content flow using NDNFlow costs an additional 62.172 ms at the CCNx daemon and another 240.624 ms at the OVS daemon. Although setting up new flows can be considered costly, the additional delay only applies to the first packet of a new flow. Once a flow has been installed, the delays of experiment (2) apply. Using a completely proactive configuration would remove the additional delay of methods (3) and (4) altogether, though at the cost of losing the flexibility of computing flow-specific paths.

VIII. Conclusion

In this paper, we have presented and designed a mechanism and implemented a prototype to realize application-specific forwarding schemes in OpenFlow-controlled Software-Defined Networks (SDNs). Specifically, we have implemented a popular Information Centric Networking proposal, Named Data Networking and its implementation CCNx. Compared to other application-specific SDN implementations, we argue that our implementation is architecturally less complex to implement, easier to extend and furthermore applicable to multiple application-specific forwarding schemes due to the stricter separation of functionalities. With this implementation, we provide the tools to control and manage application-specific flows in SDNs.

References

[1] 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 Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[2] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[3] Open Networking Foundation, "OF-CONFIG 1.2: OpenFlow Management and Configuration Protocol," https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow-config/of-config-1.2.pdf, 2014.

[4] Palo Alto Research Center, "CCNx," http://www.ccnx.org/, Dec. 2012.

[5] J. Suh, "OF-CCN: CCN over OpenFlow," http://netlab.pkusz.edu.cn/wordpress/wp-content/uploads/2012/07/Content-Networking-over-OpenFlow.pdf, 07 2012.

[6] N. L. M. van Adrichem and F. A. Kuipers, "Globally accessible names in named data networking," in *Proc. IEEE INFOCOM Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN)*. IEEE, 2013.

[7] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An openflow-based testbed for information centric networking," *Future Network & Mobile Summit*, pp. 4–6, 2012.

[8] X. N. Nguyen, D. Saucez, T. Turletti *et al.*, "Efficient caching in Content-Centric Networks using OpenFlow," in *INFOCOM 2013 Student Workshop*, 2013.

[9] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, "P4: Programming Protocol-independent Packet Processors," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 87–95, Jul. 2014.

[10] H. Song, "Protocol-oblivious forwarding: Unleash the power of SDN through a future-proof forwarding plane," in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*. ACM, 2013.

[11] Open Networking Foundation, "OF-PI: A protocol independent layer," https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/OF-PI__A_Protocol_Independent_Layer_for_OpenFlow_v1-1.pdf, 09 2014.

[12] B. Pfaff, J. Pettit, K. Amidon, M. Casado, T. Koponen, and S. Shenker, "Extending Networking into the Virtualization Layer." in *Hotnets*, 2009.

[13] M. McCauley, "About POX," http://www.noxrepo.org/pox/about-pox/, Aug. 2014.

[14] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, "OpenNetMon: Network Monitoring in OpenFlow Software-Defined Networks," in *Network Operations and Management Symposium (NOMS)*. IEEE, 2014.

[15] N. L. M. van Adrichem and F. A. Kuipers, "TUDelftNAS/SDN-NDNFlow," https://github.com/TUDelftNAS/SDN-NDNFlow, Mar. 2015.

[16] (2014, Dec.) NDN-Routing/ccnping. https://github.com/NDN-Routing/ccnping.