TOWARDS CONTROLLER-AIDED MULTIMEDIA DISSEMINATION IN NAMED DATA NETWORKING

Florian Bacher, Benjamin Rainer and Hermann Hellwagner

Institute of Information Technology (ITEC)
Alpen-Adria Universität (AAU) Klagenfurt, Austria
firstname.lastname@itec.aau.at

ABSTRACT

Software Defined Networking (SDN) and Named Data Networking (NDN) are two topics which have received lots of attention in the networking research community in recent years. While both have emerged independently from each other we believe that their core features can be well aligned to each other. Hence combining both may hold potential benefits for network operators. In this paper we investigate the advantage of having a central SDN controller which is aware of the complete topology of an underlying NDN network. In our approach we use the controller for routing Interests for names unknown to the forwarding elements and to find alternative routes in case of link congestion. Another advantage of SDN is the ability to analyze and control the network on an application-layer component which communicates with the controller. This allows the development of application-aware networks that support the specific needs of the applications that use them. As an example use case we assumed a network whose main purpose is to disseminate multimedia content with Zipf-distributed popularity among users. Having an application layer which knows about content popularity statistics we improve the dissemination of multimedia content by instructing dedicated nodes in the network to prefetch content which is expected to become popular in their geographical region or autonomous system (AS) in the near future. The aim of this approach is to reduce the distance to potential consumers and reduce the load of the core network.

Index Terms— Information Centric Networking, Named Data Networking, Software Defined Networking, Routing, Forwarding, Caching

1. INTRODUCTION

Named Data Networking (NDN) is an approach to networking whose ultimate goal is to replace the current IP-based Internet architecture. The main concept of NDN is based on the observation that nowaday's Internet traffic is primarily focused on the content itself rather than on the direct communication between hosts [1]. In NDN, for example, if a client wants to consume a certain video over the network it does not care about which server provides the appropriate content. Instead, it sends an Interest, which contains the name of the requested content, to the network rather than directly addressing the server which provides the content. An important aspect of NDN is the inherent caching of data. This means that every forwarding element, i.e., router, in NDN is able to cache chunks of data which are identified by the name of the content they belong to. Whenever a router receives an Interest for a chunk already available in its cache, it can directly send back the chunk instead of forwarding the Interest to the origin of the content. Another crucial concept of NDN is the separation of routing and forwarding, i.e., forwarding decisions for incoming Interests can be done independently from the routing protocols used by routers, as opposed to IP networks where forwarding is strictly determined by routing in order to avoid loops.

Currently there are still a number of open research questions in the area of NDN. One major topic is the issue of scalability. Since in NDN content is addressed rather than hosts this leads to a bigger address space which makes it difficult for routers to find routes and maintain their forwarding tables. This is where Software Defined Networking (SDN) comes into play. SDN aims at reducing the complexity of networking elements such as routers and switches by moving the control plane to a centralized controller [2]. This way, networking elements are reduced to simple forwarding elements which receive forwarding rules from the central controller via a so called southbound (SB) interface. This enables easy programmability of networks since changes can be specified at a single location, as opposed to reconfiguring every device within the network separately via their proprietary interfaces. This programmability in combination with the holistic view on the network provided by the controller enables network operators to dynamically react to events such as an increase of overall traffic load or failing links.

In this paper we focus on the use case of disseminating multimedia content with Zipf-distributed content popularity, i.e., where the major amount of network traffic is caused by a small number of popular multimedia content items, as described in [3]. Therefore, we extend the SDN approach to proactively cache popular multimedia content such that it is stored as near as possible to the clients that will request the specific content. We compare our approach to the pure SDN approach which only re-computes routes to the multimedia content and a pure NDN based forwarding approach in terms of Interest satisfaction ratio and the round trip time of Interests.

The remainder of this paper is structured as follows. In Section 2 we provide an overview of related work in the area of combining SDN with Named Data Networking (NDN) [4], a concrete implementation of NDN. Section 3 contains detailed information about our approach of using SDN in order to allow efficient and reactive routing and forwarding in NDN networks. Section 4 describes how statistics regarding content popularity gathered on the SDN application layer can be used to proactively instruct nodes in the network to request certain content items in order to bring them nearer to potential consumers. We give an overview of our evaluation methodology and the results obtained by our approach in Section 5 before we conclude the paper and outline our plans for our future work in Section 6.

2. RELATED WORK

There is already existing work that focuses on how to realize Information Centric Networking in Software Defined Networks, such as in [5][6][7][8]. In these papers the authors mainly describe concrete implementations and demonstrate prototypes for realizing NDN over SDN. In each of these papers, *OpenFlow* [9], the most popular SDN standard, was used since its concept of flows and pushing rules to forwarding hardware translates well to routing and forwarding in current NDN approaches. One of the key challenges regarding the implementation of NDN using OpenFlow is that OpenFlow switches only operate on the packet header of incoming IP packets and are not aware of current NDN protocols. Therefore, the authors of [8] propose to map NDN names to fields which can be inspected by OpenFlow, such as the IP address or port number. A different approach proposed by [6] is to parse the content of NDN packets at the controller in order to gain access to the fields of the respective NDN protocol fields. However, as this approach would require frequent deep packet inspection (DPI) on the controller doing so could potentially cause scalability issues.

3. CONTROLLER-AIDED ROUTING AND FORWARDING IN NAMED DATA NETWORKING

In this section we briefly revisit the design of an SDN controller-aided routing and forwarding approach in the context of Named Data Networking (NDN) [4]. A current research topic in NDN is the scalability of the forwarding plane. In contrast to IP-based networks, where hosts are addressed, NDN follows the principle that content items are addressed ("named"). This results in a bigger address space which makes it difficult for NDN routers to discover paths for unknown Interest names and maintain their forwarding tables. With our proposed approach, we try to benefit from a number of potential advantages of having a central SDN controller which is aware of the complete network topology.

First, as described in [7], having a central controller can help overcome the scalability issue of maintaining a name-based forwarding table within each NDN router. Figure 1 illustrates the basic principle of our controller-aided routing and forwarding. Our architecture consists of a network of

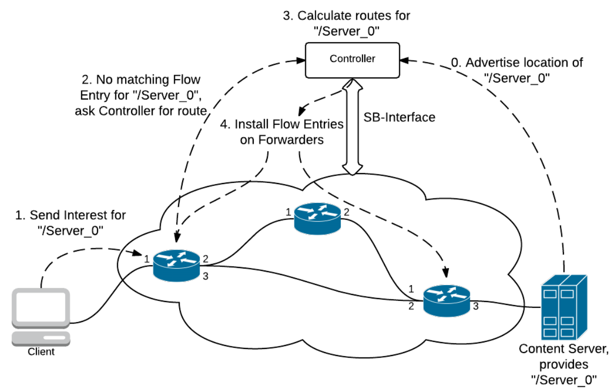


Fig. 1. SD-NDN Routing Example.

NDN routers which communicate with a central controller over a southbound (SB) Interface. In our approach, all content providers have to advertise their available content names to the controller in order for it to know about the content locations (cf. Figure 1, step 0). Whenever a client wants to consume content it sends an Interest with the content name to the next NDN router (step 1). Whenever a router receives an Interest for which the content name is unknown, the Interest is forwarded to the controller via the SB interface (step 2). Upon reception of such Interests, the controller calculates possible routes from the router to the location of the content (step 3). During this step, the controller can also take the current status of the links between the network nodes into account in order to find the best route based on given metrics. These metrics may be delay, bandwidth, hop count and/or reliability. In this context we define the reliability as the fraction of satisfied Interests sent via a link and the total number of Interests received for a given content item. As soon as the controller has calculated one or more possible routes to the content location, it will push the appropriate rules, i.e., flow entries to all nodes which lie on the possible paths via the SB interface (step 4). In our approach, a flow entry consists of an Interest prefix, a list of possible faces over which matching Interests can be forwarded and the costs for each face (i.e., hop count).

In addition to sending Interests to the controller and re-

Algorithm 1 Face Selection from Local Forwarding Table (executed on each Router)

```
1: Input: inFace, prefix
2: Output: Forward Interest or send Nack to inFace
3: excl = [inFace] //list of faces which must
   not be selected for forwarding
4: nrTriedFaces = 0
5: nrFaces = flowTable[prefix].size()
6: while nrTriedFaces < nrFaces do
7:   if reliableFaceAvailable() then
8:     face = GetCheapestReliableFace
       (prefix, excl)
9:   else
10:    face = GetRandomFace (prefix, excl)
11:  end if
12:  if face != null & face.bandwidthTokens > 0
     then
13:    ForwardInterest(face)
14:    return
15:  else if face != null then
16:    excl.add(face)
17:  else if face == null then
18:    reliableAvailable = false
19:  end if
20:  nrTriedFaces++
21: end while
22: SendNack(inFace)
23: return
```

ceiving rules, we further extend the SDN principle such that routers are allowed to report prefix-based content statistics (i.e., link congestion, reliability) to the controller. The strategy we have implemented in our approach extends the flow entries received by the controller with a value indicating the reliability of each face within the flow entry.

Our mechanism of pushing rules to forwarding elements within the network is similar to the OpenFlow standard [9], which is one of the most widely used standards in the SDN area. One difference however is that OpenFlow switches strictly forward incoming packets to all ports listed in the output action part of the matching rule for the incoming packet. On the contrary, in our approach, routers can choose one or more faces among the list of possible faces for matching Interests on their own. This way, routers are able to make more informed forwarding decisions.

The face selection of our forwarding strategy is outlined in Algorithm 1.

When Interests are received, the NDN router tries to find the cheapest face based on the cost metric (in our case the hop count) whose reliability is also above a certain predefined threshold (cf. Algorithm 1, line 8). This is done by calling the function `GetCheapestReliableFace()`, which, in addition to the Interest prefix, also takes a list of faces which shall be excluded (line 3). This list initially only contains the face on which the Interest has been received to avoid loops. If such a face is found and the necessary bandwidth on this face for the received prefix is available, the router forwards the Interest via this face (lines 12-13). If no bandwidth is available, the router tries to find another face where bandwidth is available. If no valid face has been found the router also takes faces into account where the reliability is below the specified threshold (line 10). This way, a certain amount of traffic still goes through faces previously marked as unreliable in order to test if they have become more reliable in the meantime. If no face with available bandwidth has been found the router sends back a NACK to the incoming face (line 22).

The router which receives the NACK then updates its reliability value for the prefix of the sent Interest on the face where the NACK has been received. If the reliability value drops below a certain threshold, it indicates the link for this face is congested and informs the controller. The controller will then try to find an alternative route.

4. PROACTIVE CACHING

In Software Defined Networks, controllers can communicate with an application-layer component via a northbound (NB) interface [2]. This enables network operators to develop application-aware networks which can be tailored to the specific needs of the application the network is intended for. For example, a network whose main purpose is to disseminate Zipf-distributed multimedia content might benefit from different options in terms of routing and/or caching than a network which is mostly used for exchanging a large variety of small documents between clients.

In order to support the former use case we have investigated the potential benefits of proactive caching of multimedia content in order to bring popular content nearer to potential consumers *before* they start to request it. This is achieved by gathering statistical data in terms of overall network load as well as content types requested by users of certain geographical regions. This information is analysed on the application layer which tries to identify recurring network traffic patterns. Examples for recurring traffic patterns include periods where the overall network load is low and periods with high load where the content popularity distribution can be approximated by a Zipf-distribution [3]. If such patterns are found, the application commands special dedicated caching nodes to request content which is likely to become popular within their geographical region in the near future if the current network load is below a certain threshold (or if enough overall bandwidth is available). How these periods are defined is described in more detail in Section 5. The dedicated caching nodes will be referred to as SDNDN-cache in the remainder of this paper. As soon as an SDNDN-cache has finished downloading the content for the next period, it advertises itself as provider for this content to the controller, as described in Section 3. Algorithm 2 illustrates this principle in a simplified form. The periods are stored in an array where

Algorithm 2 Planning of Proactive Caching on the controller

```
1: Input: periods, caches
2: Output: Instruct caches to prefetch content if possible
3: i = 0;
4: while i < periods.length - 1 do
5:    if periods[i].load == 'low' then
6:       for region in periods[i+1].regions do
7:          if periods[i+1].contentDistribution ==
            'zipf' then
8:             caches[region.id].download
               (region.mostPopularContent)
9:          end if
10:      end for
11:   end if
12:   waitForNextPeriod()
13:   i++
14: end while
```

each entry indicates whether the overall network load is low or high. Further, each entry contains an array of statistics of each region which indicates if the content popularity is Zipf-distributed (cf. Algorithm 2, lines 4-6). At the beginning of each period, the expected network load is evaluated (line 5). If the expected load is low, the controller checks if the content popularity for the next period is Zipf-distributed (line 7). If this is the case, it instructs the SDNDN-cache in the respective region to prefetch the content which is expected to become most popular. As our aim is to investigate the benefit of proactive caching multimedia content in NDN, we assume that our application layer is already aware of the recurring patterns in the network and their time of occurrence.

The expected outcome of this approach is to move the load from the core of the network to the connected autonomous systems (AS). Further, the inherent caching in NDN aids in relieving the link which connects the SDNDN-cache to the network since a portion of the previously downloaded content will be available in the router caches within the AS. This should ultimately result in a higher overall satisfaction rate, as well as a decrease in the round-trip-time (RTT) for the Interests sent into the network.

5. EVALUATION

To evaluate the approaches described in Sections 3 and 4 we have implemented a controller-based NDN using the *ndnSIM 1.0* simulator [10]. In order to calculate routes for incoming Interests, the controller stores the network topology including link properties such as available bandwidth, delay and reliability in a *Neo4J Graph Database* [11]. In this database, network nodes are represented by graph nodes and the links between network nodes are represented by graph edges. Neo4J provides a REST interface which can be used by the controller in order to get the shortest paths based on the previously mentioned link properties (e.g., hop count, bandwidth, delay). Further, the REST interface is used to regularly update the status of the network (e.g., when a node in the network informs the controller about a detected link congestion).

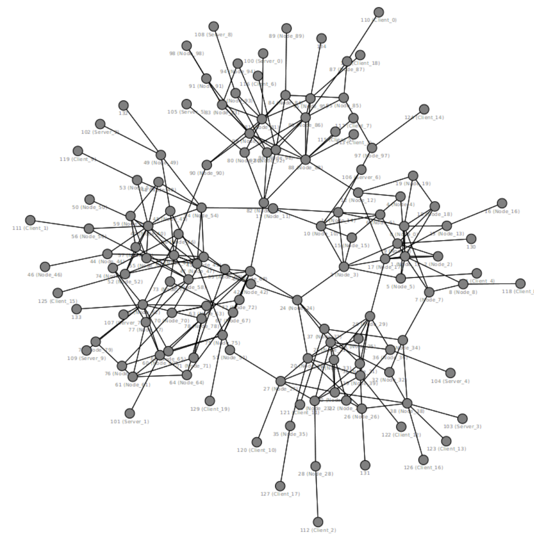
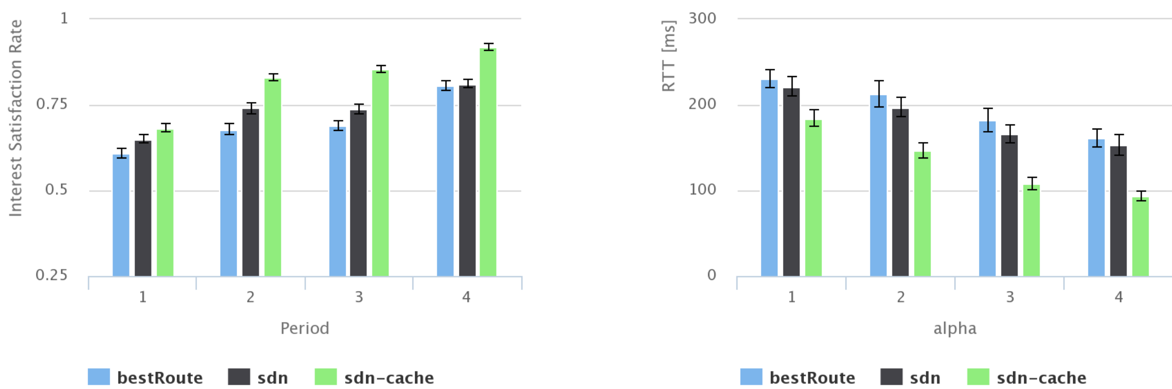


Fig. 2. Example Topology.

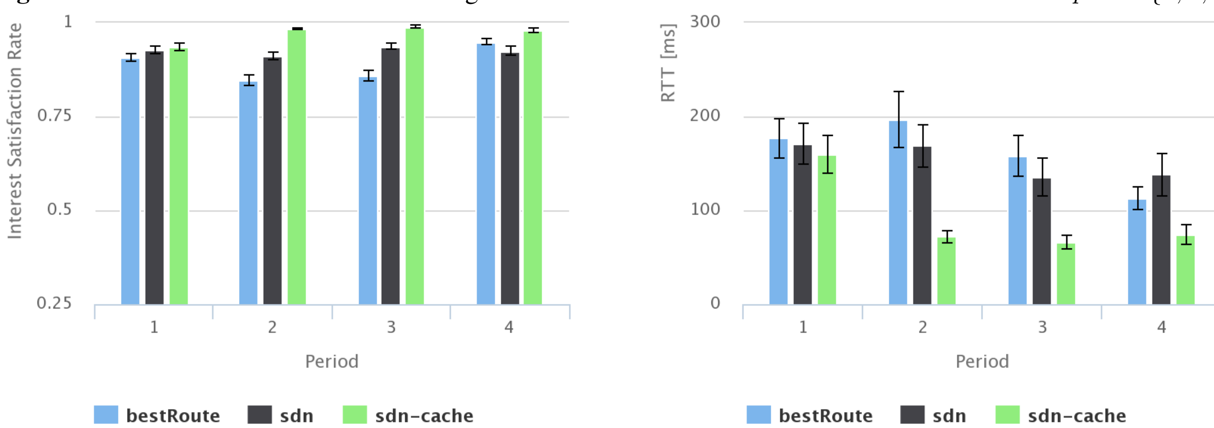
For our simulation scenarios we used the network topology generator BRITE [12]. BRITE was configured to create five autonomous systems (ASs), each consisting of 20 nodes. Among these autonomous systems we randomly distributed ten servers where each server provided different content and 20 clients. Furthermore, one SDNDN-cache was installed per AS. Figure 2 depicts an example of a topology used in the evaluation. Each simulation run was divided into four periods of 80 seconds each. Each period was further divided into a 40 seconds phase with high network load and another 40 seconds phase where the network load was low. During the high load phase the clients were configured to request content with the size of 5 MB with a constant rate of 30 Interests per second, which corresponds to a bitrate of 1 Mbps. During the low phase only the SDNDN-caches *proactively* request content. The bandwidth capacities of the links within the network were configured such that congestion was likely to occur. More specifically, the link bandwidths between the ASs were uniformly distributed between 2 and 4 Mbps. Bandwidths within an AS were uniformly distributed between 1 and 2 Mbps. The probability of selecting a certain content for each client was specified by a configuration which contains the Zipf-distributed popularity values of the contents for each period for the AS of the respective client. This simulates the behavior of having a small number of contents that are popular within a certain geographical region during a certain time of the day. During our simulations we assumed that the controller, or, more specifically, the application layer in SDN-related terminology, is already aware of these popularity values as well as when the network load is expected to be high



(a) Overall Average Interest Satisfaction Rate

(b) Overall Average RTT

Fig. 3. Simulation Results for Overall Average Interest Satisfaction Rate and RTT with 95% CI for $alpha \in \{1, 2, 3, 4\}$.



(a) Interest Satisfaction Rate per Period

(b) RTT per Period

Fig. 4. Interest Satisfaction Rate and RTT with 95% CI for each Period for $alpha = 4$.

or low. Due to this circumstance, the controller can instruct the SDNDN-caches to prefetch the content that is expected to become the most popular in their AS during the next period. In order to evaluate the performance of our controller-aided forwarding strategy we compared it to the *BestRoute* strategy provided by ndnSIM for different values of the parameter *alpha* of the Zipf-distribution. Using this strategy, each node knows all possible routes to the content servers providing content for a certain prefix, as well as the hop count for each face for every prefix. The flow tables of the BestRoute strategy are populated by a global routing helper which calculates all possible routes from each node to every content server at the beginning of the simulation. At first we evaluated how our controller-aided strategy, henceforth denoted as *sdn*, without prefetching content to SDNDN-caches compares to BestRoute in terms of overall Interest satisfaction rate and average RTT (i.e., the time between sending the Interest and receiving the corresponding data packet at the consumer). After obtaining results for this approach we investigated how our strategy combined with prefetching affects the measured parameters. The last approach will be referred to as *sdn-cache*. To investigate how the content popularity distribution affects the expected performance gain of the last approach we made simulation runs for different *alpha* values with $alpha \in \{1, 2, 3, 4\}$ for the Zipf-distribution used to specify the content popularity. The parameter *alpha* characterizes the distribution, i.e., the higher its value the fewer items claim the major part of the overall popularity. For each *alpha* value we did $N = 30$ simulation runs per strategy. Figure 3 (a) and (b) depict the results for the overall average Interest satisfaction rate and RTT, respectively. The bars in Figure 3 indicate the average values for the overall Interest satisfaction rate and RTT. The 95% confidence intervals (CI) are depicted as error bars in the charts. The results show that the *sdn-cache* yielded significantly better results than *BestRoute* in terms of Interest satisfaction rate and RTT.

The average satisfaction rate per period for $alpha = 4$ is depicted in Figure 4 (a) to illustrate the effects of our prefetching approach. The error bars indicate the 95% confidence intervals for the average Interest satisfaction rates of the respective period. In the first period, no content has been prefetched by the SDNDN-caches and thus the average satisfaction rate of *sdn-cache* is similar to *sdn*. After the first period, however, as expected, the use of proactive caching resulted in a significant improvement of the average Interest satisfaction rate.

For the Interest RTT a similar trend was observed, i.e., *sdn-cache* resulted in significantly better values than *BestRoute* after the first period. Due to space constraints, only the average Interest RTT per period for $alpha = 4$ is depicted in Figure 4 (b).

6. CONCLUSION AND FUTURE WORK

In this paper we have investigated the potential benefits of using features provided by SDN in the context of NDN, especially regarding dissemination of multimedia content. Specifically, we tried to leverage the holistic view of the network at the controller in order to quickly react to events such as link congestion detected by forwarding elements in the network and thus improve routing of Interest packets. Our evaluations of a controller-aided forwarding strategy showed promising results in terms of Interest satisfaction rate and round-trip-time, however with no significant improvement compared to the *BestRoute* strategy provided by *ndnSIM*. Ultimately, significant improvements of these metrics were obtained by using content popularity statistics known to the SDN application layer. These statistics were used to instruct special caching nodes in the networks to proactively fetch content expected to become popular in their respective geographical region in the near future.

In this paper we assumed that the content popularity statistics are already known to the Application Layer. In our future work we will try to derive these statistics on-line during our simulation runs by applying data mining and machine learning methods. Finally, we will investigate the traffic overhead caused by the communication between forwarding elements and the controller.

7. ACKNOWLEDGMENTS

This work was partly funded by the Austrian Science Fund (FWF) under the CHIST-ERA project CONCERT (A Context-Adaptive Content Ecosystem Under Uncertainty), project number *I1402*.

8. REFERENCES

[1] 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*, New York, NY, USA, 2009, CoNEXT '09, pp. 1–12, ACM.

[2] M. Jarschel, T. Zinner, T. Hossfeld, P. Tran-Gia, and W. Kellerer, "Interfaces, Attributes, and Use Cases: A Compass for SDN," *IEEE Communications Magazine*, vol. 52, no. 6, pp. 210–217, June 2014.

[3] M. Cha, H. Kwak, P. Rodriguez, Y. Ahn, and Sue Moon, "I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System," in *Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement, IMC '07*, 2007, pp. 1–14.

[4] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, D. Massey, C. Papadopoulos, et al., "Named Data Detworking (NDN) Project," *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[5] J Torres, L Ferraz, and O Duarte, "Controller-based routing scheme for Named Data Network," *Electrical Engineering Program, COPPE/UFRJ, Tech. Rep*, 2012.

[6] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling Information Centric Networking in IP Networks Using SDN," in *2013 IEEE SDN for Future Networks and Services (SDN4FNS)*, Nov 2013, pp. 1–6.

[7] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, "Supporting Information-centric Functionality in Software Defined Networks," in *2012 IEEE International Conference on Communications (ICC)*, June 2012, pp. 6645–6650.

[8] N.B. Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, "An OpenFlow-based Testbed for Information Centric Networking," in *Future Network Mobile Summit (FutureNetw), 2012*, July 2012, pp. 1–9.

[9] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and Jonathan Turner, "OpenFlow: Enabling Innovation in Campus Networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, March 2008.

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

[11] "Neo4J, the World's Leading Graph Database," neo4j.org, Accessed: 2015-18-03.

[12] Alberto Medina, Ibrahim Matta, and John Byers, "BRITE: a Flexible Generator of Internet Topologies," http://www.cs.bu.edu/brite/, 2000, Accessed: 2015-31-03.