Enabling Information Centric Networking in IP Networks Using SDN

Markus Vahlenkamp, Fabian Schneider, Dirk Kutscher, Jan Seedorf
NEC Laboratories Europe, Heidelberg, Germany
<first.last>@neclab.eu

***Abstract*—In this paper, we show how to enable Information-Centric Networking (ICN) on existing IP networks, such as ISP or data center networks, using Software-Defined Networking (SDN) functions and control. We describe a mechanism that *(i)* enables addressing and transfer through non-SDN controlled networks (i. e., the Internet), *(ii)* allows to identify ICN requests and responses, *(iii)* decouples forwarding from the object name, *(iv)* requires neither new or extended network/L3 and transport/L4 protocols nor changes of client and server OS, and *(v)* supports aggregation of routes inside the SDN controlled network. In addition, the proposed solution is agnostic of the specific ICN protocol in use, and does not require all network elements to be SDN-enabled. It supports advanced ICN routing features like request aggregation and forking, as well as load-balancing, traffic engineering, and explicit path steering (e. g., through ICN caches). We present the design as well as our first implementation of the proposed scheme—based on the Trema OpenFlow controller-framework and CCNx—along with initial performance measurements showing the feasibility of our approach.**

I. Introduction

Software-Defined Networking (SDN) is based on the concept of providing an API for packet forwarding devices such as switches and routers, which allows programmability of network elements and entire networks. In this paper, we apply the general SDN concept for a new type of networking: Information-Centric-Networking (ICN) [1]. In ICN, communication is not based on packets that are "sent from" and "destined to" hosts or host interface addresses. Instead, requesters send requests to the network, asking for Named Data Objects (NDOs) that have been published before and that are available in one or many copies in the network. The network elements that receive a request—unless they have a local copy in their cache—typically have to decide where to forward the request to, for example which interface to use for forwarding the request. Usually, network nodes participate in a routing protocol that helps to distribute this information. Alternatively, a network node can employ a Name Resolution Service (NRS) that can map NDOs to locators in underlying networks, for example IP. Once a request has been forwarded and reached the destination (e.g., a cache holding the copy of the NDO), a corresponding response message (i. e., the NDO itself or a locator) has to be relayed back to the requester, possibly passing one or more on-path caches that can cache the objects in order to satisfy future requests from their cache. This return path can be determined in different ways: For example network elements can maintain state or they can obtain some information from data structure inside messages, such as a label stack.

This paper addresses the problem of decoupling the ICN data plane (ICN request and NDO forwarding) from the ICN control plane, so that name-based routing and name resolution can be performed independently. In many networks where ICN becomes interesting, such as data centers, a centralized ICN control plane is viable and can take on additional functions. Examples include dynamically replicating NDOs across multiple nodes and load-balancing both ICN nodes and network resources.

The mechanisms for software-defined ICN that are described in this paper enable control plane entities ("controllers") to implement arbitrary control protocols for deciding about request/response forwarding. Software-Defined-Network elements (e.g. OpenFlow switches) can be kept relatively simple, focusing on efficient request forwarding. In particular, these data plane elements do not need to be ICN-enabled with the approach we present. A controller, for example, could be programmed to decide about the next hop for every message that a network element has no information for. It could also participate in large scale name resolution or name-based routing and provide corresponding hints to network elements. As a result of this approach, the ICN network can be dynamically adapted. Network elements can be kept simple and do not need to know about ICN routing or name resolution.

In the following subsections we give an overview on ICN deployment approaches, remaining challenges and the benefits of our solution. In Section II we explain our solution, explain its components, and detail on how to process ICN messages. In Section III we explain our first implementation of the approach using the Trema SDN framework and CCNx, and present preliminary evaluation of our implementation. We conclude and outline our next steps in Section IV.

A. Existing Approaches to Realize ICN

An ideal or native deployment of ICN requires all network elements, user devices, content sources as well as all intermediary network elements (routers, switches, ...) to be ICN aware. We do not believe that in the near future such an ICN deployment is viable given the need to exchange or at least update all networking equipment installed today. Therefore in this paper we aim for a step-by-step migration towards more and more part of the network being ICN enabled.

Running ICN approaches on the existing Internet—which is tailored to support host-to-host communication using IP and TCP/UDP—requires additional mechanisms. The most common approach is to create an overlay network on top of the existing Internet and implement the additional routing and

forwarding mechanisms in the overlay software. In particular overlays cause overhead for overlay management and encapsulation inside Internet protocols. The overlay approach and its overhead has been well explored in the context of P2P systems.

A slightly different solution has been presented by Blefari-Melazzi et al. [2], which aims in a similar direction as our approach. Their approach is based on OpenFlow switches and dedicated border nodes which perform name-to-location resolution (with the help of an external system) for the requested NDO. The proposal defines a new IP option which is supposed to include the requested NDOs name and an ICN specific transport protocol. Because the new IP option cannot be matched upon by OpenFlow, the border nodes encapsulate the original request in a new packet. A flow identifier (which represents the object's name with a 1:1 mapping) is embedded within the transport protocol's port numbers. Then the OpenFlow controlled network can use these port numbers to forward the packet to the appropriate location(s). The border node will keep a pending interest table to allow reversing the encapsulation and sending the response to the original requester.

B. Remaining Challenges and Problems

While aiming in a promising direction (i. e., moving away from overlay networks for deploying ICN), the approach of Blefari-Melazzi et al. leaves a set of problems and challenges:

- How does a client get to know the identity (and address) of a border node from outside the network, or where would it send its requests initially?
- The border node needs to be ICN-aware, run ICN software, and perform packet en/decapsulation. While this is an easy task for custom software running on a computer, this is difficult on today's network elements and at high network speeds. Moreover operators are reluctant to put "middle boxes" in there network, rendering this ICN-aware border node option unlikely to be widely adapted.
- By using IP options, packets will typically pass through the slow-path in network elements (i. e., they will be processed by the network element's GPU rather than the optimized network processing pipeline).
- The approach requires a new transport protocol which might be blocked at various network elements (esp. those outside administrative control).
- ICN nodes (both requester and caches/servers) cannot operate on default UDP (or TCP) sockets, thus requiring special kernel modules.
- ICN nodes need to know how to communicate with the name resolution service.

The approach we present in this paper solves all the problems above, while at the same time preserving the benefits of existing solutions, e. g., close(r) to native—all network elements are ICN aware—ICN deployment and support for aggregation of destinations/routes/rules so that forwarding tables remain manageable.

C. Benefits of our Solution

Our SDN backed ICN deployment approach seeks to provide the following benefits: *(i)* Facilitate ICN deployment over existing networks. We particularly focus on initial and partial ICN deployments that can emerge with time. We achieve this by allowing for the use of off-the-shelf network protocols and network stacks in host operating systems (i. e., unmodified IP and UDP) and enabling ICN hosts to send requests to that ICN network without any knowledge about cache locations and ICN node addresses. *(ii)* Routing or content location learning, which is a major problem in ICN solutions, is simplified through separating routing and forwarding through an SDN approach. The centralized view of the controller enables easier path selection/finding and more sophisticated forwarding decisions. Moreover, state maintenance and complexity requirements on network elements can be kept low and actually moved between SDN switches and controllers. *(iii)* Routing can be further improved through knowledge about both interests/requests and publications/content locations. Multiple requests can be aggregated into one or split across multiple paths. The controller can also manage cache content by triggering the pre-population/synchronization of caches or by routing contents through caches. The controller can also consider network utilization in the forwarding decisions thus achieving traffic engineering.

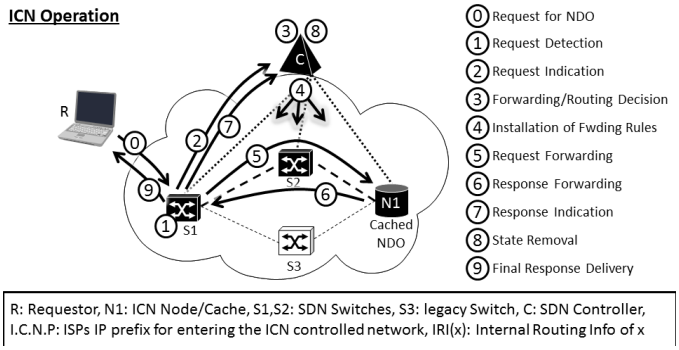

Fig. 1. Overall Operation

II. Proposed Approach

The generic method of our approach is to use SDN mechanisms to *(i)* detect ICN requests in a network, and to *(ii)* set up forwarding and return paths for ICN request/response message pairs. The general idea of our approach is to include a (message) identifier inside the default packet headers of network (IP) and transport (UDP or TCP) protocols while traversing the SDN controlled part of a network. This SDN controlled part should include most of the ICN nodes in a network. Then the identifier can be matched upon to determine forwarding paths through the SDN network[1], thus enabling close to native ICN forwarding. In order to direct packets to this ICN, we propose an IP prefix to which user devices should address their request packets. The presence of this prefix is also used to differentiate rewritten (inside SDN control) from original (outside SDN control) packets. Because the path setup inside the SDN network is determined by an ICN-aware SDN controller all of the typical ICN benefits can be realized without the need for ICN-aware network elements.

A. Overall SDN controlled ICN operation

Figure 1 depicts the overall operation: A network consists of ICN requesters (R), network elements (S1 to S3) and one or more object caches that hold copies of NDOs. The network is SDN-enabled and controlled by at least one SDN controller (C). An ICN-enabled requester R sends an ICN request for an NDO to an SDN network (step 0). A network element (S1) in that SDN network identifies (based on earlier configuration provided by an SDN controller) the request as an ICN request (step 1). The network element sends a request indication (containing the request packet) to the SDN controller (step 2). The SDN controller is expected to have ICN routing information and can identify one or more destinations for the request (forwarding decision step 3). The controller uses SDN control mechanisms to install forwarding rules in the network (step 4). These forwarding rules enable network elements to forward the request to the selected cache (step 5) and to forward an eventual response message back to R. Once the request has reached the Object Cache, it generates a response message with the requested object. This response message is forwarded according to the previously installed forwarding state back to S1 (step 6). S1 indicates the response message reception to the controller (step 7) which can then remove any state about the request (step 8). Finally the message is delivered to the original requester (step 9).

ICN protocols generally enable access to NDOs in network. Such protocols typically provide different message types such as a GET request (called ICN request above) and a corresponding response message. Parameters of a GET request include the name of the requested NDO, and often, as for the specific NetInf ICN protocol [3], a message identifier (Message ID) to distinguish messages in the network. However, our approach does not enforce a specific name structure.

We assume an ICN protocol on top of the Internet Protocol, specifically we assume that ICN messages are transferred with UDP and TCP. Yet, a dedicated ICN transport protocol would also work with our approach. In addition, we consider the ICN protocol messages to be read-only. However, the SDN controller needs to be able to understand the ICN application protocol. In particular, it needs to recognize different message types, the NDO's name, and the Message ID. The generic method is to enable an SDN controller to install appropriate forwarding state for an ICN request in a way so that network elements only have to support IP forwarding and do not require ICN protocol knowledge. This method is leveraging ICN protocol Message IDs and features of SDN instantiations such as OpenFlow [4, 5] to rewrite packet header information. The details of the method are explained in the following subsections.

B. Targeted Scenario & Conventions

Our solution focuses on the operation within a controlled IP network domain. However, the solution is as well applicable for a multi-domain scenario. In general we assume that the requester is outside the controlled network domain. The solution supports partial SDN deployment within the network domain. For example, assume an Internet Service Provider (ISP) that wants to deploy ICN caches (within domain) for its customers (outside domain). We begin by assuming UDP as transport protocol for ICN and explain the basic procedures. Later we discuss how TCP can be used for that purpose. For our solution we require:

1) An *ICN Protocol Identifier* that is carried by all packets belonging to ICN messages. A key requirement for this is that SDN network elements can match on this identifier. We propose to use a dedicated transport protocol port number, which is used as destination or source port depending on the message type (request or response). Another option would be a dedicated transport protocol which can be matched in the respective IP header field.
2) A *publicly routable network address* per domain. The purpose of this address is to allow for an easy mechanism to determine a target IP address from outside the controlled network domain. This can also be used to identify packets that have not been changed for SDN forwarding. We propose to select a specific IP prefix (can be a /32 for v4 or a /128 for v6) from the network domain's IP address pool. This could be announced on every SDN enabled network element and made public via e. g., DNS. Note, that the network domains IP routing protocol(s) should be used to distribute this prefix towards the edge of the administrative domain. This explicitly allows to announce this one prefix at multiple locations in the network, thereby accounting for geographically distributed entry points to the ICN/SDN controlled part of the network. We further assume that each ICN user device only needs the ICN IP prefix of the network operator it is currently connected to.
3) The *object's name*, which is used to determine the path of the packets (routing).
4) A *Message ID*, which will be used for forwarding decisions in the network elements on the determined path. The Message IDs will replace destination IP addresses in requests and source IP addresses in responses.
5) The SDN needs a *controller*, which has *knowledge about the location of NDOs* inside the network domain. And it needs to be able to *setup forwarding rules on all SDN network elements* in the network domain.
6) The SDN network elements need to support NAT-like IP address and port rewriting.

While 1, 3, and 4 (having a Message ID) are common to most ICN solutions and 5 (routing logic) is outside the scope of this paper, 2, 6, and 4 (encoding the Message ID in the address fields of the IP header) is a new contribution of our work. Additional contributions are the request/response processing procedures described in the following.

C. Request forwarding

Figure 2 shows the processing of ICN requests using the six requirements from the last section. When a Requester R wants to query an object from the ICN service it will first perform a lookup for the publicly routable network address specifying the ICN prefix (I.C.N.P) of the respective domain. Once the prefix has been determined an IP packet is sent (Step

[1] Strictly speaking it is rather a "path identifier" than a message ID.

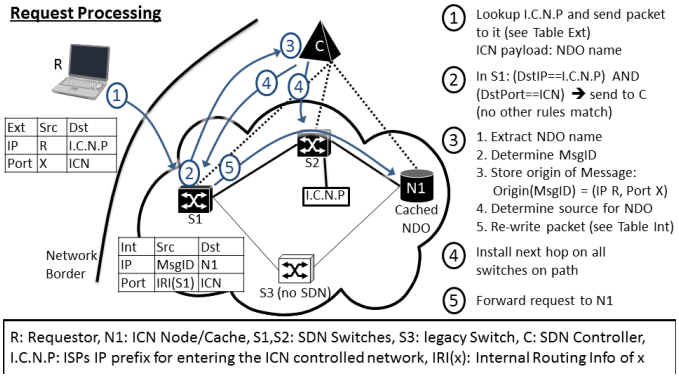

Fig. 2. Request forwarding

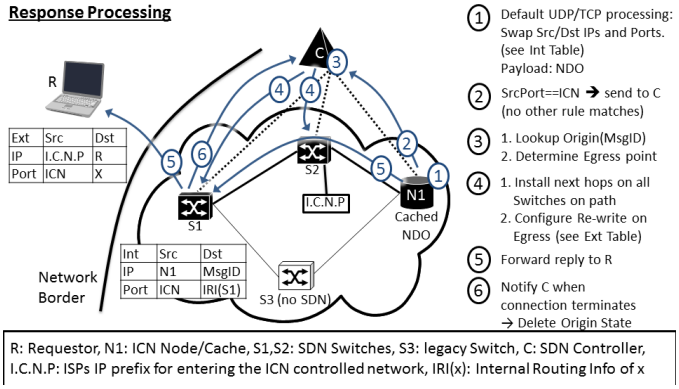

Fig. 3. Response processing

1) with destination IP address of I.C.N.P. The destination port number will be set to the ICN Protocol Identifier. As usual, the source IP address and port number will be determined by the Requester's IP stack. For this example we chose R to represent the requester's IP address and X to represent the local port number at R (see also Table 'Ext' in Figure 2). As application payload R will follow the specification of the ICN protocol which includes at least the name of the requested object.

I.C.N.P must only be announced by SDN-enabled network elements. This will ensure that IP routing sooner or later delivers the packet to S1, the "ingress" SDN network element. S1 will try to match (Step 2) the packet to its rules and find that only the "default ICN" rule matches. This rule matches for the combination of destination IP and destination port and sends packets to the Controller C if they are I.C.N.P and ICN. Upon reception of the packet by the Controller C (Step 3), C will parse the ICN protocol payload and extract the requested object name. If the ICN protocol does not use 32bit Message IDs a new random (and currently not used) Message ID is generated. The origin of the request (that is IP address R and port number X) are stored. Next the location of the NDO, that is the address of a cache that can serve the requested object, is determined. Then comparable to NAT the IPs and Port numbers of the packet are rewritten (see also Table 'Int' in Figure 2): The new destination IP is the IP of the cache from which the object should be served. The new source IP is the MsgID. The destination port number is kept to identify the packet as ICN. In addition the source port number is changed, in order to identify the "ingress" network element. This internal routing info (IRI) can be used to aggregate rules (read routes) for response packets. Next (Step 4) C installs forwarding rules (read routes) on all the SDN network elements on the path to N1 and sends the re-written packet back to S1. All these rules must include a check if the port number matches ICN. The rules can either use the MsgID as match criteria, or can be aggregated using the destination IP addresses. Eventually (Step 5), the packet is forwarded to N1, the cache serving the object. Using the IP address of N1 as destination IP one could even use non-SDN paths (e.g. S1→ S3→ N1) towards the cache.

D. Response forwarding

Figure 3 shows the processing of ICN responses. The ICN node N1 does not need a special network stack. It can simple

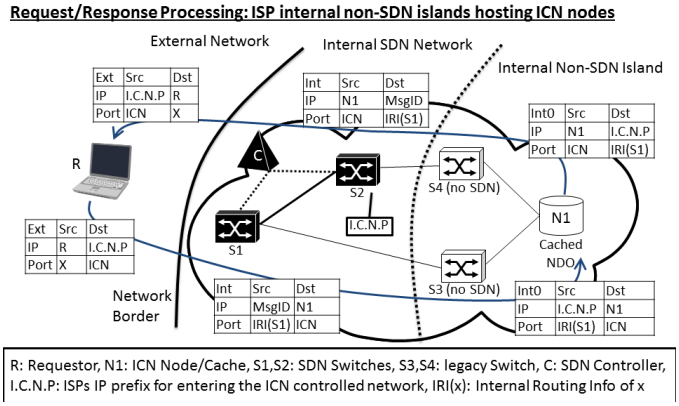

Fig. 4. Processing with non-SDN islands

listen to the ICN port. When requests arrive they will be processed and a corresponding response (positive or negative) will be generated. As with usual UDP and TCP processing for the response packet(s), the IPs and port numbers are swapped (Step 1, see Table 'Int' in Figure 3). Now two options exist: 1.) N1 has a build-in SDN stack or 2.) N1 is configured to send all ICN traffic out on a port connected to an SDN controlled network element. Either way (Step 2), the first SDN enabled element will forward the first response packet to the controller. This can be achieved by matching for ICN in the source port. However the controller might already have installed a specific (based on the MsgID) or aggregated (based on the IRI) rule to forward responses; in that case the packet(s) are not sent to the controller. When the controller receives a response packet (Step 3) it will lookup the origin of the MsgID of that packet and determine a forwarding path back to the requester. This path and the reversal of the address and port re-writing rules will be installed (Step 4). Then (Step 5) the network forwards the packet(s) to the origin, performing the address and port re-write on the egress SDN node. When the response has been completely delivered to R the egress SDN node will notify C that all state for MsgID can be removed (Step 6). Note that while request forwarding is possible over non-SDN controlled network elements, the response forwarding does not work as it is using the MsgID as destination IP addresses.

E. Other Aspects and Benefits

Non-SDN-controlled network parts: When ICN nodes are deployed in non-SDN controlled islands, communication is still possible. Figure 4 shows an example. Since we cannot use the MsgIDs as destination IPs for the response packet(s), we use addresses from the already deployed I.C.N.P prefix. However, as we want to support existing network stacks, which only swap IPs and port numbers, we also need to change the request. Thus a solution is a second re-write operation before forwarding the requests to the non-SDN controlled part of the network.

Dealing with failed requests / Cache does not have the NDO: If the cache that is supposed to deliver the NDO does not have the NDO it cannot fulfill the request. This can happen either because the controller did not have complete information, or because the cache evicted the content to cache other objects. There are two options to deal with this situation. First, the cache can reply with a ICN message indicating the content is not available. Second, the cache can "forward" the request. The handling for the second case is shown in Figure 4.

ICN Request Aggregation and Request Forking: Request Aggregation in ICN is the concept of avoiding to send two requests for the same NDO through the whole network. If some part of the path towards the cache is shared, the ICN approach is to transmit the request and NDO only once over the shared path. Realizing this with an SDN controlled network is easy: When a second request for the same object arrives before the response of the first one was initiated, the response path for the NDO can be setup to include copying the response to both ingress network elements. Request Forking is the opposite of request aggregation. The ingress node will duplicate the request and send it out to multiple caches. Once the first cache generates a response, the other request can be ignored. With our approach, the SDN controller will generate multiple copies of the request and target them to different ICN nodes/caches.

F. State Management Trade-offs

The proposed solution does allow to trade-off state complexity in the network elements for complexity in the controller. That way the approach can be customized towards the capabilities of the deployed network and the degree of ICN utilization. To reduce the state in SDN network elements, SDN rules can be aggregated by destination. It is not necessary to install rules per MsgID. For the request direction default IP route aggregation is possible because the destination IP is a routable address. For the response direction the internal routing information (IRI) that is embedded in one of the port numbers can be used. An alternative possibility is to send all ICN packets to the controller and let it make all decisions. Rules do not need to be installed.

Likewise it is also possible to reduce the state in the controller by doing one of the following: 1) The response path can already be configured when the request is processed; 2) The address re-writing at egress can be pre-configured. No request origin information on the controller; 3) In combination with SDN rule inactivity timers the whole processing can be performed in a "fire-and-forget" manner from the Controller's perspective.

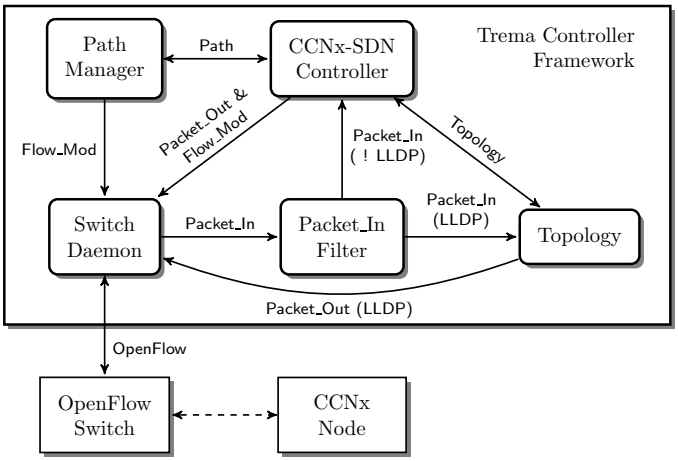

Fig. 5. ICN-SDN implementation architecture

III. Initial Implementation & Evaluation

In this Section we first provide an overview of our prototype implementation, called *CCNx-SDN-Controller*, which is based on CCNx [6] and Trema [7]. Then we briefly report on our initial evaluation of the approach.

CCNx is PARC's open source implementation of their "Content Centric Networking (CCN)" [8] ICN approach. Trema is an open-source OpenFlow controller framework developed by NEC, which allow to easily implement arbitrary network control applications.

A. CCNx-SDN-Controller Implementation

Our Controller implementation consists of different parts, as depicted in Figure 5. For each controlled switch (called Datapath in the OpenFlow terminology), the Trema *Switch Manager* (not depicted) forks a *Switch Daemon*, that is responsible for the communication with its associated *OpenFlow Switch*. Each packet yielded by the *OpenFlow Switches* is, via the *Switch Daemon*, first delivered to the *Packet_In Filter*, whose task is to filter out Link Layer Discovery Protocol (LLDP) packets and deliver them to the *Topology* component, which is likewise generating and receiving those LLDP packets for the purpose of detecting the network topology. According to our *Packet_In Filter* configuration, all other packets are handed over to the *CCNx-SDN-Controller* process.

The *CCNx-SDN-Controller* maintains a *FIB* to manage the object name to ICN node address mappings. It learns this information from observing name prefix announcement traffic of the controlled *CCNx nodes*. Moreover, it processes the incoming packets and further sends information back to the *OpenFlow Switch*, via Packet_Out messages, which are handed over to the actual *Switch Daemon*. The *CCNx-SDN-Controller* further utilizes the *Topology* component to calculate paths through the network. The *Topology* component, when queried with ingress and egress Datapath and there associated ports, provides a list of Datapath_IDs and there associated ingress and egress ports, that form the requested path. The actual path provisioning is subsequently partially delegated to the *Path Manager*, that consistently handles the path creation. It is just asked to provision the basic forwarding path, whereas the switch, that has to perform the packet rewriting actions is instructed by the *CCNx-SDN-Controller* itself. The flow entries

provisioned this way are created without a hard lifetime, but an idle timeout of 10 seconds. A side effect of this mechanism of deploying the path, is, that the *CCNx-SDN-Controller* will explicitly be notified by the *Switch Daemon* whenever its flow entries are removed.

ICN Request Path Provisioning Based on the source IP and port information the controller generates a MsgID. This MsgID along with FIB information is used to program the packet header rewriting according to our approach. We pre-provision the path to the content source on all intermediate switches using the *Path Manager* and send the Packet_Out message to the ingress switch.

Response Path Provisioning The backward path for content delivery can also be pre-provisioned upon request processing. However to allow finer grain control over the ICN operation, the response path can be explicitly provisioned as well. For this we reverse the tasks from the request provisioning, i. e., , first setup the path to the egress, and then programm the header re-writing.

B. Evaluation

Our prototype implementation shows that the benefits discussed in Section I-C can indeed be achieved and that our approach is viable and solves the challenges outlined in Section I-B. We do not use IP options or new/modified transport protocols, and thus we can implement our solution on existing operating systems. Also our solution does not require complex communication between ICN nodes and a name resolution service, as our ingress nodes can be simple SDN switches. Thus we avoid the placement of yet another set of middleboxes.

To further analyze the applicability of our concept, as well as to get a first impression of the effects on network performance, we set up an emulated network environment to run experiments with our ICN-SDN approach in comparison to a simple direct communication and overlay configuration. We find that our *CCNx-SDN-Controller* produced transfer times similar to those of a best case non-SDN scenario (in which the content consumer is already aware of the actual node that serves the requested content). Hence no additional hops are to be traversed.

We do not assume that this will be the common case. Very likely additional hops that perform CCNx routing decisions or name-to-location resolution will be required to reach a content serving node. So typically an ICN deployed as an overlay will correspond to a multi-hop ICN communication. We also ran experiments for such a scenario with 2 hops, which is representative of the ICN edge node design of Blefari-Melazzi et al. [2]. In such a scenario the transfer time increases by one third. Thus, our SDN approach provides even better performance in addition to the deployability benefits outlined above.

Our results are very initial and we plan to follow up with a broader evaluation and refinement of the approach.

IV. Conclusion & Future Work

In this paper we have introduced a mechanism to deploy ICN protocols in IP networks via the assistance of the SDN paradigm. Our approach utilizes smart packet header rewriting in combination with a single IP prefix to initially contact the ICN network. This approach is backed by the centralized SDN controller, that is used to generate paths through the SDN controlled network domain. The benefits include ease of deployment in existing IP networks, not requiring additional transport protocols or extensions to the IP protocol, and separation of the ICN data and control plane. The latter enables simpler ICN routing mechanism through centralized knowledge as well as improved traffic engineering capabilities. Furthermore our approach allows trading the need for keeping state in network elements for reduced interactions between the controller and network elements.

Through an implementation and subsequent evaluation, we have demonstrated the practicability and benefits of our approach. Our preliminary results show that compared to a typical overlay deployment we can significantly shorten the transfer times, and achieve almost optimal performance. As future work we intend to advance our general approach further, extend the evaluation, and apply it to different ICN instantiations.

Acknowledgment

This work has been supported by the GreenICN project (GreenICN: Architecture and Applications of Green Information Centric Networking), a research project supported jointly by the European Commission under its 7th Framework Program (contract no. 608518) and the National Institute of Information and Communications Technology (NICT) in Japan (contract no. 167). The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the GreenICN project, the European Commission, or NICT.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 26–36, 2012.

[2] N. Blefari-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*, 2012, pp. 1–9.

[3] D. Kutscher, S. Farrell, and E. Davies, "The NetInf Protocol," draft-kutscher-icnrg-netinf-proto-01, Internet Research Task Force, Aug. 2013.

[4] Open Networking Foundation, "OpenFlow Switch Errata, Version 1.0.1," 2012, https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.1.pdf.

[5] OpenFlow project at Stanford, "OpenFlow Switch Specification, Version 1.0.0," 2009, http://www.openflow.org/documents/openflow-spec-v1.0.0.pdf.

[6] "CCNx OpenSource project," 2013, http://www.ccnx.org/.

[7] "Trema project," 2013, http://trema.github.io/trema/.

[8] 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*, ser. CoNEXT '09, 2009, pp. 1–12.