Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Design and Implementation of a Software-Defined Approach to Information-Centric Networking

Master Thesis

Markus Vahlenkamp

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Markus Vahlenkamp

Design and Implementation of a Software-Defined Approach to Information-Centric Networking

Master Thesis eingereicht im Rahmen der Masterprüfung
im Studiengang Master Informatik
am Studiendepartment Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Professor: Prof. Dr. Thomas C. Schmidt
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Abgegeben am 2. Dezember 2013

Hochschule für Angewandte Wissenschaften Hamburg
Fachbereich Technik und Informatik
Zusammenfassungsblatt zur Master Thesis

Markus Vahlenkamp

Title of the Master Thesis / Titel der Master Thesis
Design and Implementation of a Software-Defined Approach to Information-Centric Networking

Keywords / Stichworte
Future Internet, Information-Centric Networking (ICN), Content-Centric Networking (CCN), Software-Defined Networking (SDN)

Abstract
This work is concerned with the integration of the two networking paradigms of ICN and SDN in order to provide an enhanced evolutionary path from today's IP networks towards ICN. We introduce a scheme that allows for the operation of SDN based infrastructure, which provides network-wide ICN awareness. Further, an advanced approach, which allows for the forking of ICN packets throughout the SDN domain is developed and evaluated. This approach introduces the ability of aggregating requests and a mechanism to populate off-path caches as well as the parallel requesting of multiple caches. The qualitative advances that the SDN supported deployment provides compared to a pure overlay deployment are investigated plus the quantitative results of our Implementation.

Kurzzusammenfassung
Die vorliegende Arbeit beschäftigt sich mit der Integration der beiden Netzwerkparadigmen ICN und SDN. Dabei wird das Ziel verfolgt, einen evolutionäre Migrationspfad von heutigen IP Netzwerken hin zu ICN zu bieten. Wir zeigen ein Schema auf, dass die initiale Bereitstellung eine SDN gestützten Netzwerks mit erweiterter ICN Verarbeitung ermöglicht. Darüber hinaus wird ein erweiterten Ansatz entwickelt und evaluiert, der die SDN interne Duplizierung von ICN Paketen erlaubt. Hierdurch wird neben der Zusammenführung von Inhalteanfragen und Mechanismen zur Befüllung von nicht auf dem direkten Übertragungsweg befindlichen Caches auch die parallele Anfrage mehrerer Caches ermöglicht. Die qualitativen Vorteile eines SDN gestützten, gegenüber einem reinen Overlay-Betrieb werden ebenso beleuchtet, wie quantitativen Ergebnisse unserer Implementierung.

In cooperation with:

NEC Laboratories Europe
Kurfürsten-Anlage 36
69115 Heidelberg
Germany
www.netlab.nec.de

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The Internet usage is constantly rising. More and more users gain access to this global network in recent times, not only via traditional computers but also via rapidly spreading tablet computers and smart phones. The main usage of the Internet is thereby steadily shifting from mostly host-centric communication towards the repetitive distribution of pictures, music and videos – in general all sorts of static content. This host-centric Internet of today needs to specifically support the distribution of the vast amounts of traffic generated by major players and their response time requirements. Additions like Content Delivery Networks (CDNs), load-balancers and the like have been applied atop the current Internet infrastructure layers to fix these shortcomings.

However, rising interest is generated by the Information-Centric Networking (ICN) paradigm, which differs from today's Internet in various aspects. ICN introduces content awareness into the network. The network maintains information about content and the location that the information can be acquired from. Further it builds up on the request / response paradigm, thus the communication is driven by the content receiver. Instead of requesting the network to deliver information to a particular entity in the network, the request for content is handed over to the network, which is subsequently responsible for finding suitable sources for the desired content, as well as forwarding it to the requester. Sources are further not only origin but also cache nodes, which serve as an inherent component of the network infrastructure.

All of today's ICN protocols support their operation atop of today's Internet infrastructure. They mostly support the use of Transmission Control Protocol (TCP)/Internet Protocol (IP) as a convergence layer, which allows ICN enabled parties to form overlay networks on top of the Internet. However, the modification in the basic operation principals requires the network infrastructure to support this new paradigm to exploit its full potential. The routing system has to know where the requested content can be acquired from. Further the forwarding elements need to provide storage for the purpose of caching the forwarded content.

This work presents a scheme for the initial deployment of ICN protocols via Software-

Defined Networking (SDN) mechanisms. This will be achieved by providing a generic, not ICN protocol specific way of enabling ICN-awareness throughout the network, without requiring any changes to the end-hosts or the SDN forwarding elements. The SDN capable infrastructure is utilized to support enhanced ICN traffic forwarding.

The remainder of this document is structured as follows. Chapter 2 is divided into three main parts. The first subsection introduces the general ICN concept. Afterwards the Named Data Networking (NDN) scheme along with its actual implementation termed CCNx is introduced in further detail. The second subsection deals with SDN. It also starts with a general overview and continues with a specific elaboration of the OpenFlow SDN protocol. In Chapter 2.3 we give an introduction to the work of related projects and papers that also deal with the integration of ICN and SDN.

Chapter 3 details the explicit problems we deal with throughout this work. First of all the state management of ICN is discussed in Section 3.1. Subsequently security threads (Section 3.2) and scalability issues (Section 3.3) are addressed. The challenges faced when starting to deploy ICN today are listed in Section 3.4 before we detail the challenges that apply to the deployment of ICN over SDN in further detail in Section 3.5.

Chapter 4 introduces the concept of this work. The actual focus and goals are refined in greater detail in Section 4.1. We introduce our basic approach in Section 4.2. This approach as a first step leaves open some of the requirements introduced in Section 4.1. Hence, in Section 4.3 we introduce and further enhance the basic to the advanced approach that allows for ICN request and response forking operations. Additionally this section comprises an analytical estimation of the complexity of our forking approach. Section 4.4 finally covers additional advances that are applicable to both, the basic as well as the advanced approach.

Chapter 5 gives detailed information on the implementation of the concept delineated in the previous section. First the CCNx specialties in accordance to our general ICN over SDN approach are emphasized. This is followed by the overview of the ICN-SDN controller architecture. Subsequently the mode of operation is specified. The measures required to perform the forwarding rule provisioning are detailed. The two last sections finally describe the missing information, to finally put the running system into place. It is illustrated how CCNx client nodes have to be configured and a basic mechanism to register ICN name prefixes with the ICN-SDN controller is shown.

Chapter 6 provides information about the evaluation we performed. Section 6.1 gives detailed information about the hardware and software components used throughout the measurements. Following, the measurement setup is delineated in Section 6.2. This includes a description of the topologies used, the data that is collected, how the environment is initialized and the procedure description of how the measurements are conducted. The corresponding results are then summarized and discussed in Section 6.4.

Finally the entire work is summarized in Chapter 7 along with an outlook on future work and research directions.

Chapter 2

Related Work

In Section 2.1 we will give an overview of the general concepts that ICN is composed of. Subsequent the concrete ICN implementation of CCNx, which relies on the NDN scheme is introduced. Furthermore, Section 2.2 starts of with an introduction to the general building blocks of SDN, before providing a detailed insight to the SDN approach of OpenFlow. Finally this chapter closes with Section 2.3, the presentation of different projects that already provide ideas and actual work on the integration of ICN and SDN.

2.1 Information-Centric Networking

ICN is a paradigm to computer networks that is focused on the dissemination of content objects. Therefore, the network itself is enriched with knowledge about the content that it is supposed to transfer. This contradicts the mechanics of today's Internet, which has only the notion of IP addresses and thus only the capability to address hosts. This mutation is thus a fundamental paradigm shift.

Different ICN proposals have been developed in the past, all implementing the general idea of ICN. Among them are for instance TRIAD [1], DONA [2], NDN [3, 4], PSIRP / LIPSIN [5, 6] and NetInf [7]. They all take slightly different approaches in one or the other design choice, but aim for the general ICN goals such as content caching and location independent naming as described in [8, 9].

In the following we will first introduce the general concept and give an overview of ICN before presenting the NDN approach, which is the most popular one. Further we explain CCNx, the prototype implementation of NDN, which we chose as a representative of an ICN approach throughout the evaluation of our approach.[1]

[1] This ICN section originates in most parts from our previous work [8].

2.1.1 Concept / Overview

Each node participating in an ICN explicitly addresses content, instead of using host identifiers of content sources as the primary addressing scheme. Enabled by this paradigm alteration a variety of actions can be performed, whilst other actions have to be performed by ICN intermediary and end devices. A non-exhaustive list of those actions is listed in the following. Further details can be found in [8, 9, 10].

Caching Through caching network resources can be saved. Whenever some content is delivered to a content consumer, the content is cached within the network in order to satisfy subsequent requests from a nearby replica.

This behaviour carries different implications for the content distribution. On the one hand the network and server load is reduced. The content doesn't have to be transmitted all the way from the origin server to the client. Thus the work is offloaded to the caches, holding the benefit of reduced network bandwidth utilisation as well as origin server resource savings. On the other hand the delivery properties such as transmission delays caused by for instance network congestions are positively influenced through the use of a nearby cache. The overall Quality-of-Experience (QoE) for the end user will increase.

Naming Today DNS hostnames are used to reference content. Thus CDNs manipulate DNS responses and perform HTTP redirections to steer users towards different spatial distributed replicas. All this needs to be done due to the properties of Uniform Resource Locators (URLs). URLs identify the content, but they are also used to map the identifier to the content's location within the network.

This coupling of the identifier and locator of URLs is for instance one of the reasons why consumers suffer content unreachability. Content may still be available, but resides on a different server or a path on a server and is thus no longer accessible through its previous URL.

ICN names though provide content identification without the coupling to the content's location. This property is in consequence also exploited to better support the in network caching properties of ICN [11, 12].

Security Today's network security techniques, especially when it comes to secure data distribution, mainly consist of securing the communication channel, instead of securing the data itself. SSL and TLS are used for the secure transmission of data end-to-end. This is something that is not expedient when using intermediary caches distributed all over the network. Thus some mechanism for secure data dissemination is required that supports some kind of man-in-the-middle caches spread all over the network, without violating security or privacy properties.

For instance in the existing ICN projects mechanisms for data integrity checks are popular to be coupled with the naming of content objects. They provide a mechanism called self-certifiability, in which the names of a particular object reflects the hash values of the data it refers to. This is somehow comparable to the concept of cryptographically generated IPv6 addresses [13] where also parts of the addresses are generated through the use of cryptographic hashes. The use of cryptographic hash functions provide sufficient strength to be able to proof the data integrity today. Further, RFC6920 [14] specifically deals with the use of hash based naming in ICN.

Routing and Forwarding As suggested by the already published ICN proposals and prototypes [9], two general approaches for routing and forwarding emerge. The one-step resolve / retrieve method where content requests are immediately routed towards an origin node, and the two-step resolve / retrieve where a Name Resolution Service (NRS) is queried for the information that is needed to deliver the content request towards an instance of the content.

One-step resolve / retrieve Figure 2.1 displays the one step resolve / retrieve mechanism. It is divided into two phases. In the first phase the rendezvous between the request message and the content itself is performed. In the second phase the content is delivered towards the requester. The illustration depicts a request for some piece of content that is to be retrieved. This request arrives at Node1 where the name of the requested content is looked up in the name routing table to further be delivered towards the source. When the request arrives at a node that is able to provide the requested content, the requested content is delivered to the content consumer. The content forwarding is then usually performed via a Reverse Path Forwarding (RPF) scheme. The RPF entries are created on each involved node, when passing on the request.

Two-step resolve / retrieve Figure 2.2 displays the two-step resolve / retrieve mechanism. Depicted above the ICN nodes is the NRS that introduces a layer of indirection, which is used to map ICN content names to topological network addresses. For the implementation of the NRS different options are known today. As depicted in Figure 2.2 the use of distributed hash tables or distributed databases are amongst them.

In contrast to the one-step approach, the two-step approach comprises three phases. In the first phase, the NRS is utilised to resolve content names to topological addresses. These topological addresses are subsequently, in phase two, used to route the request towards a copy of the requested content. Finally in the third phase, the content is delivered towards the requester.

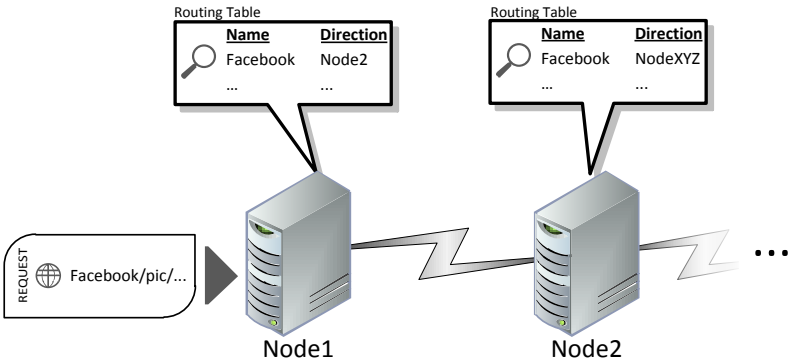


Figure 2.1: Conceptual view of one-step resolve / retrieve

Typically the topological address of the requester is attached to the request so the content can be delivered towards the requester.

2.1.2 NDN / CCNx

The NDN concept [15] originates from the Palo Alto Research Center (PARC). NDN is the concept that serves as the basis for the prototype implementation of CCNx [16]. Figure 2.3 depicts the general mode of operation of NDN. Interest packets are created by a content consumer to request any content, the Interest packets are then routed in a hop-by-hop fashion towards a known source of the content.

Content that should be made available via CCNx needs to be published so that content consumers will be able to retrieve the content through issuing request messages, known as Interests. Every piece of content in CCNx is made available through the use of certain names. These content names are used to identify and locate the actual content and thus to forward the requests. In some way they take over certain parts of IPs responsibilities of today's networks. Like in IP networks every content router needs to know where particular parts of the namespace are located. This information is distributed between name-based content routers through the use of a name-based routing protocol [17].

The name-based routing information populates the Forwarding Information Base (FIB)-table of the name-based router and is used to route incoming requests towards the content source.

Figure 2.4 depicts the architecture of a CCNx based router. The Faces indicated on the right side of the box are the generalization of an interface in the NDN scheme. This may be a connection to other nodes or a connection to an application running locally on the actual node. Besides the already mentioned FIB-table the router consists of the Pending Interest Table (PIT) and the Content Store. The PIT is used to store information

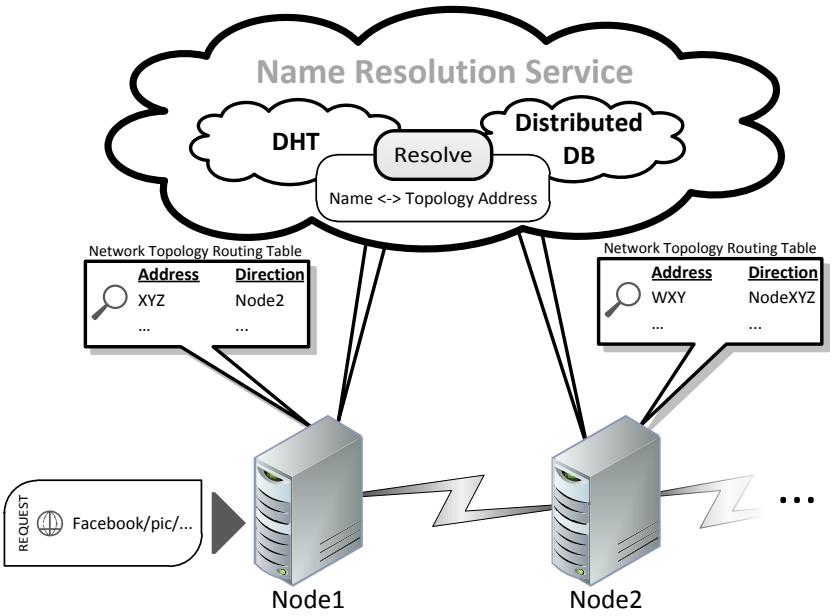


Figure 2.2: Conceptual view of two-step resolve / retrieve

about Interest messages while they are passed on to the next content router or a local application.

Interests in fact are the representation of a request message of the NDN concept that is propagated from a content requester towards a content source. The main purpose of the PIT though is to aggregate content requests. For pending Interest messages, which request the same piece of content, at most one Interest is forwarded towards a neighbour router. Subsequent Interests that arrive at the content router while an active Interest is pending, are noted in the PIT but their forwarding is suppressed. When the content chunks subsequently arrive at the router, following the reverse path of the Interest message, they are delivered towards every requesting consumer that previously sent an Interest for that particular chunk. This behaviour apparently results in a per chunk multicast like dissemination behaviour.

The Content Store is used to cache the received content to be able to deliver it to consumers that subsequently issue an Interest for the particular piece of content. It also allows the underlying mechanism to evolve from synchronous to asynchronous multicast. Without the Content Store, the multicast like behaviour will just be exploited when additional Interests for already pending but not already satisfied Interests arrive at a CCNx node. The Content Store alleviates this temporal coupling by locally storing the acquired content for later requests, resulting in an asynchronous per chunk multicast like dissemination.

Due to the request / response approach, the communication is always driven by the

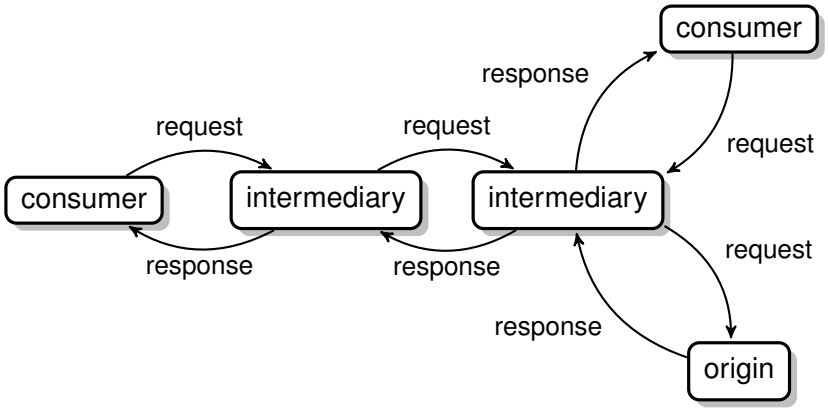


Figure 2.3: Abstract CCNx overview – adapted from [9]

receiver. Through the generation of an Interest, the client announces its willingness to receive a particular piece of content. This Interest is send to a content router that processes the Interest message in the following way [18].

1. Content store lookup is performed. If a content object matching the Interest is found within the Content Store, it is transmitted out the arrival interface of the Interest message. The Interest message is then dropped because it is satisfied and no further processing in needed.

2. PIT lookup is performed. If a PIT entry matching the content name already exists, meaning that an Interest for that piece of content is already forwarded to neighbouring routers, the incoming face is just added to the corresponding PIT entry and the Interest message is discarded.

3. FIB lookup is performed. A corresponding prefix for the name of the Interest packet is looked up in the FIB-table. If a matching prefix is found, an entry is created within the PIT describing the Interest. Subsequent the Interest is forwarded out one or more faces noted within the FIB.

4. No FIB match found. The node has no chance to satisfy the Interest, thus the Interest message is discarded.

These steps are performed on every content router on the way up to a source of the name. Whenever a particular piece of requested content arrives at a content router, a PIT lookup is performed to find all faces a corresponding Interest was received on. The resulting list of faces is used to transmit the data chunks towards all requesters that issued an Interest for that particular piece of content. Once the pending Interest is satisfied, the PIT entry is removed and the content object is stored within the local nodes' Content Store for future requests.

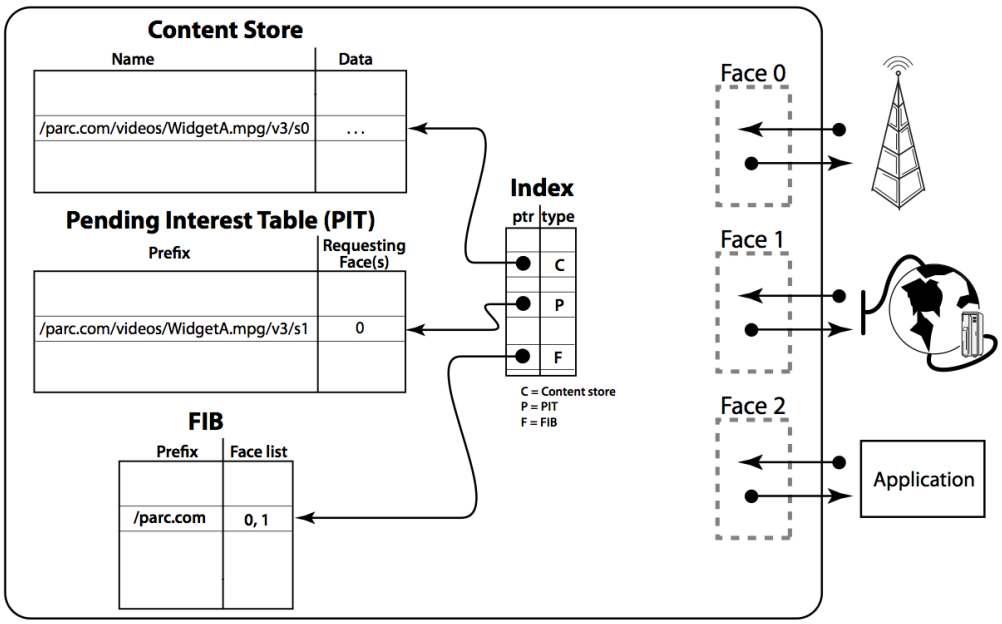


Figure 2.4: CCNx router overview [3]

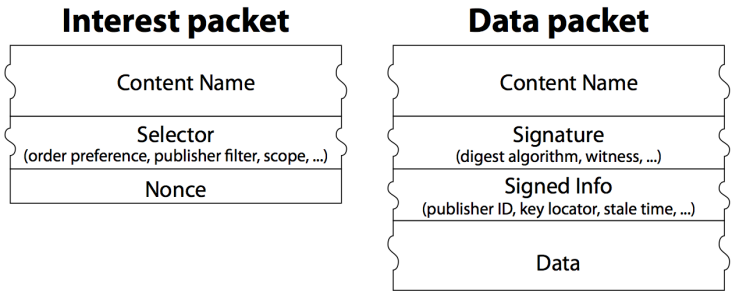


Figure 2.5: CCNx packet structure [3]

PIT entries use a soft-state model. If they are not satisfied or refreshed within a certain period of time, the PIT entries are dropped by a cleanup mechanism to prevent the PIT from overflowing. Further, if content arrives that no PIT entry exists for, the forwarding of the particular content is interrupted. However, according to the CCNx technical documentation [18], it is up to the implementation to cache the content or to simply drop it[2].

Figure 2.5 shows the structure of an Interest message as well as a data packet. Both of them carry a content name identifier that is used to either lookup the corresponding routing entry in the FIB or the outgoing Face in the PIT. Further the data packet also contains signature information to assure the authenticity of the data that follows.What is

[2]Our examination revealed that the official CCNx implementation adds the content to its local cache.

omitted in this illustration is the type indicator, which is the first header field in each CCNx packet.

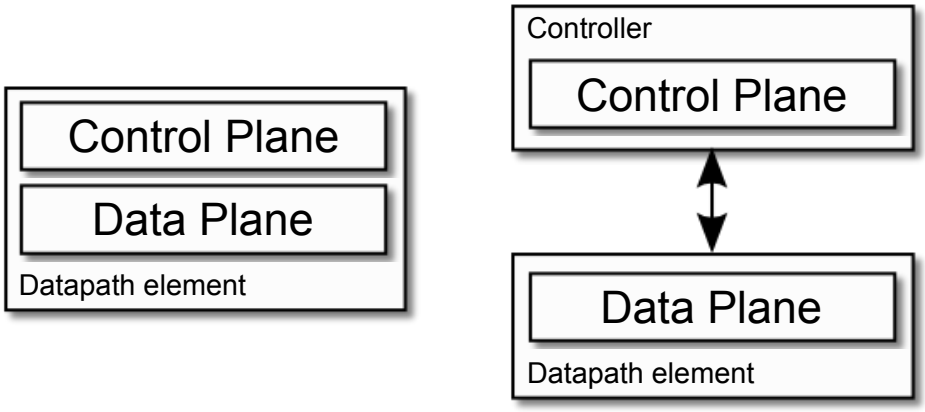
2.2 Software-Defined Networking

Fostered by for instance an increasing demand for flexible computing infrastructure, such as Platform-as-a-Service (PaaS), Infrastructure-as-a-Service (IaaS) and the like, SDN has become a vibrant topic in the field of computer networks. SDN as a term is meanwhile associated with different meanings. However, according to [19] SDN encompasses the separation of the control and data plane of datapath elements as depicted in Figure 2.6 – this is the understanding of SDN we also stick to throughout this work. Via this separation the control plane is externalized and forms an own entity by itself, the SDN controller. It communicates with the actual data plane via a SDN protocol. This separation of functionalities and responsibilities entails different advantages but also challenges, which we elaborate next. Throughout this section we will also give a more detailed overview of the general concept of SDN before eventually introducing OpenFlow, a standardized SDN protocol.

2.2.1 Concept / Overview

Due to the disintegration of the data and the control plane a (logical) centralized controller can provide its service for multiple datapath elements at once. This allows the control plane to utilize a comprehensive perspective of the whole network. The global network view introduces a distributed state abstraction and thus allows for enhanced decision making when controlling the data plane behaviour. Furthermore, as soon as there is a central entity controlling the data plane of various datapath elements it also becomes easier to provide an integrated interface to controlling the network. Applications do not need to have any enhanced knowledge of the underlying network structure or how to provision a certain communication channel in detail. By providing distributed network control abstraction with appropriate interfaces, the controller applications are able to partially hide the network complexity from business applications. Hence, as soon as applications are enabled to define and provision the network like they need it, a huge boost in service provisioning times accrues and the flexibility of the infrastructure rises.

Also the evolvability of the network is in focus of the SDN efforts. By defining the forwarding behaviour of datapath elements via standardized interfaces, the introduction of new protocols or mechanism to handle packets is simplified – of course just within the limited scope of the SDN. Nevertheless, it is possible to independently evolve the control and the data plane via SDN.



(a) Pre-SDN (b) SDN

Figure 2.6: Datapath element design

On the downside, the separated control plane also introduces increased latencies. Data plane packets have to be carried towards the controller for inspection and instructions on how to handle them need to be communicated back. Also the increased load caused by the aggregation of the control plane of various datapath elements within a (logical) centralized controller may lead to scalability issues and might provide single points of failures. All these are limitations which one has to keep in mind when dealing with SDN driven networks.

Different approaches exist on how to use SDN to configure the network. These approaches enclose the flow granularity, flow setup and the controller distribution.

Flow granularity The granularity of flow entries allow a coarse respectively finer matching on particular packet flows. Individual flows mostly consist of exact matches on certain header fields, whereas the aggregated flow entries rely on wildcard matches, which fit a broader value range. Individual fine-grained flow matching is seen as a good fit for network edge forwarding, whereas aggregated large-scale flow forwarding is considered the right fit for backbone parts of the network. Further, by for instance matching on certain additional fields the network might be able to split up flows for the same destination across multiple paths. Hence, the flow match granularity specifies the granularity with which traffic streams that are transmitted through the network can be controlled. It might be worth mentioning that the amount of provisioned flows is seen as one limiting factor of the SDN approach, since the Ternary Content-Addressable Memory (TCAM), like commonly used for line-speed forwarding lookups in network elements, is quite energy consuming and expensive. Hence, the available TCAM size per datapath element is quite limited.

Flow setup policy The setup time for network flows may vary. Utilizing reactive flow provisioning policies are just created whenever they are explicitly needed. Whenever packets arrive, for which the datapath element has no matching flow entries, the controller is triggered to provide the necessary instructions. The other extreme represents the proactive flow setup. Flow entries are installed prior to being triggered by arriving packets. This reduces the delay that the first packet is subject to, in case of the reactive mechanism. No delays for packet redirection towards the controller, its processing and flow setup occur. On the other hand unused flow entries may populate the flow tables that for instance consume valuable TCAM space.

Controller distribution The placement and cardinality of controllers to datapath element associations is not predefined. There can exist one dedicated controller for each datapath element, located right next to the datapath element, or even remote accessible, or a centralized controller for multiple datapath elements. When deploying controllers on a per switch basis, the advantage of a unified view of the network is reduced, at least at this layer. However, the controllers might of course implement some communication mechanism – which is not considered in SDN standardization at the moment – to share information amongst each other. Otherwise the latency path might be reduced by placing the controller besides the datapath element. Also the per controller load is reduced and can thus be utilized for enhanced packet processing.

There exist different projects and initiatives that develop and try to standardize SDN protocols. Further, different network equipment vendors introduced their own SDN strategies with proprietary approaches, which maybe extended by interfaces of commonly available standards. The Forwarding and Control Element Separation (ForCES) working group of the Internet Engeneering Taskforce (IETF), to name only one, already published [20] in 2003, a document describing the requirements for separation of IP control and forwarding. Since 2008 a new approach called OpenFlow [21] is present. Both efforts follow the similar idea of separating the control from the data plane by standardizing the protocol for information exchange between both planes. However, according to [22] both protocols differ with respect to their architecture, the forwarding model and their interfaces.

Since OpenFlow is the technology with the broadest implementation base, we will examine OpenFlow in detail in what follows.

2.2.2 OpenFlow

OpenFlow [21], as mentioned above, is the most popular of the already available SDN protocol standards. The initial OpenFlow specification was created in 2008 at the Stanford University. Since version 1.1.0 was published in 2011, the Open Networking Foundation (ONF) is responsible for the evolution, further development and standardization of OpenFlow. The actual version of the OpenFlow specification at the time of writing this document is 1.3.1, published in September 2012. Since most of the implementation base is only OpenFlow 1.0 enabled, we will continue by giving an overview of the functionalities of version 1.0 and subsequently elaborate on the key changes that have been made in the subsequent specifications.

As depicted in Figure 2.7 OpenFlow is a protocol used for communication between the externalised controllers and the datepath elements. To enable secure communication a Transport Layer Security (TLS) secured communication channel can be utilized between the datapath elements and the controllers.

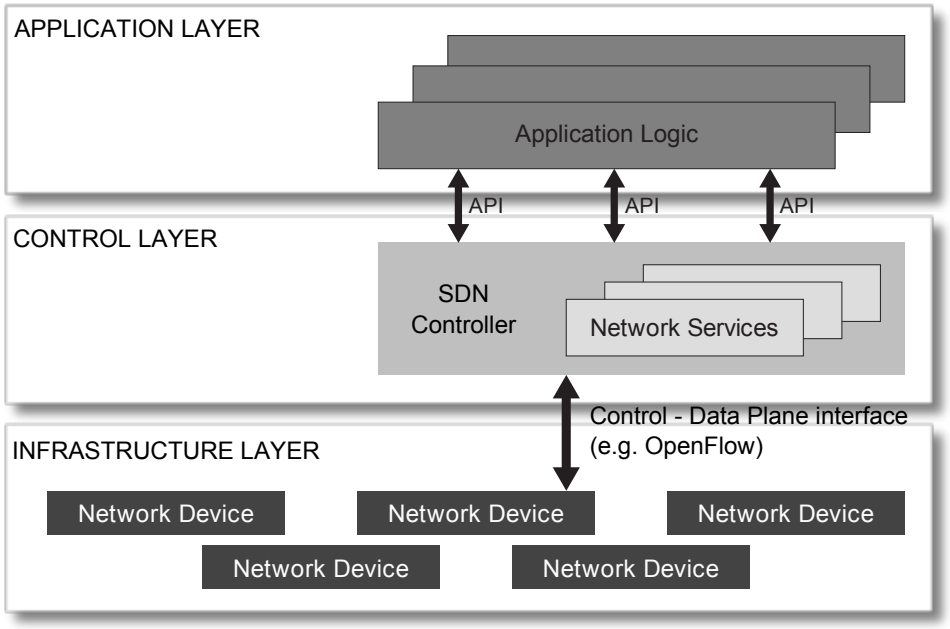


Figure 2.7: SDN architecture overview – adapted from [23]

The OpenFlow specification deals with a model that comprises of Flow Tables, Flow Entries, Matches and Actions. The Flow Tables host multiple Flow Entries. Each Flow Entry in turn consists of a Match expression as well as a corresponding Action. Whenever packets enter an OpenFlow datapath element, the initial Flow Table is evaluated to lookup an appropriate Flow Entry in order to subsequently apply the associated

Actions. The lookup is performed on certain packet header fields. These Actions can result in further lookups in a different Flow Table, in rewriting certain header fields as well as packet transmission actions, to name only some. These Actions are not even mutual exclusive and hence can be combined.

Further the OpenFlow protocol consists of various types of messages, which are used for communication between control and data plane. We will now introduce the most essential message types that the present work relies on.

Packet_In messages are send from the datapath elements towards the controller, whenever the datapath element does not have a matching flow entry available in its Flow Table or the matching Flow Tables explicitly requests the packet delivery towards the controller. Packet_In messages contain the packet that triggered the controller interaction either partly or as a whole. Having the necessary packet information available enables the controller to inspect the packet and in the further course allows for the instruction of the switch on how to handle the packet.

Packet_Out messages are send from the controller towards the datapath elements. Via Packet_Out messages the controller is either able to send self-carfted packets via the control plane to a datapath element, which are subsequently forwarded by that datapath element, or the controller can instruct the datapath element to forward a buffered packet. The Packet_Out message basically tells the datapath element on which physical port to output the packet. It is also possible to add additional packet modification instructions that the datapath element applies before forwarding the actual packet.

Flow_Mod messages are send from the Controller towards the datapath elements to either add, modify or delete existing flow entries on datapath elements. These flow entries may also comprise of rewrite and output instructions.

Port_Status messages are generated by the datapath element to inform the controller about port state changes or if ports have been added or removed from the datapath element.

xy_Statistics messages are used by the controller to gather statistics about flow entries, flow tables, ports, etc.

Packet_Out and Flow_Mod messages carry the information about which action is to be performed by the datapath element, either to a buffered or a contained packet (Packet_Out) or to future packets (Flow_Mod). These actions comprise of simple instructions like forward packet out of particular ports, but packet rewriting actions are also applicable. Hence, VLAN tags can be added or removed, IP or MAC addresses and further header fields can be rewritten. Multiple of those actions are allowed per Packet_Out

Ingress port	Ether src	Ether dst	Ether type	VLAN id	VLAN prio	L3 src	L3 dst	L3 proto	L3 ToS	L4 src	L4 dst

Table 2.1: OpenFlow 1.0 match structure; L3 = IPv4; L4 = TCP/UDP

and Flow_Mod message, hence, a combination of different packet modification operations is possible.

To define which packets should be altered, the Flow_Mod message also contains a match structure that defines which values must be present for the associated actions to be triggered. The match structure of OpenFlow 1.0 is a fixed field structure containing the fields shown in Table 2.1. This structure is apparently not very flexible and lacks extensibility. For instance IPv6 fields are missing besides other desirable fields. Hence, with version 1.2 of the OpenFlow specification, the extensible match structure was introduced. The match part is changed from a fixed length entity to a Type-Length-Value (TLV) structure. Enabled by this change the IPv6 as well as the Multiprotocol Label Switching (MPLS) header field match structures have been specified. Further the match classification has been divided into different classes. One class is the OpenFlow basic class that all fields mentioned so far belong to. Another class is the experimenter class that can be used by developers to evaluate certain new matches. Doing so, developers are enabled to implement their own match types to extend the OpenFlow capabilities even further. More classes are designated, which we do not rely on in this work.

The major specification changes of the OpenFlow protocol are listed in what follows [24].

OpenFlow 1.0 First OpenFlow version with broad vendor support.

OpenFlow 1.1 Added support for multiple Flow Tables, Group Tables, Virtual Ports and MPLS.

OpenFlow 1.2 Added support for extensible matches, IPv6, controller role change and experimenter extensions.

OpenFlow 1.3.0 Added support for per-flow meters, IPv6 Extension Header handling and Provider Backbone Bridging (PBB). Refactored capability negotiation and increased Flow Table Miss handling flexibility.

OpenFlow 1.3.x Clarifications and corrections.

2.3 ICN over SDN

In the past there have already been initiatives to allow ICN to be operated over SDN. In what follows, we will give an overview of these different approaches and their modes of operation.

2.3.1 Software-Definded Internet Architecture

Raghavan et al. argue in their paper [25] that the coupling of the architecture and infrastructure of networks result in substantial costs for the development and deployment of new network protocols. The authors advocate the decoupling of network architecture and infrastructure by leveraging the SDN paradigm in conjunction with a distinction between the core and edge forwarding mechanisms like known for instance from MPLS.

The Software-Definded Internet Architecture (SDIA) called design approach promises to ease the adoption of new Internet architectures, like for instance ICN. Raghavan et al. criticize that any significant change to network layer protocols requires the substitution of all forwarding devices, since the forwarding logic is protocol specific and often times implemented in hardware. This implementation in hardware is done to speed up the processing but at the same time limits the flexibility for the evolution of the protocols.

The design builds up on the separation of core and edge network addressing schemes like previously proposed for instance in [26, 27]. For the intra-domain design a “fabric-like” approach is proposed, which utilizes arbitrary and decoupled forwarding mechanisms in the core and at the edge of the network. Furthermore, the control plane protocols and mechanisms are independent per network domain. As a consequence of this claim, the three communication patterns *edge-to-edge*, *edge-to-host* and *host-to-host* have to be supported.

Network edge nodes are according to their design expected to use software defined forwarding instructed by the SDN controller that savvies both addressing schemes and is thus able to decide how to handle edge and core packet delivery. For establishing the inter-domain communication each domain is represented by a single logical server in the algorithm used to compute inter-domain routes.

As the authors also point out, the concept of decoupling architecture from infrastructure can help to deploy ICN services. Hence, our take aways for the operation of an ICN over SDN are that ICN has to be operated via a common core forwarding mechanism, which is preferably not specific to ICN. Therefore a scheme must be developed for the mapping of ICN to SDN core addresses and vice versa. Further, the core forwarding mechanism can be different in distinct domains. The responsible (logical) SDN controllers of the domains do only have to agree up on a mechanism and format for their routing information and packet exchange.

Further, OpenFlow in the sight of the authors improves the situation in terms of decoupling architecture and infrastructure to some extent. Nevertheless, they do argue that entirely general packet matching capabilities would be required to overcome the current limitations as a whole, which is not expected to happen soon due to substantially higher component costs.

2.3.2 CONET

Blefari-Melazzi et al. [28, 29] in the course of the European research project OFELIA propose and evaluate their approach on how the ICN functionality can be supported via the use of the SDN paradigm. Their work is based on Content Network (CONET) [30], a NDN based ICN implementation.

2.3.2.1 Changes to CCNx

CONET differs in some aspects from the CCNx implementation, of which we want to point out just two pivotal changes.

Forwarding-by-Name CONET introduces a routing logic centralization to CCNx. Therefore NRS nodes are used as a central component in each domain. CONET forwarding elements still comprise of the FIB and PIT. However, other than in the CCNx implementation, an entire FIB – which is called Routing Information Base (RIB) – is only maintained by the NRS nodes. The forwarding nodes FIB is used as a cache for the NRS maintained RIB entries. Whenever a required forwarding entry is missing on the CONET node, the NRS is queried and the received forwarding information is cached in the local FIB.

The NRS nodes of different domains form the inter-domain routing infrastructure. Hence, the NRS nodes are in some sense comparable to a Border Gateway Protocol (BGP) Route Reflector (RR) [31]. BGP nodes, in contrast to CONET , maintain a full routing table instead of using the RR as a fallback for cache misses. Further, the NRS nodes of CONET itself establish connections to NRS nodes of other domains for the purpose of exchanging inter-ICN routing information. This architecture is put into place to prevents the propagation of all name forwarding entries to each forwarding node on a global scale. Nevertheless, all global NRS nodes need to be notified in case of a routing table change.

Segmentation and transport Since CCNx chunks are by default 4 KByte in size, CCNx relies on fragmentation or segmentation of underlying protocol layers (IP/User Datagram Protocol (UDP)/TCP) to fulfill the Maximum Transmission Unit (MTU) requirements of the underlying network. By performing those kind of operations, the

naming information that is put in front of each chunk is only present in the packet carrying the first segment. Hence, in the following packets this information is no longer available to intermediate network devices, they consequently cannot base their decisions on this information. To overcome this issue, Blefari-Melazzi et al. introduce a novel protocol layer that provides a new segmentation mechanism. The chunks are thus split into so called *Carrier-Packets* [32, 33] that meet the MTU requirements and at the same time carry the required name information as part of the header in each packet (Figure 2.8b).

2.3.2.2 Packet formats

The authors elaborate on different packet formats, which might be used to transport CONET *Carrier-Packets* within IP and further extend these by measures to support efficient Application-Specific Integrated Circuit (ASIC) based matching.

IP option The first proposal, depicted in Figure 2.8a, introduces a new IP option header. This proposal is according to the evaluation of the authors basically possible but carries certain drawbacks [30]. The evaluation shows that routers on some paths drop IP option carrying packets on a statistical basis or in the worst case even all of them. The measurements further show that even with low bandwidth transmissions of below 6 Mbit/s, the throughput is in all cases lower than without the CONET IP option. The same applies for the measured delays, which also increased in all cases. These effects are caused by the fact that packets with IP options are sometimes discarded as a cautionary measures to Denial-of-Service (DoS) attacks. Packets that carry IP option headers have to be processed by the processor of various routers instead of being processed purely via ASICs. Fransson et al. [34] also confirm these results of slight increases in delay and jitter and a severe increase in loss rate.

CONET transport Figure 2.8b illustrates the proposed transport layer protocol (Information-Centric Transport Protocol (ICTP)) that the authors designed for basic delivery through today's Internet. The IP protocol number carries the specific value assigned to ICTP indicating that ICTP is used as the transport layer protocol. Subsequently all ICN related information is carried only within the IP payload.

CONET transport + Tag Explicitly for the transport of CONET packets via SDN an additional tag is introduced, like shown in Figure 2.8c. Since the name component of CCNx is of variable length and matching of variable length data with ASICs is a complex and costly operation, an 8 byte fixed length tag field is introduced in

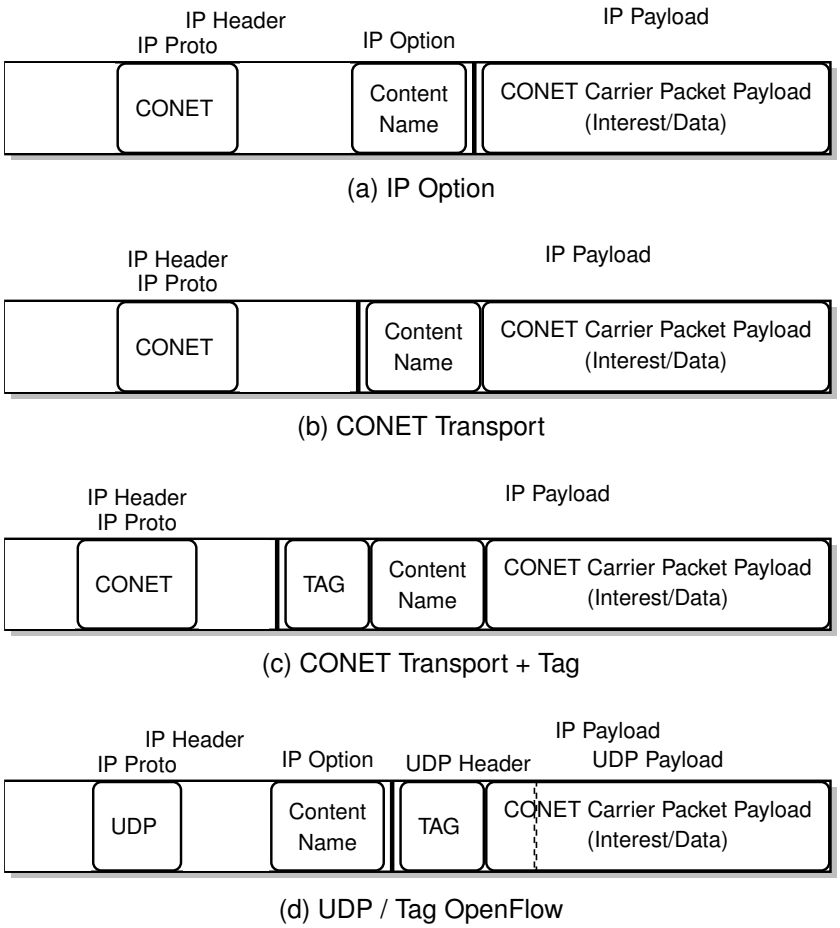


Figure 2.8: CONET packet format options – adapted from [28]

between IP header and its ICTP payload. This tag field should further be used by SDN switches to perform their flow entry matching on. By introducing such new tag field, the OpenFlow protocol needs to be extended to support these novel header fields. Consequently this approach is just applicable to OpenFlow protocol versions >= 1.2 running on switches that are also modified, able to understand and perform matching on this newly introduced header field.

Further the use of MPLS and Virtual Local Area Network (VLAN) tags is mentioned as an implementation opportunity, which is then quickly declined due to possible interference that arise if those techniques are already in use in a particular domain.

UDP / Tag OpenFlow To overcome the limitation in applicability introduced by the above approach, a combination of the *IP option* and *CONET transport + Tag* is presented in Figure 2.8d. The indicated IP protocol field is set to UDP, but the

packet is not provided with a valid UDP header. The UDP header fields are abused to carry the forwarding tag. Since the UDP header consists of four fields, namely *source* and *destination port, length* and *checksum* of which only the source and destination ports (2 * 2 byte) are OpenFlow matchable, the tag size is reduced to 4 byte and encoded in these fields. The ensuing content starts in the middle of the regular UDP header, which is possible since the hosts network stack is modified such that no regular UDP processing of the packets is performed. As a consequence of this protocol field abuse no UDP communication is possible any more, at least on the underlying IP address. Nevertheless, by these measures it is possible to implement a fixed length tag matching for ICN even with OpenFlow version 1.0.

2.3.2.3 SDN implementation

Blefari-Melazzi et al. opted for the *UDP / Tag OpenFlow* approach, mainly because their testbed did not support the *flexible matching* feature of OpenFlow 1.2. Their SDN border nodes, supported by the NRS, perform the tagging of incoming packets. The NRS is used to ensure that the association between content names and tags is unambiguous, since each border node requests these tags. Furthermore, the NRS is responsible for establishing paths for the tagged content through the network. Edge nodes, when serving as egress nodes for Interest packets, maintain content name to tag associations in order to map returning content to the specific return path.

To maintain the ability of transporting regular UDP packets through the SDN special purpose IP addresses are used. Hence, if packets are matched for their tag value, the match also contains these reserved IP addresses. Unique IP addresses are assigned to both, Interest and data packets to explicitly distinguish these types.

2.3.3 PURSUIT

The PURSUIT [35] approach to ICN relies on so called zFilters for packet forwarding. zFilters as described in [36] are bloom-filter structures that are used to identify the links a packet has to traverse on its way through the network. Each link on all forwarding elements is assigned a specific bloom-filter link-id that is determined in the switch bootstrapping phase from an entity called the *Topology Manager*. Packets that are to be delivered through the network carry a forwarding identifier with them, which is the bitwise *OR* combination of the link-ids of all links the packet is supposed to traverse. Whenever a PURSUIT node requests content, it sends its request to the rendezvous system, which in turn looks up the location of the requested content and with the support of the *Topology Manager* constructs two forwarding filters. One forwarding filter for the path from

rendezvous system towards the publisher and another from publisher to requester. The latter forwarding identifier is then along with the requested content name delivered to the publisher via the help of the first identifier. The publisher uses the provided information to subsequently deliver the requested content to the requester.

The authors of [37] adopt the same mechanism for their SDN approach. Requests are delivered to the *Topology Manager*, which determines the bloom-filter forwarding identifiers and subsequently instructs the OpenFlow controller to provision the required forwarding rules for all affected switches. The source and destination Media Access Control (MAC) address fields are used to carry the forwarding identifier in each packet. By changing the semantics of the MAC layer address fields the whole network needs to be SDN enabled. Further the network stack of the involved ICN nodes needs to be put into promiscuous mode, because packets that need to be processed carry part of the zFilter in the destination MAC address instead of the actual destination MAC.

Since a bit-wise wildcard match on MAC addresses is not supported by OpenFlow yet, the forwarding rules installed by the controller are exact matches. This constraint limits the benefits of using the bloom-filters. For each path through the network a separate forwarding rule needs to be installed, instead of only one bit-wise wildcard forwarding rule per interface, like in the SDN-less PURSUIT implementation.

2.3.4 Info-Centric Data Center Network (IC-DCN)

The authors of [38] propose the integration of the SDN and ICN paradigm as a new architecture for Data Center Networking (DCN).

Jun Ko et al. follow the previously introduced concept of using fixed-length labels for data forwarding on the data plane. Packets are mapped onto network paths as soon as they enter the SDN domain and are then forwarded according their particular label.

However, the authors present varying ideas to improve the efficiency and scalability of the general approach of using fixed-length label based forwarding. They argue that due to the rather static topology in DCN environments with fairly infrequent changes, the *routing service* is less stressed than in more flexible environments with a higher degree of node mobility. Consequently, to improve the cache efficiency the implementation of cache-aware routing is suggested. Through the use of en-route caching, cache hits take place only opportunistically, while in SDN the centralized controller allows for cache-aware routing. The authors propose a hierarchical routing strategy to improve the hit ratio. The idea of this hierarchical routing is for paths of different consumers to merge as early as possible on the path towards a producer. Therefore the controller computes multiple Shortest Path Trees (SPTs) rooted at each producer. Whenever a content request arrives, the controller, by utilizing hash-based assignment, maps the request to a particular SPT rooted at a particular producer that is providing the requested data. An

even more complex strategy is the preservation of routing requests along with the assigned label such that subsequent requests – also of different consumers – follow paths of the previously selected SPT, which includes a higher probability of having a cached copy available.

In particular their SDN controller maintains the following functions:

- *Routing service* – Collects topology and content location information and decides on routing paths
- *Naming service* – Assigns namespaces for producers and assures consistency together with the integrity of publications
- *Policy service* – Provides policy repository for the management of the system

The authors also explicitly take up on the issue of the scalability of the approach. They propose to utilize a name to label mapping cache at the edge nodes. Further, multiple path assignment servers are used. Hash functions are used to consistently select the valid servers to query for name to label mappings. For redundancy and fault tolerance purpose, different hash functions are used.

2.3.5 Discussion

The various approaches introduced so far impose different requirements or restrictions on the environment that they are operated in.

PURSUIT requires the ICN-SDN nodes network interfaces to operated in promiscuous mode in order to hand over the data to the upper layers. The implementation of the zFilters with their generic wildcard based matching and forwarding on link identifiers sounds beneficial, also for the implementation in combination within SDN networks. However, the limitation that MAC address are only matchable via exact matches in OpenFlow version 1.0 immediately ruin these possible benefits.

The work of Blefari-Melazzi et al. shows that the integration of CONET and SDN is applicable. However, they also revealed different weak points.

IP option The forwarding in pre-OpenFlow 1.2 networks relies on the utilization of IP option headers. This is no problem in closed environments, however, if the interconnection of CONET domains is performed through the public Internet, the service level is likely degraded due to the IP option handling of certain intermediate forwarding devices.

Border nodes Border nodes need to be ICN aware. They are required to perform encapsulation and decapsulation operations, which is an easy task for custom software to run on a general purpose computer, but leads to difficulties with today's ASIC based forwarding network elements.

Network stack changes The network stack of CONET nodes need to be modified, to support the abuse of the IP protocol field with the UDP value while not carrying a UDP packet. Thereby generally prohibiting regular UDP communication with other hosts.

The design presented by Jun Ko et al. relies on the integration of ICN and SDN nodes. SDN nodes are required to understand the ICN protocol and provide cache space. Further, the cache-aware routing provides just a partial improvement by changing from probabilistic to more predictable cache selection while still being limited to on-path caches. The approach of improving the scalability by utilizing hash functions to deterministically distribute the path label requests requires the ICN nodes to be aware of the hash function and its parameters as well as being able to perform this operation, which is not possible with today's standard OpenFlow equipment.

Introducing these requirements and restrictions makes the implementation and deployment of these approaches difficult. Therefore, we think that the elimination of these points is key to improve the deployability of ICN over SDN.

Chapter 3

ICN Research Challenges

In the following, we will give a detailed overview of the problem space this work is concerned with. In Section 3.1, we start by elaborating on the challenge of state management in ICN, followed by security threats that can harm the ICN infrastructure in Section 3.2 as well as scalability and deployment issues ICN suffers in Section 3.3. Finally, we elaborate on the challenges that arise when deploying an ICN-aware SDN in Section 3.5.

3.1 State management in ICN

ICN introduces new states to the network to perform the required actions of processing content requests and providing the requested data. The content must be located within the network and subsequent be delivered to the requester.

ICN utilizes a routing or publication system to locate content within the network. It relies either on the FIB in the one-step resolve / retrieve or the NRS in the two-step resolve / retrieve based ICN implementations. The routing / publication systems are used to map the location independent identifier to a topology dependent one. Therefore, the NRS points directly to location addresses, whereas the FIB points towards the origin of the content on a hop-by-hop basis. The two-step resolve / retrieve approach leaves the actual forwarding of content to the underlying network layer protocol. Whereas the one-step resolve / retrieve approach performs the localization of the content solely via the forwarding of the actual ICN request.

The subsequent delivery of content is handled different in various ICN proposals. The one-step resolve / retrieve leaves RPF states on all hops along a traversed path while forwarding the request in accordance with the information maintain in the local FIB. These RPF states are maintained on a per request basis and are thus present in large quantities as well as being subject to frequent creation and deletion. The data plane

consequently interacts with and thereby influences the control plane of ICN forwarding nodes on a constant basis. This behavior is contradistinctive to the behavior of the prevalent routing system where control and data plane interact in a way that only explicit routing protocol messages result in the modification of control plane state while regular packet handling requires only the lookup of control plane related information. Utilizing the two-step approach, typical IP addresses are used for the location identification of content. An exception is for instance PURSUIT, which uses Bloom filters to describe the interfaces a packet has to traverse [6]. In the IP case, the content is send back to the sender of the request simply by swapping the source and destination location identifier while PURSUIT requests explicitly contain the Bloom filter used to deliver the content.

Altogether, ICN introduces the new states of content publication and in certain ICN types RPF states to the network. They are essential for the functioning of the ICN approaches. Further they are present in large amounts and also quite dynamic, considering the deletion and creation of RPF states per content request. The underlying infrastructure must be capable of maintaining these states.

3.2 Security threats to ICN infrastructure

Introducing these new states to the network opens the door to multiple security threats, we reported about earlier on in [39, 40, 41]. The RPF states, which are logically part of the control plane, are driven by the data plane of the network and are thus directly influenced by the users. Requests passing by on the data plane trigger the creation of RPF states on the control plane. This design decision allows to seriously restrain the service level of network elements. We specifically identified the following threats:

Resource exhaustion The RPF states have to be stored for a period of time, either until the requested content is received or until the request times out, hence, some amount of memory is consumed per each RPF entry. If bulks of those states are already created and no further memory is available, the ICN forwarding node is not capable of storing mandatory information required for content response forwarding. As a consequence, the ICN node is not able to properly perform the ICN packet forwarding anymore. Further, processor resources are required for the state maintenance operations and can therefore also pose a bottleneck and a security threat – either maliciously or not.

State decorrelation RPF states must exist for each hop. An entire path needs to be established and functioning. This requirement increases the probability of path failures. Due to dropped forwarding states the packet forwarding is interrupted somewhere in the network. This effect is reinforced with increasing path lengths.

In Addition to that, valuable RPF memory space is taken on nodes upwards in direction of the requester, which have to await the RPF state expiration or its refresh. This might even lead to a pile up of states on these particular nodes.

Further, the publication system has to be opened up to users. The users must be able to publish content. ICN approaches that rely on a hierarchical content name structure might suffer less on this problem, a general routing entry allows for the publication of content under a more specific name in the namespace. However, ICN proposals using non-hierarchic namespaces do not allow for this aggregation. Hence, each unique piece of content has to be registered on its one, increasing the load and thereby the pressure on the publication system. Consequently this can be exploited for DoS attacks.

3.3 Scalability problems in ICN

The step of exposing knowledge about content to the network itself places a high burden on the infrastructure. Today's network routing protocols deal with subnets, an aggregation of network nodes. BGP as the prevalent global routing protocol actually has to deal with a routing table that contains nearly $5 * 10^5$ entries [42]. However, ICN has to take the actual content into account that is served by nodes in the network. Dependent on the ability to employ aggregation or not, the numbers of registered second level Domain Name Service (DNS) domains or the amount of unique URLs in the index of the web search engines serves as a rough estimator. The amount of registered second level DNS domains amounted to $2.52 * 10^8$ in April 2013 [43] and raised further to $2.65 * 10^8$ entries in Q3 of 2013 [44]. Even bigger is the amount of indexed URLs contained in the web search engines. Googles search index, according to their own information [45] contained 10^{12} entries back in the year 2008. Both of these numbers increase over time and of course the publication / routing system must even meet future requirements.

The bottom line is that ICN has to managing a much bigger content name space that is also even more dynamic than today's network node addressing. In the worst case ICN has to keep track of pieces of content and their location. Also the registration of multiple content copies within the NRS can increases the complexity and thereby acts as an opponent of scalability.

Additionally, per request states like introduced by some of the ICN approaches limit the scalability. Resources for the maintenance of the required RPF states increase with the number of users and their content requests. It is no longer only the bandwidth and FIB lookup speed that influences the throughput of a network device. Furthermore, the memory size and state maintenance performance, respectively the processing power influences the forwarding efficiency to a great extent.

3.4 Deployment challenges in ICN

The deployment of ICN is also considered a challenge. The network layer protocol is by concept the convergence layer that all packet forwarding parties have to agree up on. The data link layer specifics only have to be agreed up on by adjacent nodes and the transport layer specifics are negotiated between the communication endpoints. These days the entire Internet mainly converged to an infrastructure that is dedicated to and thus optimized for the forwarding of IP packets. Introducing a new protocol into this well established ecosystem raises major challenges. Since network layer protocols are what each transient packet forwarding party needs to deal with, the introduction of novel protocols, replacing IP or even running them in parallel is not an easy task.

The deployed network infrastructure performs packet forwarding in hardware to reach the line speed of today's core network interfaces. This infrastructure can thus not be enabled to support ICN functionality via the rollout of a simple software upgrade. Further also the resource requirements in terms of memory will exceed the available resources to store the increased amount of information introduced by the content name routing, at least in case of a one-step resolve / retrieve ICN. Additionally, the inherent functionality of caching requires further storage capabilities. Network nodes need to be provided with memory or storage for passing packets.

Due to these limitations all ICN protocols offer the capability to be operated over the actual IP enabled network infrastructure. In what follows, we provide a specific list of different deployment scenarios and the certain challenges they entail.

Integrated A two-step resolve / retrieve ICN approach can utilize the IP infrastructure as the underlying packet transport mechanism. To recollect, the NRS is used to map content names to topological identifier. The network layer protocol though does not necessarily have to be newly invented and ICN customized. Hence, TCP/IP can be used to exchange content between ICN nodes.

This method is rather easy to achieve in today's networks. The basic transport infrastructure does already exist. Operators do only need to deploy and provide the NRS and make it available to ICN users. However, certain ICN approaches will benefit from the introduction of their specifically tailored transport protocols.

Overlay A one-step resolve / retrieve ICN approach can also utilize the IP infrastructure for packet forwarding. But other than the two-step resolve / retrieve systems no connections towards arbitrary nodes are established in an ad-hoc manner. The connections are pre-configured and serve as the emulation of direct connections between specific nodes.

The bootstrapping of the overlay network has to take place and entry points into the overlay must be known to new ICN nodes. The configuration of multiple transport

tunnels between nodes is required that subsequently act as neighbor nodes in the overlay. However, they might network proximity-wise be quite far apart. Hence, the content is carried long distances without being ICN-wise processed by intermediary nodes on the way. This in fact counteracts the general idea of ICN, to cache content close to the users. To prevent the manifestation of these unfavorable constellations, the topological coordination of the overlay is desirable. A coordination mechanism that organizes the topology and thereby prevents unfavorable connections to be established.

Native The native deployment requires the nodes involved in the packet forwarding process to be ICN enabled. No underlying protocol like IP or TCP/UDP is required. All nodes on the path are ICN enabled and perform regular ICN forwarding while additionally providing supportive functionalities like content caching.

This method requires the most modification of the network. All forwarding elements have to be removed and replaced by ICN enabled counterparts – in case of devices that perform the forwarding process in hardware – or at least get upgraded to introduce the ICN processing capabilities. These measures encompass high investments for network operators and will thus very unlikely be performed in one step. Nevertheless, this method is mostly efficient since all nodes cooperatively support the ICN paradigm and can thus perform optimal processing and forwarding operations.

These specified deployment methods all encompass their own challenges and drawbacks. It is desirable to support a close to native deployment without the costly step of replacing all boxes throughout the network.

Further, ICN utilizes the content-oriented communication pattern. Content is first of all published within the network and subsequently delivered from cache nodes or the origin towards the requesting node. According to Detti et al. [30] the conversational communication pattern are less focused in the ICN community. However, it is characteristic for Voice-over-IP, Video Telephony, Management traffic and the like. Schemes exist to support this conversational pattern, see [46], but the effectiveness and benefits compared to IP is according to the authors questionable. Hence, an ICN only network as proposed by some projects would give advantage to the content dissemination use case, but at the same time adversely affect the conversational communication.

It is visible that an evolutionary path from IP to ICN networks is required and taking the concerns about the conversational communication pattern into account, maybe also the long term co-existence of both networking approaches is favourable.

3.5 Challenges of ICN over SDN

SDN technologies these days are mainly developed with focus on the conversation oriented IP protocol. OpenFlow makes no exception in this case. The SDN forwarding elements are optimized for the matching of the fixed length headers like present in these IP packets. ICN implementations like CCNx on the other hand use variable length name fields that the forwarding decision are based on. Hence, performing these variable length matches on the ICN packets is not feasible for SDN forwarding elements. Raghavan et al. [25] underline this while claiming generic full packet matching for their SDIA design approach. However, the authors themselves do not believe in this being implemented in hardware forwarding elements, due to an essential increase in equipment costs.

The challenge of combining these networking techniques without fundamentally changing the basic working principals of both along with the requirement of operating ICN side-by-side with IP, either for the evolutionary migration or the general parallel operation, exists.

Further, the control plane of the SDN forwarding network devices is centralized within the controller. The SDN controller thereby maintains an enhanced view of the topology and the actual traffic condition. It is thereby enabled to oversee the cache status, whether content was previously requested or not and further, if it is available in a cache somewhere within its controlled SDN domain.

Similar to the NRS in the two-step resolve / retrieve ICN approaches, the controller is aware of content locations respectively the topological identifiers of the nodes providing the content. Hence, the ICN-SDN approach enables a one-step resolve / retrieve approach to harness NRS capabilities and for instance utilize off-path caches. The controller can monitor the cache nodes that belong to its own domain and collect certain performance parameters. The range of information can thereby reach from the available storage over the processing resources to the interface utilization of the cache nodes. All these information can subsequently be considered in the cache node selection process with the objective of preventing the overflow of already saturated nodes. The same applies to network resources. If the controller is provided with the necessary information it can establishes explicit paths through the network and designate them to the actual content requests and responses in order to avoid overloaded links when forwarding ICN packets.

Additionally, the controller is able to group content regarding certain namespaces, content types or any other content information that is deducible from the ICN header onto particular cache nodes. Doing so, the amount of network wide storage can be reduced while at the same time the cache hit probability is increased. Content is not held available in multiple nodes but within the node or cache group that is responsible for the type of content or the particular name space. Hence the content is available only once

within the entire network, or a particular region. Consequently, a lower overall amount of network storage can hold a larger amount of de-duplicated content. Of course, this might result in increased distances between requester and cache and thus can result in increased latencies since the actual cache might be a few hops further away. However, for the initial deployment it is supposed to be beneficial that through the de-duplicated content caching, the overall amount of required cache space is reduced, while at the same time the hit rate is increased, through the direct steering of requests towards the designated ICN caches.

At last, the necessity to additionally introduce an overlay management, either manually or via a network service that keeps track of the location of ICN nodes and makes sure that the connections among the nodes is proximity-wise advantageous, is made redundant. The SDN controller has the topology information available to provide paths through the SDN network in any case. Beyond that, the controller is able to take even inactive paths into account that the regular overlay management has no notion of.

Chapter 4

Concept

In Section 4.1 the introduction of our approach to the initial deployment of ICN over SDN starts with the elaboration of the objectives we pursue. The chapter continues in Section 4.2 with the introduction of the basic approach for an SDN that enables enhanced ICN-aware forwarding. Section 4.3 covers the advancements we apply to the basic approach namely allowing for multiple caches to be queried in parallel, to perform request aggregation or to drive cache population.

4.1 Objectives

We aim at the integration of ICN and SDN. The primary goal thereby is to investigate an evolutionary path from existing IP networks towards an initial real deployment of ICN in islands like for instance Data Center (DC) environments or throughout Internet Service Provider (ISP) networks. It is necessary to continue to provide connectivity for prevalent network protocols while at the same time enabling a close to native processing of ICN packets. The Overlay as well as the Integrated approach entail certain drawbacks as already mentioned in Section 3.4. It shall be evaluated whether an ICN-aware SDN approach can provide benefits for the initial deployment of ICN protocols.

The following catalogue lists requirements that an ICN enabled SDN approach must meet.

Island interconnection For the ICN-SDN approach an interconnection scheme must exist. It is not sufficient to operate a standalone ICN island. Islands have to be inter-connected and must further also be able to communicate with non-SDN based ICN domains or clients. Therefore the controller has to have basic information about external ICN nodes that might be used as next hop communication partners. Hence, some name routing protocol needs to be in place and supported by the controller to exchange the required information.

ICN diversity Multiple ICN approaches and proposals in different maturity levels exist. Further, the research on ICN is still in a fairly early stage. Hence, a universally applicable mechanism not only usable with one specific ICN approach is beneficial and even deployments of different ICN implementations will be operated in the same network.

Network stack Former approaches, which intent to support the delivery of ICN over SDN like [30, 37] require the modification of the regular network stack. Thereby preventing the processing of other valid network communications like UDP in case of CONET. To allow for a simplified deployment, the replacement of the actual network stacks of common ICN nodes is to be prevented.

Parallel deployment A crucial requirement is the ability for the co-existence of the ICN along with the IP protocol in the network. Since the general focus is on the initial deployment of ICN, IP connectivity has to be provided at least transiently.

OpenFlow v1.0 The only OpenFlow version that is broadly supported by hardware switches these days is OpenFlow v1.0. Hence, to support the implementation of ICN-SDN already today the approach is meant to be executable even with this version.

Cache knowledge The ICN-SDN must be able to deal with caches and steer requests towards them to emulate a native ICN deployment including caches throughout the network.

Request aggregation CCNx is able to detect multiple concurrent requests via the use of its PIT and prevents the upstream forwarding of more than one request towards the content origin. The returning data is subsequently duplicated and delivered downstream to all requesting nodes. Such request aggregation mechanism should even be possible SDN wide. The controller is aware of pending requests and is thereby able to aggregate them network wide, without being limited to on-path aggregation.

Response forking Response forking describes the process of duplicating the content response without the prior reception of a request. This mechanism is further intended to be used for the steered population of content caches. The controller should be able to decide which content to store in which cache and subsequently drive the cache population.

Request forking By duplicating request packets, the ability to query multiple ICN cache nodes for the requested content is intended. Hence, the controller does not need

precise knowledge of cache content, but is enabled to query a group of caches in parallel.

Controller redundancy A viable redundancy concept is required. A scheme to, for instance, group multiple physical controllers into one distributed logical controller such that in case of a controller failure the network remains operational.

4.2 Initial approach

In the following we are going to introduce our initial approach by describing the basic mode of operation in Section 4.2.1. Subsequently Section 4.2.2 is concerned with the integration of the ICN-SDN into the network. In Section 4.2.3 and 4.2.4 the request and response processing is detailed. The utilization of the ICN-SDN as a transit network is covered in Section 4.2.5 before Section 4.2.6 finally closes this section with a discussion about this initial approach and the achieved objectives.

4.2.1 Mode of Operation

The initial approach builds up on a division of the network addressing, comparable to MPLS [47]. The addressing scheme is split into a network internal and a network external part. Inside the SDN an addressing scheme is used that the SDN forwarding elements are able to process while the externally addressing relies on the common ICN forwarding atop the convergence layer of TCP or UDP.

For the SDN internal forwarding, protocols are used that are already supported by the SDN devices. In our case, we misuse IP and transport layer header fields to carry the identifier (Message IDs (MsgIDs)) that determine the packet forwarding paths. To be able to do so, an abstraction layer is introduced that maps internal to external addresses. In fact, content names are mapped to MsgIDs. The controller determined MsgID value is assigned to the actual ICN packet at the ingress node into the ICN-SDN. Subsequently the packet is forwarded along the MsgID dependent path. Hence, not only basic packet header matching capabilities of the SDN forwarding nodes are required but also packet header rewriting capabilities. As a result, our ICN-SDN approach is capable of explicitly establishing edge-to-edge forwarding paths between pairs of ICN nodes while taking advanced ICN and network information into account.

SDN nodes and the ICN cache nodes are thereby separated entities and considered as such in the future course of this work. The controller is only able to instruct SDN forwarding elements on how to forward certain packets. Consequently all established paths only traverse SDN nodes. ICN and SDN nodes can of course be deployed co-located with each other but as a result, intermediary on-path caches do not inherent

exist on paths within the ICN-SDN. ICN cache nodes have to be explicitly included in the network forwarding process and appear as a communication end-point from the SDN perspective.

Our approach is further not specifically designed for a particular ICN implementation, it just relies on a small set of requirements, in terms of the protocols and network equipment in use:

ICN-aware SDN controller The SDN controller needs to be able to extract the content names of request and response packets. It also needs to maintain an ICN routing table to determine the SDN external next-hop ICN node. Furthermore, to carry the packets towards the likewise determined SDN edge nodes, the controller is also required to provision forwarding paths throughout the network via the setup of forwarding rules in SDN network elements.

ICN content names Each ICN packet transferred through the SDN network needs to carry the name of the requested content with it. The naming scheme is not of relevance for the approach but the controller must be able to determine the name of the content that is requested via each packet. Hence, the approach is applicable for flat as well as for hierarchical namespaces.

ICN protocol identifier The ICN protocol identifier is used to recognize packets that rely on our ICN-SDN approach and therefore requires explicit ICN-aware forwarding decision making via the controller. We propose to use a specific transport protocol port number to identify the relevant packets. Consequently the SDN forwarding elements need to be able to match on IP and the transport header fields.

SDN-IP prefix The SDN-IP prefix is used to attract packets of ICN nodes. Packets directed at this IP prefix are delivered towards the ICN-SDN edge. The SDN-IP is used as the next-hop identifier for ICN clients. It is further carried as the destination IP address in request and the source IP address in response packets.

Message ID (MsgID) The MsgID is used by the forwarding elements to identify the actions that have to be applied to arriving ICN packets. The MsgIDs are encoded in the regular TCP/IP header of each packet. They are determined by the ICN-SDN controller.

TCP/IP rewriting support The TCP/IP rewrite support is required since the MsgID we introduced needs to be written into the packet header of each ICN packet. To efficiently perform these actions, the rewriting has to be supported by the SDN forwarding elements without relying on the controller.

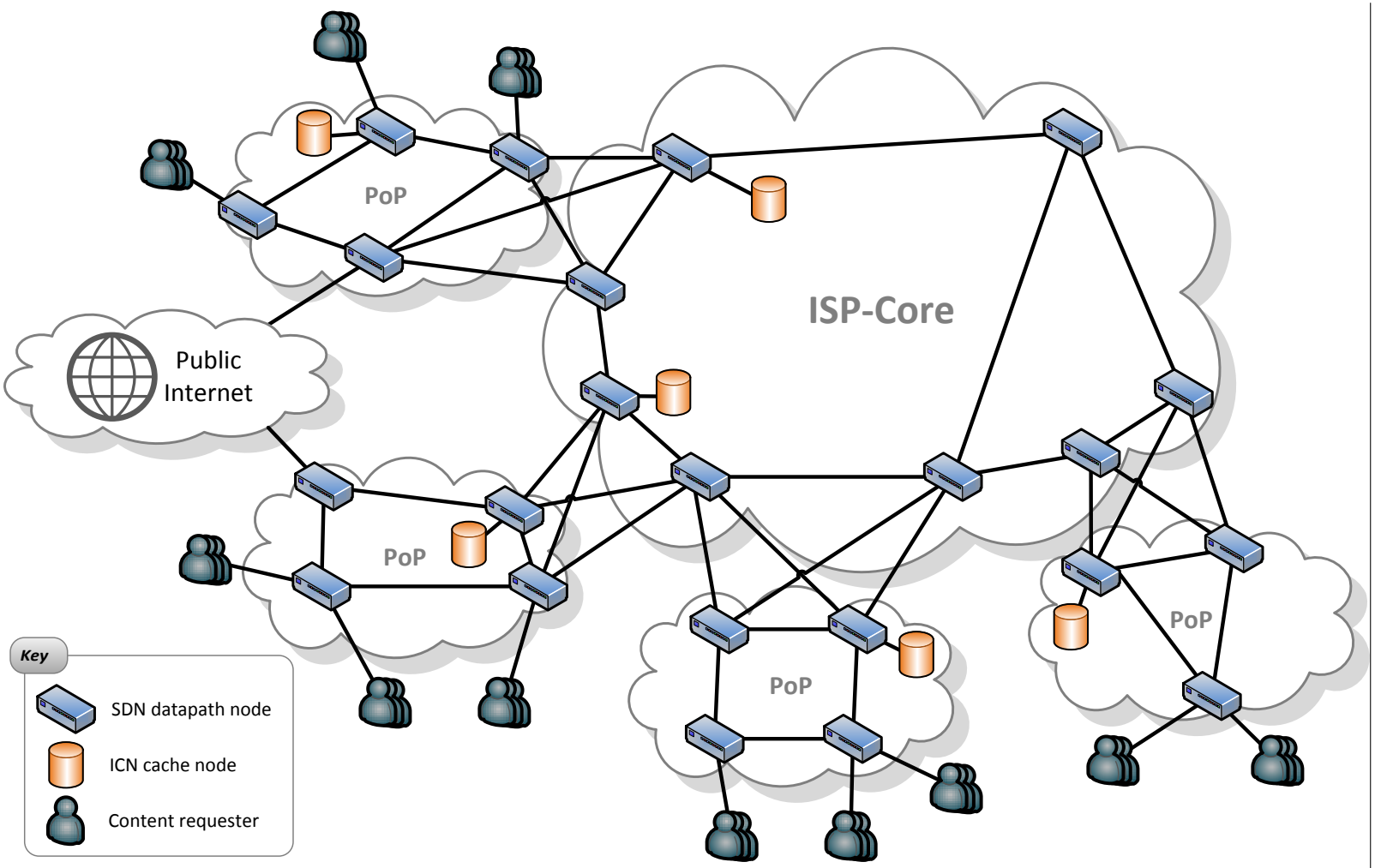


Figure 4.1: Data plane view of an example topology for the envisioned ICN enabled SDN ISP deployment

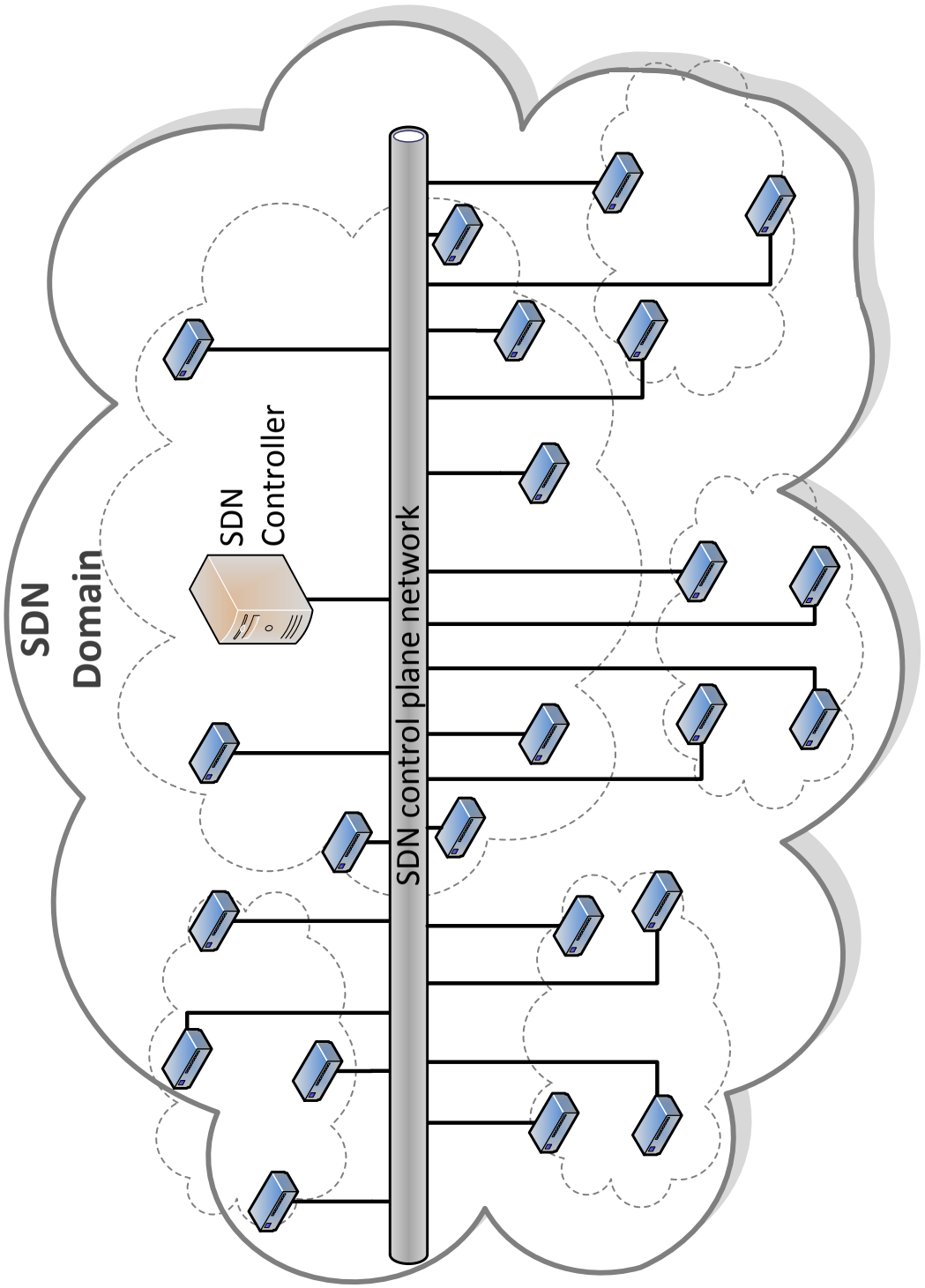


Figure 4.2: Control plane view of an example topology for the envisioned ICN enabled SDN ISP deployment

Depicted in Figure 4.1 and Figure 4.2 is a characteristical ICN-SDN deployment scenario. Figure 4.1 thereby depicts the data plane view. The network consists of a core and multiple Points of Presence (PoPs). Two of the PoPs provide connectivity to the public Internet. All of the PoPs further provide users, acting as ICN content requester with the connection to the global network. The entire network topology consists of multiple SDN datapath nodes. Some of the SDN nodes are accompanied by directly attached ICN cache nodes.

Further Figure 4.2 depicts a logical view of the SDN control plane network. This is used by the SDN forwarding nodes to communicate with the central controller deployed in the center of the ISP core network.

All the traffic created by or directed to the users is handled within the SDN domain. This applies to ICN as well as to the regular IP communication. Whenever a user triggers the creation of a content request, it is handed over to the ICN-SDN edge, more specific the SDN forwarding node that the user is connected to. The forwarding element inspects the packet and determines that it is directed to the SDN-IP. The request is subsequently handed over to the SDN controller, which further inspects the packet and extracts the content name. After performing a forwarding respectively routing lookup it is able to provision the required path through the network, either towards a particular ICN cache node, another ICN user or towards the public Internet. Finally, the instructions on how to forward the packet are send towards the ICN-SDN edge node.

This method of network integration is not mandatory. Further, if the SDN-IP prefix is announced in the IP routing, there might even be IP routers between the requester or origin node and the ICN-SDN edge. Hence, in the most extreme case, even ICN nodes from the public Internet are able to direct their traffic towards the ICN-SDN.

4.2.2 ICN-SDN network integration

For the ICN-SDN to communicate with non-SDN parts of the network, the SDN island is assigned an IP prefix. This prefix is used by the ICN nodes as the next hop identifier for packets, which are supposed to enter the ICN-SDN. Considering ISP customers or DC nodes, the entire namespace would probably be mapped to that particular prefix[1]. The actual IP prefix used by the ICN-SDN operator might thereby be distributed via a Dynamic Host Configuration Protocol (DHCP) option [48] or configured manually. In addition, a specific transport layer port is specified towards which end nodes direct their ICN requests. This combination of IP address and transport layer port is further used by the ICN-SDN edge nodes to identify packets that need special ICN treatment.

Figure 4.3 illustrates the overall operation of our approach. Depicted in the figure is an ICN host that issues content requests as well as the controller that is managing the

[1] CCNx for instance might utilize a default routing entry for `ccnx:/` pointing to the ICN-SDN IP prefix.

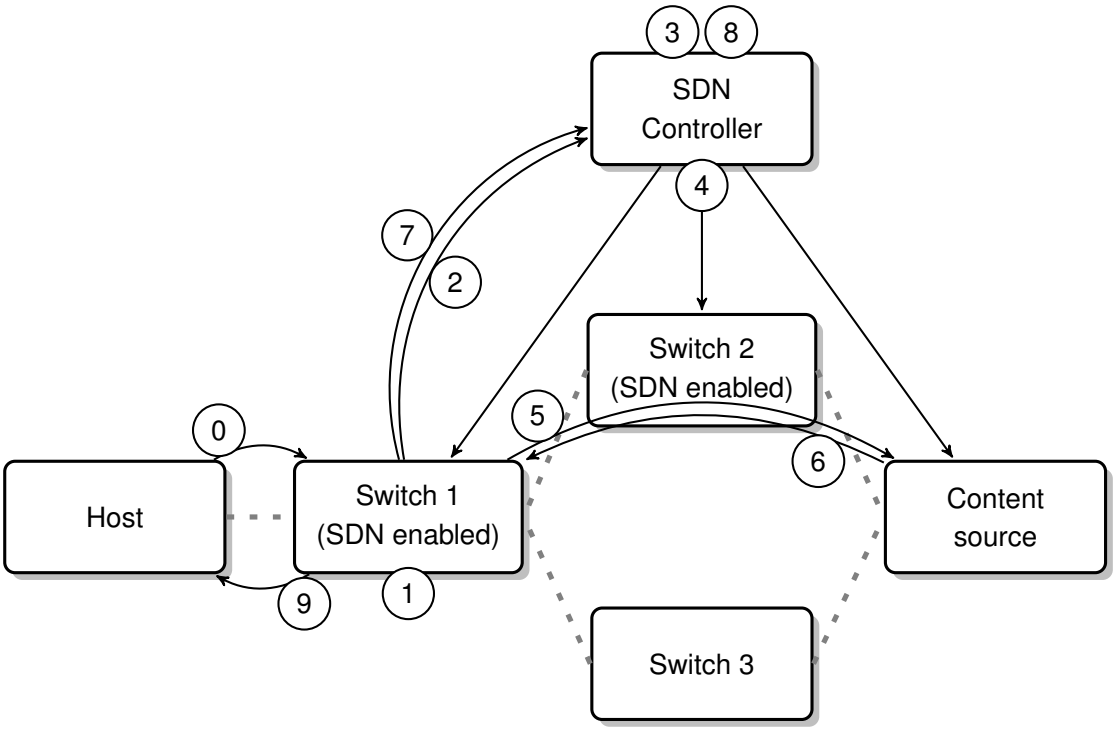


Figure 4.3: Overall ICN-SDN operation overview – adapted from [49]

SDN network elements. Furthermore on the right side an ICN node is depicted that is capable of serving requested content.

The request / response cycle illustrated works as follows:

(0) The Host issues a content request that is transmitted towards the SDN edge.

(1) The issued request arrives at the SDN ingress switch. This switch uses common SDN packet matching mechanisms and detects that the packet is an ICN request.

(2) The packet is subsequently forwarded to the SDN controller.

(3) The controller extracts the requested content name and via its ICN routing table subsequently determines where to forward the request to.

(4) As soon as the location of the requested content is known, the controller installs forwarding rules for the request, as well as for the corresponding content reply. This rule provisioning is performed on all switches that the packets have to traverse. Eventually the ingress switch is instructed by the controller to start forwarding the packet.

(5) Subsequent, the request is forwarded along the established path.

(6) The content source replies with the requested content that is forwarded along the backwards path, which was already provisioned in step (4).

(7) Switch 1 indicates the response message reception to the controller.

(8) The controller can purge all state of that particular request.

(9) Eventually the response is delivered to the requesting host.

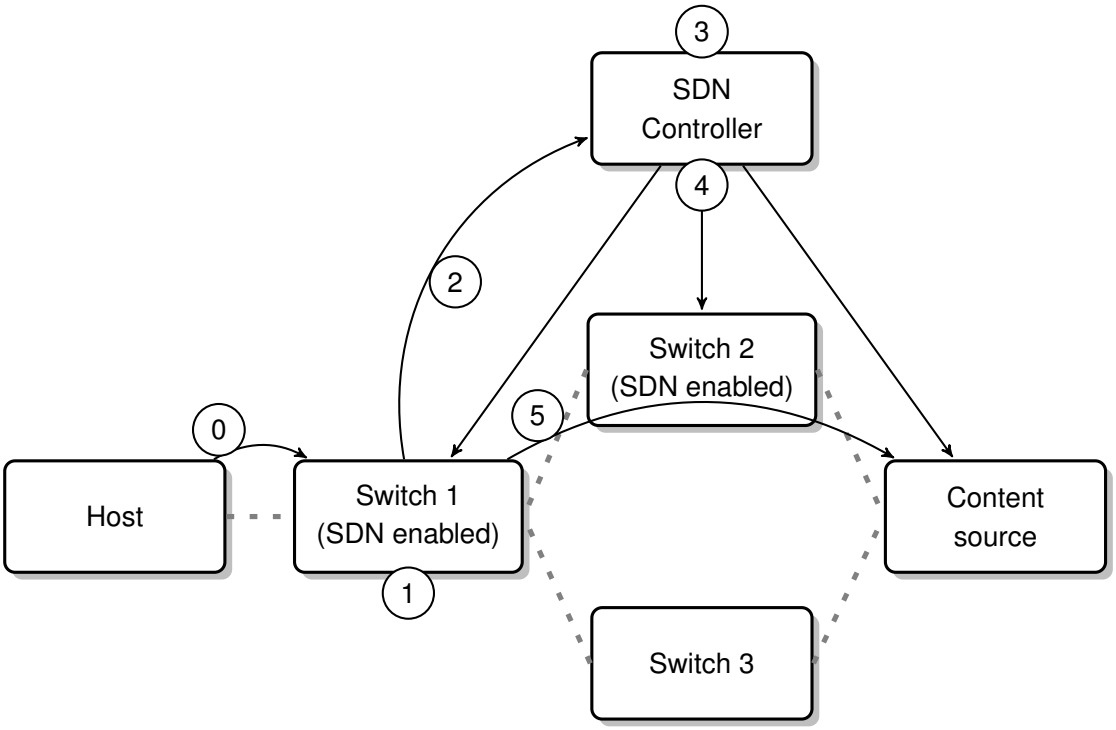
4.2.3 Detailed request processing

Figure 4.4 illustrates the request forwarding and packet header rewriting process of our approach in a more detailed manner. We assume that the SDN-IP is already known to the host such that requests can immediately be direct to the actual SDN island. (0) The depicted host creates its request packet with the parameters specified in Figure 4.4b. The request is directed to the SDN-IP and the dedicated ICN port while the source fields carry the IP and port number that the requesting hosts ICN process runs on. As the payload of the requests, the ICN implementation specific format is used, thereby carrying at least the name of the solicited content.

As soon as the first SDN enabled switch receives the ICN packet, a corresponding forwarding entry is looked up in the flow table. The packet matches on the pre-configured flow rule for SDN-IP and ICN port as the destination header fields. (2) The associated action prompts the switch to send the packet towards the SDN controller. (3) Upon the receipt of the packet the controller will examine it and extract the name of the requested content. Subsequently, a source for the solicited content is looked up, a path through the SDN part of the network is calculated and a MsgID is temporarily assigned. The association between MsgID and the source IP as well as the transport protocol port combination is preserved on the controller for further lookups. Next, the packet rewriting instructions are constructed. The source IP equals the determined MsgID and the source port is set to the Internal Routing Identifier (IRI) of Switch 1. The corresponding destination fields are set to the IP address and port number of the selected content source.

The IRI is a SDN internal identifier that uniquely identifies a SDN forwarding element. It is used to aggregate response paths through the SDN and thereby reduces the amount of provisioned flow rules. Response messages are routed towards the egress forwarding element using the IRI value previously attached to the ICN request. The detailed process of the response forwarding is presented in Section 4.2.4.

In step (4) the rewriting rule, as well as the basic forwarding rules, are installed on the corresponding forwarding elements. The basic forwarding rules match on the destination IP and ICN port of the content source. Using the actual IP address of the content source



(a) ICN request forwarding visualization

0	Source	Destination
IP	*Host-IP*	*SDN-IP*
Port	*Host-Port*	*ICN-Port*

(b) Header of packet as send by requester

5	Source	Destination
IP	*MsgID*	*Content source*
Port	*IRI(Switch 1)*	*ICN-Port*

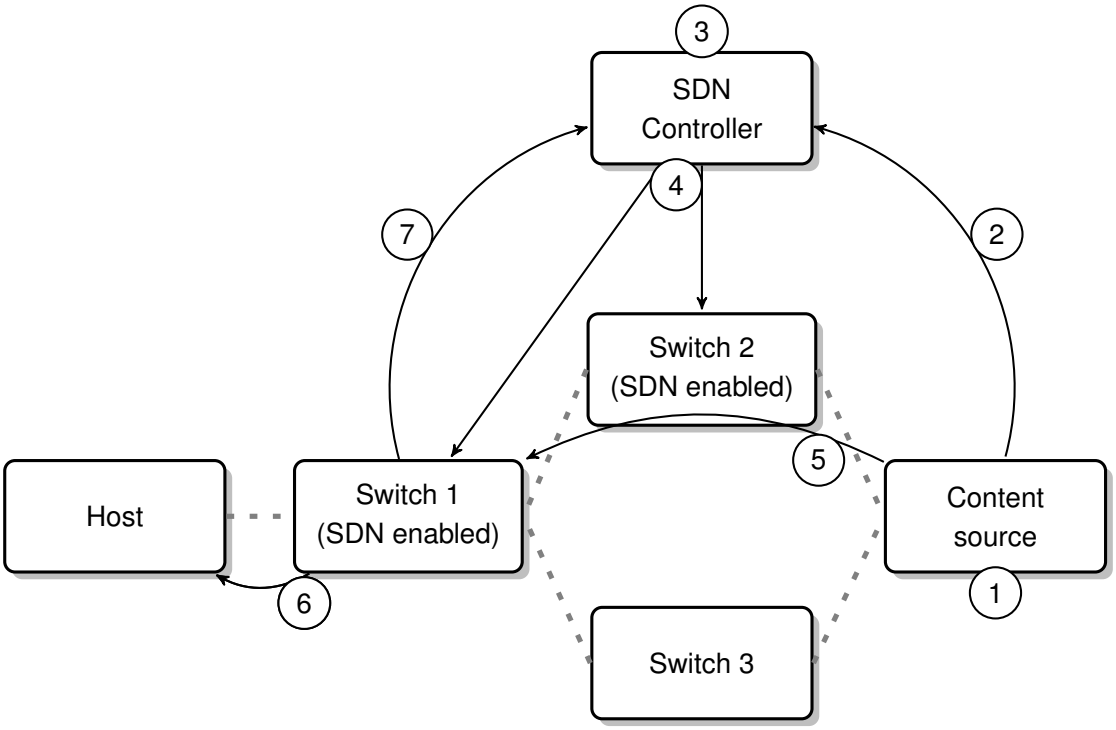
(c) Header of packet after SDN ingress rewrite

Figure 4.4: ICN-SDN request processing and forwarding

as the destination value would now also allow for the delivery of the request packet through the non-SDN part of the network (Switch 3). Eventually at the end of step (5) the content request arrives at the content source where the ICN process handles the request.

4.2.4 Detailed response processing

Figure 4.5 visualizes the process of response forwarding that is performed when the previously requested content is transferred through the SDN.



(a) ICN content forwarding visualization

1	Source	Destination
IP	*Content source*	*MsgID*
Port	*ICN-Port*	*IRI(Switch 1)*

(b) Header of response packet issued by content source

6	Source	Destination
IP	*SDN-IP*	*Host-IP*
Port	*ICN-Port*	*Host-Port*

(c) Header of response packet after rewrite operation of Switch 1

Figure 4.5: ICN-SDN response processing and forwarding

In step (1) the content source performs the default TCP/IP processing by swapping

source and destination ports as well as IP addresses of the former received content request to construct the response packet (see Figure 4.5b). In Figure 4.5a the content source is already SDN-enabled. (2) The packet matches the source port rule (*source port = ICN-Port*), which prompts the first SDN-enabled node – in this case the content source itself – to direct the packet to the controller. However, if the first SDN node already received a more specific rule from the controller that matches the MsgID included in the response packets the packet is processed according the associated forwarding actions (see step (5)). (3) The controller inspects the received packet and looks-up the MsgID via the name field of the ICN packet to determine the egress forwarding element including the physical egress port. In step (4) the controller deploys the determined flow rules to the switches that form the path from content response ingress to the response egress switch.

All switches, except the egress switch are configured to forward the packet according to the IRI, which is contained in the destination port of the response packet. The SDN egress switch is configured to perform the last header rewrite operation (6). The source IP and port is finally set to the parameter values that the request was directed to. Further the destination fields are set to match the source fields of the initial request (see Figure 4.5c). Doing so, the ICN-SDN appears like a general TCP/IP node that performs regular source / destination field swapping when sending packets back and forth. The content is subsequently delivered to the requesting host and finally the content egress switch informs the controller of the finalized content delivery. This might for instance be triggered through the expiration of an idle timer, which is attached to the flow rule. After the controller knows that the MsgID is no longer actively in use for content transmission, the associated state is removed and the controller can make use of the MsgID for subsequent content requests.

Since we included a legacy switch it is worth noting that the response forwarding in contrast to the request forwarding is impossible to be performed through the non-SDN parts of the network. The response has to traverse the SDN because the destination IP field contains the arbitrarily selected MsgID that is not compatible with basic IP forwarding.

4.2.5 Transit ICN-SDN deployment

When ICN nodes are not directly connected to the SDN, our approach is still applicable but requires additional packet rewriting steps. The request egress node has to change the source IP address of the outgoing request to the IP of the SDN. Thereby the content packet will arrive at the ICN-SDN after the content source performed the IP source / destination field swapping. The field values of the packet are transferred trough the SDN, as shown in Table 4.1a. Table 4.1b depicts the values after the request packet

	Source	Destination
IP	*MsgID*	*Content source*
Port	*IRI(Switch 1)*	*ICN-Port*

(a) Header of request packet within the SDN

	Source	Destination
IP	*SDN-IP*	*Content source*
Port	*IRI(Switch 1)*	*ICN-Port*

(b) Header of request packet when leaving the SDN

Table 4.1: Transit request packet rewriting

	Source	Destination
IP	*Content source*	*SDN-IP*
Port	*ICN-Port*	*IRI(Switch 1)*

(a) Header of response packet when entering the SDN

	Source	Destination
IP	*Content source*	*MsgID*
Port	*ICN-Port*	*IRI(Switch 1)*

(b) Header of response packet within the SDN

Table 4.2: Transit response packet rewriting

left the ICN-SDN. Furthermore, Table 4.2 shows the same header fields for the content packet that is issued in reply to the request. Listed in Table 4.2a are the values the packet carries when residing outside the borders of the SDN while Table 4.2b represents the header after re-entering the ICN-SDN.

One might perceive that there is no unique mapping from the SDN-IP in the response packets destination IP in Table 4.2a to the MsgID in Table 4.2b. The packet is doomed to be inspected by the SDN controller to establish this mapping. It extracts the content name from the packet and further determines the MsgID the packet belongs to.

4.2.6 Discussion

Through our basic approach, we are now able to utilize SDN as a supportive technology for the enhanced deployment of ICN. We decomposed the problem into the smaller components of mapping the external addressing scheme to an internal one and back, as well as the internal forwarding itself. The ICN-SDN controller is able to establish paths through the network that solely rely on IP and transport layer header information. The established paths through the network are optimized for ICN forwarding, since the controller is aware of the network condition and at the same time takes information deducible from the ICN protocol layer into account. However, we still do not meet all the requirements introduced in the the beginning of this chapter.

The requirement of supporting the different ICN packet forking mechanism is not met by this initial approach. In case of response forking or request aggregation the utilization of the IRI entails that packets are always delivered to the request ingress SDN node, fur-

ther no packet duplication takes place within the ICN-SDN. To effectively perform these response forking actions, individual content needs to be identified when entering the ICN-SDN. Since ICN nodes can issue multiple requests at the same time, it is important to be able to distinguish different responses to subsequently apply diverse actions. This is not envisaged in this basic approach. It is only clear where packets have to be delivered and how the headers have to be rewritten when leaving the ICN-SDN to reach the requester. Beyond that, the processing in case that the ICN-SDN is utilized as a transit network is suboptimal. It is not that only requests have to be inspected by the controller also responses have to be examined to be successfully mapped to the content requests and thereby determine the outbound parameters.

Finally, also the duplication of request packets is not yet provided along with desirable mechanisms to suppress the duplicate delivery of content through the SDN network. To handle these outstanding requirements and correct the shortcomings, we revise the initial and thereby develop an advanced approach, which is detailed in the following section.

4.3 Advanced approach

To further benefit from operating an ICN over SDN, the utilization of packet forking is proposed. ICN forking enables the duplication and delivery of ICN packets through the SDN network. This forking treatment can be applied to request as well as to content packets. Allowing for the aggregation of multiple equal concurrent requests, for the simultaneous filling of caches while delivering content to the requester or the querying of multiple cache nodes in parallel.

We can further observe that the ICN-SDN needs to be able to uniquely map each arriving content to its corresponding request packet to efficiently perform the required forwarding actions. In the basic approach, without the explicit support of the controller, we are only able to identify the requester of an arriving content packet since the request is tied to a requester dependent MsgID. But for the sake of scalability, we want to achieve this mapping without or with the least possible additional controller involvement.

4.3.1 ICN packet forking use-cases

For the future course of our analysis, we stick to particular scenarios that we are going to describe after introducing the design space for the deployment of forking enabled ICN-SDNs:

4.3.1.1 Design space

The design space of the ICN packet forking approach is dividable into different areas as delineated in the following.

Response forwarding strategy We anticipate two response forwarding strategies for the ICN-SDN request forking approach:

Delay Optimized (DO) If all flow rules for the content responses, including those of the egress node are pre-provisioned, the response is directly send to the requesting node. No further controller interaction is required. However, if the SDN nodes send a notification about the forwarded data to the controller, it is up to the controller to decide which response paths to prune. This decision eventually results in the provisioning of drop rules on all other than the selected egress node and thereby prevents multiple subsequent deliveries. No additional delay is introduced when applying this mechanism. Responses are delivered as quickly as possible towards the requesting node, but multiple deliveries of the same content chunks are possible at least until the drop rules are installed by the controller.

Bandwidth Optimized (BO) If the egress switches are not already provisioned with rules for the forwarding of content responses, the ICN-SDN nodes will query the controller on how to handle the response packet. It is thus again up to the controller to decide which response to deliver to the consumer node while fully preventing all other responses from being propagated through the network. A viable strategy for this decision process might be to use the node which replies fastest. All additional responses can be discarded via drop rules already at the edge of the ICN-SDN. This mode, of course, implies an increased processing load for the ICN-SDN controller but prevents the network from waisting valuable core bandwidth.

Cache knowledge The controllers level of knowledge about the cached content also influences the usefulness of the forking approach. Its knowledge might range from full over partial – also in different nuances – to no cache content knowledge. If the controller has full knowledge about each ICN cache nodes content, no forking would be necessary. The controller would always know if and where the requested content is cached. It would be able to select the best cache for the content at each time, whereas best is not rigidly defined. It can be network proximity wise closest, least loaded, bandwidth-wise favourable or the like. However, this mode of operation potentially adds a high burden on the controller. It would have to be notified about all cache modifications like addition and eviction actions. This information exchange will become increasingly extensive depending on the network size and

utilization. In the partial knowledge category, the controller only has a basic notion of where content might be cached, instead of being sure where exactly, or even if it is cached at all. This case occurs when the controller for instance only tries to learn cached content by observing the ICN traffic, or when groups of available caches coordinate the caching among themselves without keeping the controller informed. Hence, forking ICN requests might be beneficial when the controller has no knowledge about cache mapping, it might also be worth choosing multiple caches randomly or by querying all available caches, depending on the cache node count and the network load.

Forking cases For the forking strategy also different approaches exist. The forking approach might be applied to particular content, namespaces or content types. Another forking strategy might be to only fork initial requests of coherent pieces of content until the controller is able to decide which ICN node is the preferred one to serve the remaining content and subsequently truncate all dispensable forks. Additionally a combined strategy could be utilized to perform the request forking of initial requests just for certain namespaces or content types.

4.3.1.2 Considered scenarios

Considering the above mentioned design space, we focus on the case in which ICN caches are deployed within the SDN domain. In general, we do not expect the controller to have any in-depth cache knowledge, since this would add high processing overhead for quickly changing cache content and thereby reduce the scalability of the approach. Hence, we assume that the controllers knowledge is limited to only the existence and location of ICN cache nodes.

Specifically we envision the following two cache constellations:

Explicit cache No content mapping function exist or is known to the controller. The caches are deployed within the SDN domain close to the ICN-SDN edge, as well as next to particular core SDN nodes. The controller will determine cache nodes of the ICN-SDN network through some sort of heuristic and directed requests immediately towards these caches, in hopes the content is available in any of the selected caches. Therefore the request is forked and forwarded towards each of the candidate caches. As soon as one of the caches replies, the controller can truncate all other content replies of the remaining caches and further only utilize that particular cache to direct all remaining requests to. This mechanism pays of best in cases in which the content is large and only the initial requests are forked to determine a cache that is able to answer the request.

Load-balanced cache The content chunks of a particular piece of content are spread over different cache nodes such that each of the x cache nodes holds $\frac{1}{y}$ of the content chunks, with $x \geq y$. By forking the requests, all nodes receive the requests for all chunks. Just those nodes that have the content chunks available reply with the content. By doing so, a balanced distribution of load is achieved across the involved cache nodes.

This approach requires only slight additional knowledge of the controller. Only a mapping of content to a cache group is needed. Furthermore, the added overhead for duplicated requests is tolerable since requests are orders of magnitudes smaller than the content responses.

4.3.2 Request forking

Since in the ICN paradigm, content is not bound to any specific host, different network nodes may be able to provide the data that is requested by the content consumers. It can thus be beneficial to fork content requests and query multiple possible ICN nodes to provide the content. Figure 4.6 illustrates such forking scenarios. The Requester (R) issues a content request that the controller decides to deliver towards the three Content Origins (C). In the first instance, the request is thus delivered towards the Fork node (F), which then duplicates the request and sends it off towards the three Content Origin nodes. Subsequently non to all nodes might respond to the request, depending on their cache contents. The requester will then dependent on the response forwarding strategy receive non, one or multiple responses.

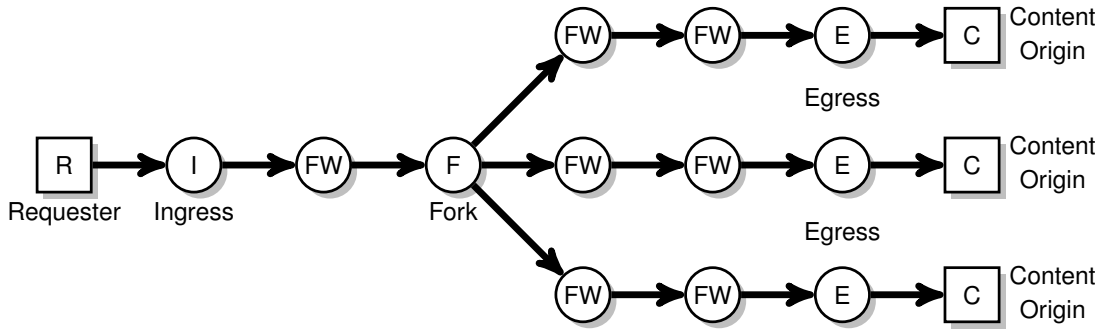


Figure 4.6: Possible ICN-SDN request forking scenario

4.3.2.1 Mode of operation

In the following the mode of operation of the request forking mechanism is discussed in detail. The roles that ICN-SDN nodes might act as are introduced in the first step. Finally the packet matching and rewriting workflow is shown.

Node roles Like illustrated in Figure 4.6 our approach relies on different roles that the ICN-SDN nodes perform. They might differ for each content dissemination. Furthermore, these roles are also non-exclusive such that in the extreme case all roles can be performed by a node at once.
The roles we identified are:

Ingress (I) Ingress nodes are nodes through which requests enter and responses leave the ICN-SDN island.

Forwarding (FW) Forwarding nodes are nodes that forward the ICN packets throughout the network according to the switch-id that is carried as part of the packet.

Egress (E) Egress nodes are nodes through which the requests leave and the content response subsequently enter the ICN-SDN island.

Forking (F) Forking nodes are nodes that duplicate request or response packets and send them towards multiple different next hop nodes.

Each switch is assigned at least one unique switch-id, maybe more if the amount of 2^{16} MsgIDs is insufficient. These switch-ids are carried in the IP fields of network packets within the ICN-SDN domain to indicate a packets destination. Destination in this case refers to a node that has to apply special processing to a packet, like the Fork, Ingress or Egress node. To not interfere with publicly routed IP addresses, we encourage using the private IP address space as introduced in [50] for the switch-ids.

Matching and rewriting process Table 4.3 and 4.4 illustrate the packet header values the actual ICN packets carry when they are forwarded through the ICN-SDN. The caption indicates between which type of nodes the packet carries the particular values, while the highlighted cells indicate the header fields that the subsequent nodes provisioned flow rules match on.

Interest packets, when issued by the requesting node, carry the SDN-IP and the default SDN port as the destination identifier (Table 4.3a). The ingress switch is performing its packet matching on these ICN-SDN specific values (highlighted cells). By this measure it identifies if the packet is supposed to be handled via the special ICN processing or via general bridging mechanisms. The source fields reflect the IP address of

	Source	Destination
IP	*R-IP*	*SDN-IP*
Port	*R-Port*	*SDN-Port*

(a) Packet header R → I

	Source	Destination
IP	*SW-ID(F)*	*SW-ID(F)*
Port	*MsgID2*	*MsgID2*

(b) Packet header I → FW; FW → FW

	Source	Destination
IP	*SW-ID(F)*	*SW-ID(F)*
Port	*MsgID2*	*MsgID2*

(c) Packet header FW → F

	Source	Destination
IP	*SW-ID(E_x)*	*SW-ID(E_x)*
Port	*MsgID3*	*MsgID3*

(d) Packet header F → FW; FW → FW

	Source	Destination
IP	*SW-ID(E_x)*	*SW-ID(E_x)*
Port	*MsgID3*	*MsgID3*

(e) Packet header FW → E_x

	Source	Destination
IP	*SDN-IP*	*C_x-IP*
Port	*MsgID4*	*C_x-Port*

(f) Packet header E_x → C_x

Table 4.3: Request forking – header rewriting for request forwarding – highlighted cells indicate fields the subsequent node performs matching on

the requester and the transport layer port the ICN daemon operates on. At the ingress node into the ICN-SDN, the packet is rewritten as depicted in Table 4.3b. Source and destination IP addresses carry the switch-id of the forking switch. The two port fields carrying the MsgID2 are set to a fork node dependent, unique MsgID value. Packets are forwarded by the ingress and forwarding nodes according to this switch-id towards the fork node. The fork node identifies packets that it has to handle in a special way by matching on its own unique switch-id in the source and destination IP fields of the packet. The controller provisions a flow entry that, besides matching the switch-id, also matches the particular MsgID2 to distinguish between different flows that require different actions (Table 4.3c). This forwarding and forking mechanism is allowed to happen multiple times at multiple nodes throughout the network. For the simplicity of illustration we just conduct the forking once. After being forked each request is forwarded (Table 4.3d) to its egress node in the same fashion, see Table 4.3b. The egress switch-ids are used to guide the packet towards each egress node (E_x). The combination of SW-ID(E_x) and MsgID3 is again unique for the particular content transfer. The flow rules installed on the egress switch (Table 4.3e) lead to the required rewriting and forwarding action such that the packet is subsequently delivered towards the content origin / cache (C_x), like depicted in Table 4.3f.

Table 4.4 illustrates the packet rewriting in case of content response forwarding. The

egress switches perform matching on the content origin IP address for the source, as well as the SDN-IP and the MsgID4 for the destination fields. In this way, the content responses can be uniquely mapped to the flow rules that form the path back to the requesting node (Table 4.4b). The egress node identifies the action to perform by matching for its own switch-id and the MsgID1 to identify the appropriate flow rule (Table 4.4c). This flow rule causes the switch to adjust the header fields for the packet to be delivered to the requester, like illustrated in Table 4.4d.

	Source	Destination
IP	*C_x-IP*	*SDN-IP*
Port	*C_x-Port*	*MsgID4*

(a) Packet header C $\rightarrow$ E

	Source	Destination
IP	*SW-ID(I)*	*SW-ID(I)*
Port	*MsgID1*	*MsgID1*

(b) Packet header E $\rightarrow$ FW; FW $\rightarrow$ FW

	Source	Destination
IP	*SW-ID(I)*	*SW-ID(I)*
Port	*MsgID1*	*MsgID1*

(c) Packet header FW $\rightarrow$ I

	Source	Destination
IP	*SDN-IP*	*R-IP*
Port	*ICN-Port*	*R-Port*

(d) Packet header I $\rightarrow$ R

Table 4.4: Request forking – header rewriting for response forwarding – highlighted cells indicate fields the subsequent node performs matching on

4.3.3 Request aggregation / response forking

Request aggregation describes the capability of merging content requests in the network. Multiple requesting nodes issue content requests that are subsequently recognized as requests for the same content. They are subsequently forwarded only once towards an ICN node. The ICN-SDN approach thereby overcomes the limitation of providing only on-path and allows for network wide request aggregation.

Response forking can thereby be seen as a sub-case of request aggregation in which the controller decides which cache to populate with the content without prior reception of an additional content request from that node.

A sample process of forking and forwarding the requested content is depicted in Figure 4.7. A request is issued by the Requester (R) in the left half of the picture. SUbsequently the controller either received an additional request from the Requester (R) in the upper right corner, which is aggregated with the previous one or decides itself that the Cache (R) should receive the content. As a result, it adjusts the forwarding rules such that the provisioned forwarding paths ensure that the packets are forwarded to the

Response Fork node (F), which duplicated the packets and sends them on towards the left Requester (R) and the Cache / Requester (R) in the upper right corner.

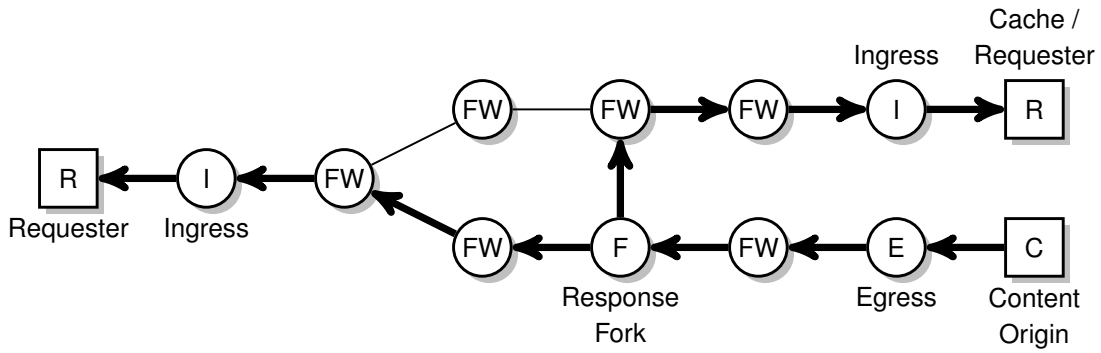


Figure 4.7: Possible ICN-SDN response forking scenario

4.3.3.1 Mode of operation

Table 4.5 illustrates the different packet header modifications of the request packet with the successor nodes match fields indicated via the highlighted cells.

The ingress SDN node receives the packet directed at the SDN-IP and the SDN-Port with the corresponding source fields of the requester. The switch matches on the SDN-IP and the SDN-Port and thus identifies that the packet is designated to explicit ICN-SDN forwarding (Table 4.5a). After the controller has determined the egress switch and provisioned the egress rule, the packet is forwarded through the SDN network. This forwarding is performed according to the switch-id of the egress switch (Table 4.5b). Arrived at the egress node, the egress node matches on its own switch-id as well as the MsgID2 to determine the rewriting rule it has to apply to the packet (Table 4.5c). Eventually the packet is send towards the content origin with the specific ICN-SDN values as the source parameters, as depicted in Table 4.5d.

After the request packet arrived at the content origin node, it replies with the actual content response. It is not relevant for the processing if the flow rules for the response paths are already provisioned at the time of request provisioning or just when the response packets arrive at the SDN edge. In the following, we assume that the response paths already exist.

The response arrives at the ICN-SDN edge with the SDN-IP and the MsgID3 as destination fields (Table 4.6a). The match condition comprises of the SDN-IP, the MsgID3 as well as the content origin source IP. The edge switch rewrites the packets IP fields with the switch-id of the request forking switch and the forking switch dependent MsgID4

	Source	Destination
IP	*R*	*SDN-IP*
Port	*R-Port*	*SDN-Port*

(a) Packet header R → I

	Source	Destination
IP	*SW-ID(RE)*	*SW-ID(RE)*
Port	*MsgID2*	*MsgID2*

(b) Packet header I → FW; FW → FW

	Source	Destination
IP	*SW-ID(RE)*	*SW-ID(RE)*
Port	*MsgID2*	*MsgID2*

(c) Packet header FW → E

	Source	Destination
IP	*SDN-IP*	*C-IP*
Port	*MsgID3*	*C-Port*

(d) Packet header E → C

Table 4.5: Response forking – header rewriting for request forwarding – highlighted cells indicate fields the subsequent node performs matching on

as the transport layer port values (Table 4.6b). The fork switch identifies its own switch-id in the packet and thus also takes the MsgID4 into account, to identify the special packet rewriting and duplication flow rule it has to apply (Table 4.6c). The packet is further forwarded to each edge node (I_x) that has to deliver the packet towards a requester or cache (R_x)(Table 4.6d). All edge nodes match on their switch-id as well as the dependent MsgID5, as soon as the packet arrives (Table 4.6e). The edge nodes then rewrite the packet so that the response can be delivered to the actual requesters or caches (Table 4.6f).

4.3.4 Flow entry count per SDN switch

One of the limiting factors in SDN driven networks is the amount of flow rules a SDN node is able to handle. Therefore, we analyse the amount of flow rules that need to be installed for our approach to function. Following, we will separately analyse the different roles an ICN-SDN node can perform and what impact each role has on the amount of flow table entries.

To shape our formulas, we introduce the following variables that the amount of switch flow rules rely on.

α

Amount of concurrent[3] content requests per unique ICN client node.

β

Fork factor that defines which fraction of the concurrent[3] requests are forked.

[3] Active or not already timed out requests

	Source	Destination
IP	*C-IP*	*SDN-IP*
Port	*C-Port*	*MsgID3*

(a) Packet header C → E

	Source	Destination
IP	*SW-ID(F)*	*SW-ID(F)*
Port	*MsgID4*	*MsgID4*

(b) Packet header E → FW; FW → FW

	Source	Destination
IP	*SW-ID(F)*	*SW-ID(F)*
Port	*MsgID4*	*MsgID4*

(c) Packet header FW → F

	Source	Destination
IP	*SW-ID(I_x)*	*SW-ID(I_x)*
Port	*MsgID5*	*MsgID5*

(d) Packet header F → FW; FW → FW

	Source	Destination
IP	*SW-ID(I_x)*	*SW-ID(I_x)*
Port	*MsgID5*	*MsgID5*

(e) Packet header FW → I

	Source	Destination
IP	*SDN-IP*	*R_x-IP*
Port	*SDN-Port*	*R_x-Port*

(f) Packet header I → R

Table 4.6: Response forking – header rewriting for response forwarding – highlighted cells indicate fields the subsequent node performs matching on

γ

$$\gamma = \begin{cases} 0, & \text{if all incoming chunk requests need to be processed by controller.} \\ 1, & \text{if SDN ingress switch is able to identify sets of chunk requests.} \end{cases}$$

$\#AN$

Number of ICN-SDN associated ICN nodes.

$\#SN$

Amount of SDN enabled nodes within the ICN-SDN island.

In the following steps, we are going to create a formula that describes how many flow rules will have to be maintained by an SDN node, taking part in our ICN-SDN forking approach.

Due to the complexity, which partly result from the dependence on a large parameter space, we have to simplify the general assumptions we draw for the overall operation of the system. Hence, this approximation describes the upper boundary of the number of flow rules in which all requests that reach the ICN-SDN are processed by only one particular ICN-SDN node.

$$FR_I = (\alpha * \gamma + 1) * \#AN \tag{4.1}$$

The amount of flow rules installed on the ingress node dependents on the amount of requests that it serves as the first hop node within the ICN-SDN. For each requester, the ingress node needs a rule defining the packet rewriting operations that are to be performed for the content responses. Dependent on the ICN protocol, it may be possible to identify coherent subsequent chunk requests. In this case ($\gamma = 1$), rules for each active content request ($\alpha * \#AN$) will be deployed to the ingress. While increasing the amount of flow rules, the general control plane load is dramatically reduced, which is beneficial for the overall scalability of the system.

$$FR_{FW} = \#SN \tag{4.2}$$

Forwarding nodes only need to forward packets according to the switch-ids, which are associated to ICN-SDN enabled network nodes. Hence, the number of flow rules is limited by the number of SDN nodes that reside within the particular ICN-SDN island.

$$FR_E = 2 * \alpha * \#AN \tag{4.3}$$

Egress nodes will be provided with two flow rules per each content request that leave the ICN-SDN island via them. The first rule describes the packet rewriting that needs to be applied to the outgoing request packet, the second rule is used to subsequently handle the incoming response.

$$FR_F = \alpha * \beta * \#AN \tag{4.4}$$

Forking nodes, which are in fact special forwarding nodes, need a particular rule for each ICN request or content response that they are supposed to fork. This is approximated by the amount of concurrent requests in the network ($\alpha * \#AN$) along with the expected forking factor (β).

By applying the above formula in the basic operation case, we expect the amount of flow rules per SDN node to be limited as follows:

$$FR_{\text{(basic operation)}} = FR_I + FR_{FW} + FR_E \tag{4.5}$$

$$FR_{\text{(basic operation)}} = \#AN + 3\alpha * \#AN + \#SN \tag{4.6}$$

If we also consider the request forking case in our estimation, we end up with the following formula.

$$FR_{\text{(request forking)}} = FR_{\text{(basic operation)}} + FR_F \tag{4.7}$$

$$FR_{\text{(request forking)}} = (1 + 3\alpha + \beta * \alpha) * \#AN + \#SN \tag{4.8}$$

Assuming the following relation of the parameters $\#SN < \#AN$ and β being less than 1, it is obvious that the role with the highest demand in flow rules is the egress role (Formula 4.3) and in certain cases also the Ingress role (Formula 4.1).

As indicated above, the resulting formulas describe the worst case scenario in which all functions are performed by only one node. Egress and ingress roles will likely be mixed together. Nevertheless, we are able to spilt up these roles to a certain extend. Requester facing nodes are more likely to act as ingress nodes, whereas nodes at interconnection points are to some extend configurable via the controller and the ICN routing protocol to act as ingress or egress nodes and for which namespaces they do so.

By these measures, the amount of flow rules that today's OpenFlow hardware switches are able to handle are sufficient as a starting point. For instance the NEC ProgrammableFlow switch PF5240 supports 64K-160K (Maximum) flow entries [51]. This eventually results in 32K-80K concurrent content transfers that can be handled by one egress ICN-SDN node.

4.3.5 Cost estimation

To further assess the applicability of the forking approach we evaluate the complexity in terms of resulting network load and the amount of exchanged control and data plane messages, in what follows.

4.3.5.1 Assumptions

Since we are using CCNx as the basis for our implementation, we consider that consecutive request packets that belong to a related content request are not distinguishable by the SDN forwarding elements. Consequently, to decide how the packets should further be processed, each chunk request has to be delivered to the controller.

For our estimation, we further assume that request and response paths are symmetrical such that all response paths can be pre-provisioned along with the request paths. This also implies that request egress nodes are assumed to also serve as the ingress nodes into the ICN-SDN for the content packets.

The content ingress nodes are not populated with the necessary response forwarding rules. Following the previously mentioned BO case, the controller has to be informed about each incoming content packet. Based on this notification, it is up to it, to reactively provision flow rules to forward the traffic of one of the SDN switches, while at the same time providing the other switches with rules to discard the corresponding packets.

To keep the analysis manageable, we also assume that all content requests are forked just once throughout the network so each resulting branch has the same length and thus consists of the same amount of hops. Figure 4.6, for example, depicts a constellation which meets these assumptions. The analysis is meant to compare packet forking in the ICN case to packet forking in the SDN case. To preserve the comparability, the ICN topology is assumed to look the same like the ICN-SDN topology, meaning that the ICN packets have to traverse the same amount of hops in both cases.

Throughout the process of building a (simplified) model for the packet forking approach, we assume that the generic switch-id dependent forwarding rules have to be installed each time a request for content is issued. This represents the worst case of operation, since for normal forwarding operations through the network we rely on switch-ids that identify the node that have to apply special treatment to the packet. This special treatment may be that the packet needs to be forked by that node or the node acts as the ingress or egress node and has to rewrite the packet accordingly.

4.3.5.2 Request forking

Besides the controllers limitation to examine a restricted amount of incoming packets, which can probably be scaled by utilizing a logical controller instance out of multiple physical nodes, the amount of exchanged packets between controller and the switches (Packet_In, Packet_Out and Flow_Mod) also restricts the applicability of our approach. The switches need to process the OpenFlow messages and adjust their TCAM configuration accordingly. In the following, we will especially assess the data plane load to estimate the required network capacity for the request forking approach.

Load per content request To determine the load that our request forking approach creates, we will now build a rough analytical model for the ICN-SDN as well as for a pure ICN forking case. By doing so, we are able to determine under which conditions the ICN-SDN forking is efficiently applicable.

In what follows, we utilize the following variables to build the model.

RC_x with $x \in \{SDN, ICN\}$
: Cost for request forking in case of an ICN-SDN or ICN implementation.

mc_x with $x \in \{PI, PO, FM, R, D\}$
: Cost per SDN control plane message; Packet_In (PI), Packet_Out (PO) and Flow_-Mod (FM), as well as the data plane packets containing content data (D) and content requests (R).

l_x with $x \in \{F, E\}$
: Length of path from first hop to the actual fork node (F) and from fork node to the egress nodes (E).

d
: The branching degree that describes how many packets are send out of the forking node for each incoming content request.

We start creating our model by only taking the amount of packets into account that are exchanged on the control as well as on the data plane.

Each incoming ICN packet to the SDN triggers a Packet_In (mc_{PI}) at the controller. After having processed the packet the controller replies with a Packet_Out message (mc_{PO}) containing the instructions for the switch on how to handle the packet. Further the controller sends Flow_Mod messages to the fork node (mc_{FM}) and all egress nodes ($mc_{FM} * d$) for the request forwarding. Additionally the ingress node is provided with a flow rule for the response rewriting and forwarding (mc_{FM}). The egress nodes are not provided with response forwarding flow rules (BO case) such that an incoming response triggers another Packet_In (mc_{PI}). According to its policies, the controller then instructs the egress node to forward the packet via a Packet_Out (mc_{PO}), whereas the other egress nodes are provided with drop rules ($mc_{FM} * (d-1)$). Eventually the ICN request and response costs are also added up in the model. The content request is just forwarded once up to the request forking node, since all other responses are discarded at the edge ($mc_D * (l_E + l_F)$).

The model of the pure ICN case consists of only ICN request and response packets. The request is delivered to the fork node ($mc_R * l_F$). The fork node duplicates the request and sends it into the different directions ($mc_R * (l_E * d)$). In contrast to the ICN-SDN, the pure ICN fork node receives the content replies of all its forks, hence, the message cost for the data dissemination towards the fork node consists of ($d * l_E * mc_D$). Subsequently the fork node delivers only one response towards the requester ($l_F * mc_D$).

The resulting formulas for the calculation of the request forking cost look like the following.

$$RC_{SDN} = 2 * (mc_{PI} + mc_{PO}) + mc_{FM} * (2 * d) + mc_R * (l_F + l_E * d) + mc_D * (l_F + l_E) \tag{4.9}$$

$$RC_{ICN} = mc_R * (l_F + l_E * d) + mc_D * (l_F + l_E * d) \tag{4.10}$$

In our opinion, the length of the fork branches (l_E) as well as the forking degree (d) are the main parameters that influence the efficiency of the request forking approach. Before scrutinizing their influence, we first determined reasonable values for all remaining parameters.

All message cost parameters are normalized to the size of a response packet (MC_D), which we assume to be in line with a MTU size of 1500 Byte. All other packet costs are given as the fraction of the 1500 Byte data packet. They are chosen via actual packet analysis. Further for the length of the path from the ingress to the forking node, we chose

a constant value of 4, which we consider a reasonable value.

$$MC_D = 1.0 \qquad MC_R = 0.11$$
$$MC_{PI} = 0.16 \qquad MC_{PO} = 0.11$$
$$MC_{FM} = 0.1 \qquad l_F = 4 \text{ [Hops]}$$

In Figure 4.8 both formulas (4.9 and 4.10) are drawn with the above mentioned parameter values, in dependence of the fork branch length (l_E) and the forking degree (d). It is visible that the costs of the ICN implementation quickly outweigh the costs of our ICN-SDN approach. A closer analysis is performed later on, first we take a look at how much each of the parameters influences the trend of the break even line of the functions.

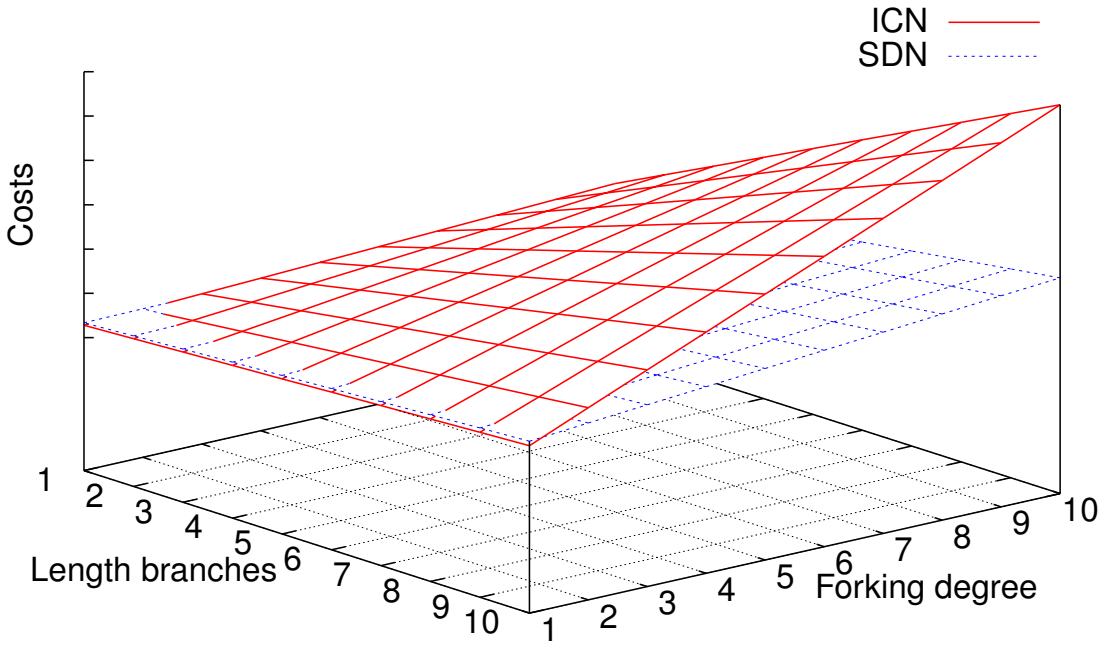


Figure 4.8: Message cost functions of the ICN-SDN and the pure ICN case, in dependence from the branch length and the forking degree.

Figure 4.9 with its subfigures displays the influence the different parameters have on the break even line of the cost functions. The previously determined default values are taken and in each of the graphs one of the variables is varied to illustrate its impact on the whole model. The cost values are varied between 0.05 and 0.4, in steps of 0.05

(Figure 4.9a - 4.9d). The hop count up to the fork point l_F on the other hand is evaluated between 2 and 10 (Figure 4.9e).

Analysing the graphs, one can see that the variables mc_{PO} and mc_{PI} (Figure 4.9a and 4.9b) influence the break even point in the same way, which is reasonable since they have the same multiplier. The Flow_Mod cost (Figure 4.9c) has the highest influence on the profile of the graph, at least when considering short branch lengths. The graph depicting the hop length to the fork node (Figure 4.9e) shows that at a path length of up to eight hops the ICN-SDN forking approach is still beneficial to the content dissemination. A look at the influence of the ICN request costs (Figure 4.9d) reveals that the relative costs of pure ICN compared to the ICN-SDN approach does not change, instead only the overall cost (hidden z-axis) is influenced.

Figure 4.10 shows the relative cost difference of both functions of Figure 4.8, whereas the pure ICN cost is taken as the reference value. We expect our request forking mechanism to be beneficial for every branch length and degree combination above the 0% level. Of course, these results rely on the assumptions we draw according to the environment that the approach is operated in. Further, the formula only applies to values above a forking degree of two, less than two forks are no forking hence the costs have to be calculated different. Hence, starting at two branches with branch length of one hop already saves 0.9% of the load created in the ICN forking case. With a fork length of two hops, the savings even amount to above 10%. This effect manifests itself when the branch lengths or forking degree is increased.

4.3.5.3 Request aggregation / response forking

In what follows, we also take a closer look at the network load in case of request aggregation respectively response forking.

Load per aggregated request The parameters our request aggregation model relies on are specified below.

RC_x with $x \in \{SDN, ICN\}$

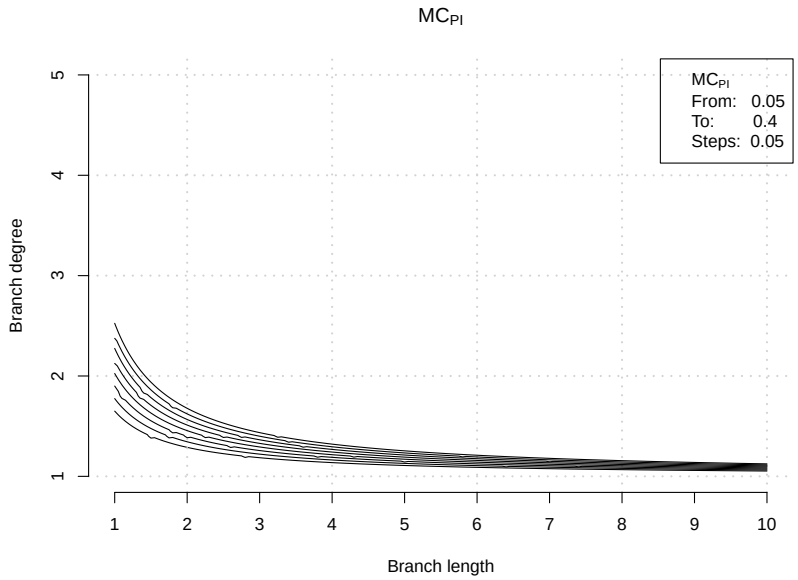
Cost for request aggregation in SDN or ICN case.

mc_x with $x \in \{PI, PO, FM, R, D\}$

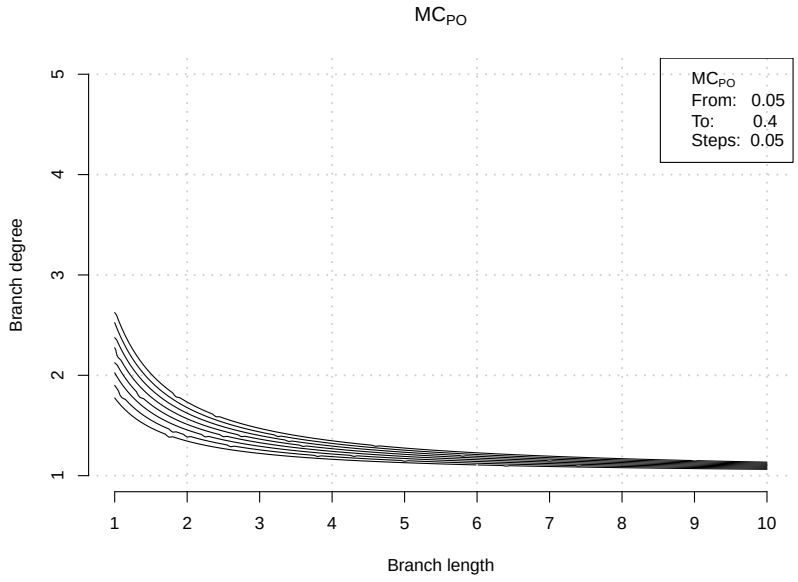
Cost per control plane message; Packet_In (PI), Packet_Out (PO) and Flow_Mod (FM) as well as the data plane packets containing content data (D) and content requests (R).

l_x with $x \in \{F, E\}$

Length of path from first SDN content hop to the actual fork node (F) and from fork node to the content egress nodes (E).

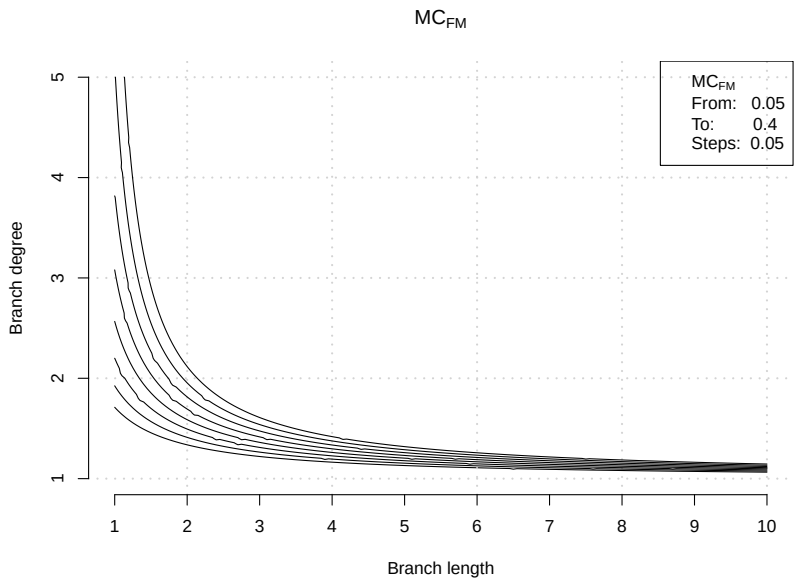


(a) Variation of Packet_In message cost

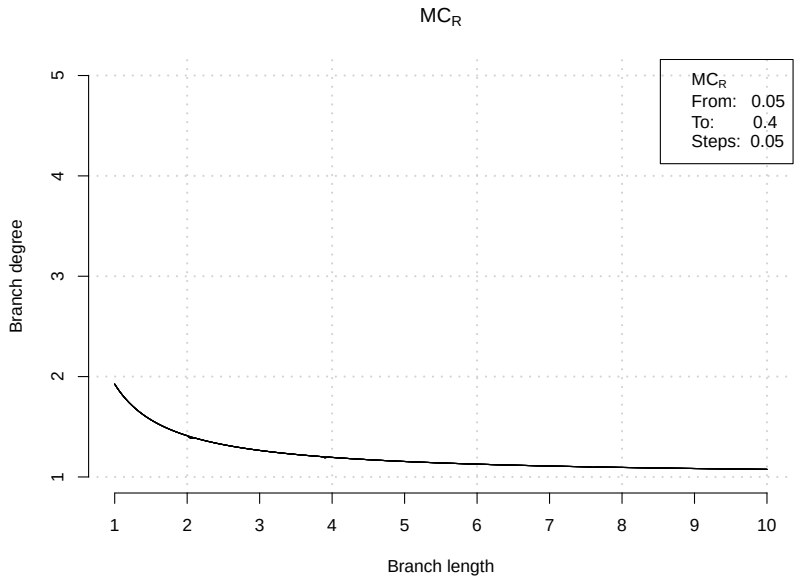


(b) Variation of Packet_Out message cost

Figure 4.9: Impact of certain parameters on the model

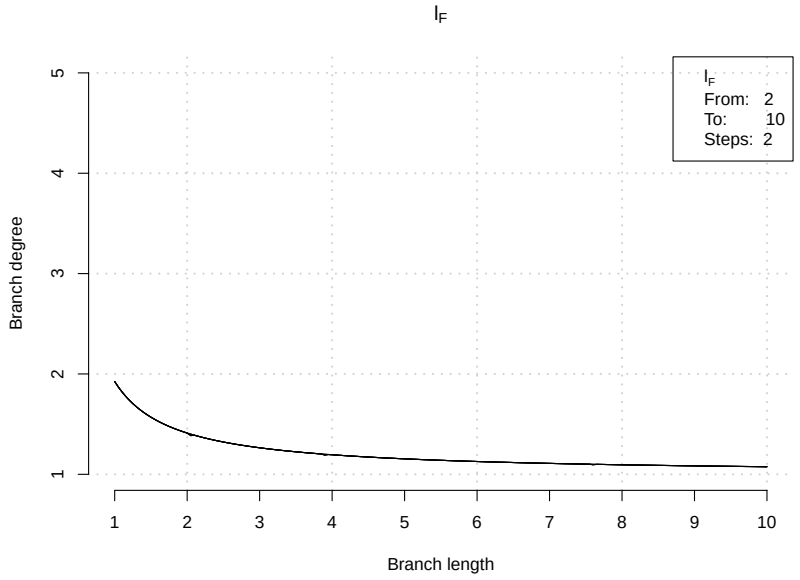


(c) Variation of Flow_Mod message cost



(d) Variation of request message costs

Figure 4.9: Impact of certain parameters on the model (cont.)



(e) Variation of hop length to fork node

$$MC_D = 1.0 \qquad MC_R = 0.11$$
$$MC_{PI} = 0.16 \qquad MC_{PO} = 0.11$$
$$MC_{FM} = 0.1 \qquad l_F = 4$$

(f) Default values used

Figure 4.9: Impact of certain parameters on the model (cont.)

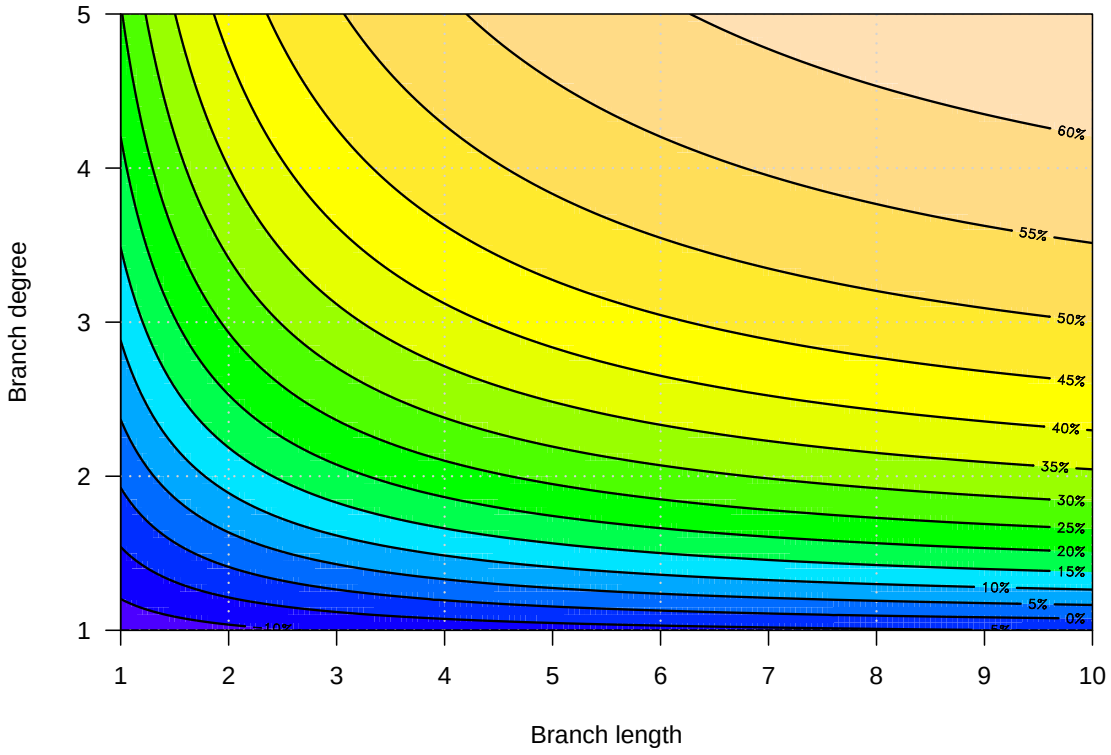


Figure 4.10: Relative message cost gain of the ICN-SDN approach in case of request forking, with ICN costs as the reference value.

d

The branching degree that describes how many packets are send out of the forking node for each incoming content request.

In case of request aggregation, the ICN-SDN costs consist of the following parts. Each ingress ICN-SDN node delivers the received content request via a Packet_In message to the controller ($d * mc_{PI}$), which responds via Packet_Out messages ($d * mc_{PO}$) that discards all but one request; probably the first. This content request is delivered all the way from the requester to the content source ($mc_R * (l_F + l_E)$). The edge switch facing the content source is instructed to rewrite and forward the request (mc_{FM}). The fork node is also instructed to duplicate, rewrite and forward the content packets (mc_{FM}). Finally the edge nodes towards each requester ($mc_{FM} * d$) has to be setup via Flow_-

Mod messages. All forwarding nodes in between do not need explicit rules, they perform the forwarding according to the default provisioned switch-id forwarding rules. Eventually the requested content is forwarded along the path from source to fork node ($l_F * mc_D$) and from fork node towards each requester ($(l_E * d) * mc_D$).

In the pure ICN case, the content requests of all requesting nodes are forwarded up to the point where their paths merge. From there on only one copy of the request is forwarded further towards the source ($(l_F + l_E * d) * mc_R$). The same path is then taken in reverse by the content packets on their way from content source to the request aggregation node and then further towards each content requester ($(l_F + l_E * d) * mc_D$). Hence, the following formulas result from our environment and the assumptions we applied.

$$RC_{SDN} = d * (mc_{PI} + mc_{PO}) + (d+2) * mc_{FM} + (l_F + l_E) * mc_R + (l_F + l_E * d) * mc_D \tag{4.11}$$

$$RC_{ICN} = (l_F + l_E * d) * (mc_R + mc_D) \tag{4.12}$$

Figure 4.11 depicts the relative cost difference of both functions for the parametrization described in Figure 4.9f. Hence, all parameters are equal to Figure 4.10. The pure ICN cost is again taken as the reference value against which our ICN-SDN approach is compared. The entire graph consists of only negative values, indicating that the response forking approach, within the plotted parameter space, creates a higher load in the network than the general CCNx response forking creates. In the constellation of two branches with a length of two hops, the disadvantage of the ICN response forking approach is the biggest, while with a rising number of branches and degree, the efficiency difference between both is reduced. The values below a degree of two are in this case irrelevant, since forking is only applicable with multiple branches. The maximum difference with this approach is at two branches with a length of one where it adds up to 11% more network load that is created by the ICN-SDN response forking.

4.3.6 Discussion

The advanced approach enables the utilisation of ICN packet forking throughout the ICN-SDN network. Thereby, the requirements of allowing for ICN request aggregation as well as ICN response and request forking are fulfilled.

In the analysis, performed in Section 4.3.5, we only considered the size of messages that traverse the data and the control plane. We have not taken into account any specific processing load that is introduced by various specific packets on the controller. Also no weighting of message types has been carried out. A weighting that regards control plane packets are more valuable than data plane messages, since control plane issues can easily influence the functioning of the entire network, while data plane issues probably only influences links or certain forwarding nodes.

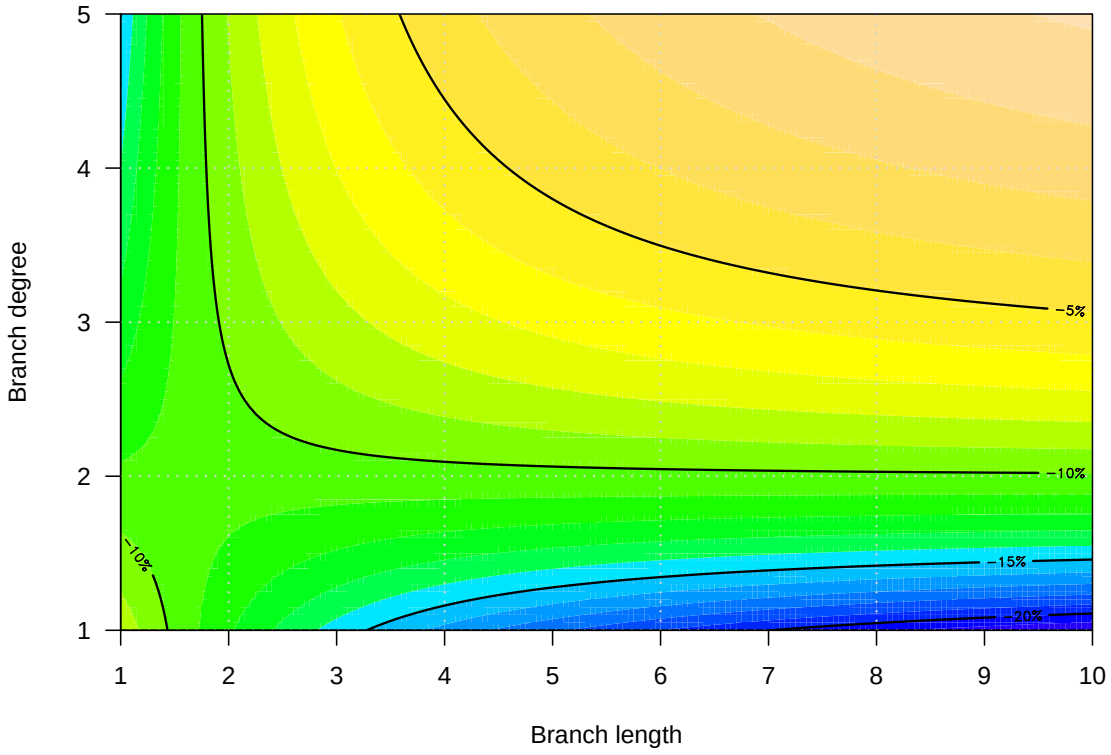


Figure 4.11: Relative message cost gain of the ICN-SDN approach in case of response forking, with ICN costs as the reference value.

The theoretical evaluation of the approaches indicates that the request forking approach is able to outperform pure CCNx forking through the ability of preventing unnecessary responses from being forwarded through the network. The picture is different when taking a look at the response forking. Within the plotted parameter space the ICN forking performs more efficient compared to our advanced approach. However, it is still quite good compared to the overlay case in which none of the network nodes in between is ICN aware and the content would have to be transferred end-to-end for each node.

Finally, via our advanced approach, we are able to additionally provide the required ICN packet forking capabilities that where missing in the basic approach. In addition to that the regular transit processing is improved and does no longer require the explicit interaction with the controller to map the response to a certain requester. However, one

crucial point is still missing, which is the concept for controller redundancy, to prevent network outages due to single controller issues.

4.4 Additional advances

In the following, we present some minor additions that might further improve the ICN-SDN approach.

4.4.1 Combined request and response forking

We are not limited to operate request or response forking on its own. Both mechanisms can also be combined as illustrated in Figure 4.12. In this scenario the content cache as well as the content source are queried for content. If the cache is able to provide the requested data, it is delivered to the requesting node (Figure 4.12b). If on the other hand only the source can provide the content, the content is forked within the SDN and delivered to the caches and the requesting nodes (Figure 4.12c).

4.4.2 Enable TCP

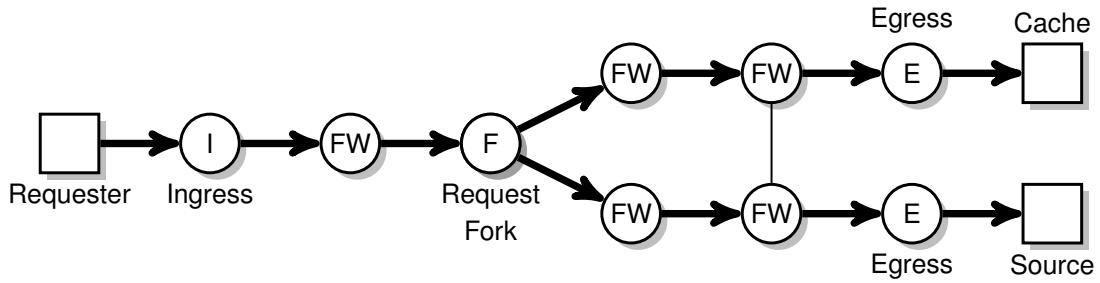
All mechanisms discussed so far base on the utilization of UDP as the transport protocol for the ICN packets. When using TCP as the underlying transport protocol, four major challenges arise:

1. The TCP connection is established – via TCPs three-way handshake mechanism – before any higher layer data is transferred. Hence, the name of the data to be requested is only transmitted after the TCP connection – with a particular node – is already established. Within this connection establishment phase the controller is not able to decide which content source to use, and thus where to send the initial TCP-SYN packet to.

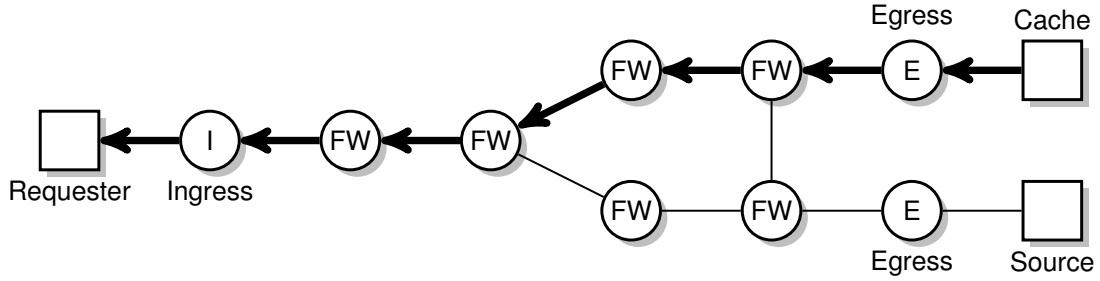
2. In case of a content miss, the source can not seamlessly be changed. A new connection setup to an actual source of the content is required.

3. Copying and forking of content responses to multiple requesters is not supported by TCP. Explicit handshakes have to be performed, sequence numbers and the like have to match.

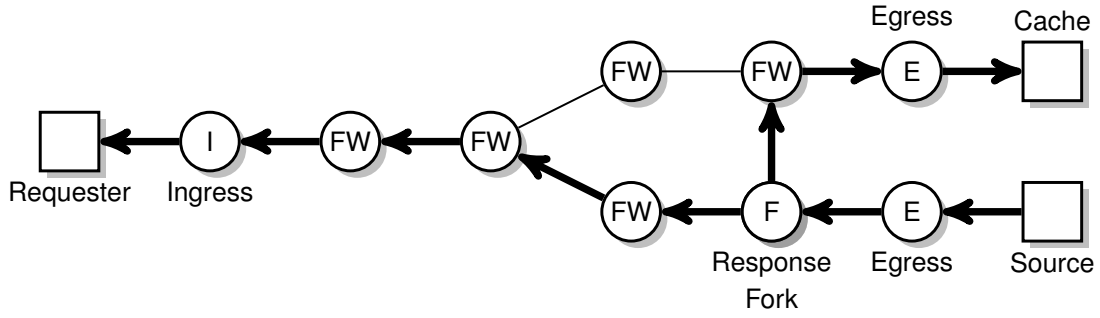
To enable the use of TCP regardless of the above mentioned challenges, we envision two possible solutions.



(a) Content request is forked



(b) Cache responds



(c) Source responds and cache is populated

Figure 4.12: ICN-SDN request and response forking

1. **TCP + UDP signaling packet** – Right before initiating the TCP connection by sending the TCP-SYN packet, the content requesting host issues a UDP packet that contains the content name which will be requested in the following TCP session. In addition, the source port number is required for the controller to distinguish different concurrent ongoing content transfers. The UDP signaling packet is processed by the controller and subsequently dropped. As soon as the TCP-SYN packet arrives at the ICN-SDN, the controller already knows which content will be requested within the TCP session and is thus already able to forward the TCP-SYN packet towards a controller selected appropriate content source.

2. **TCP proxying** – The controller redirects the TCP packets to a dedicated TCP proxy node that is used to terminate the TCP connection. Subsequently the proxy node is able to use either TCP or UDP to acquire the content in place of the requesting node.

While the first approach only solves the first challenge, the second approach allows to cope with all of the three challenging constellations but of course at the price of increased complexity and processing requirements. The second challenge could also be solved by including a TCP proxy in the content serving nodes such that in case of a content miss the node is able to act as a TCP proxy and requests the content for the requesting node, relaying and at the same time caching it for subsequent requests. By using UDP for this content acquisition the forking approach can again be used.

4.4.3 State management trade-offs

The proposed ICN-SDN approach allows for the trade-off of state management between entities. Hence, it is possible to shift the state management complexity between controller and forwarding elements to varying extents.
To reduce the state in SDN network forwarding elements, the following measures are feasible:

- The amount of flow table entries can be reduced by aggregating particular entries. For example, the content forwarding rules can be aggregated by only taking the switch-ids into account, different content requests – entering the ICN-SDN at the same switch – carry the same switch-id and can thus simply be forwarded according to the request ingress switch. All SDN forwarding elements only need to know their egress interface towards the specific switch-ids. Consequently just the content egress switch needs specific rules on how to rewrite and forward the packets for each requesting node in particular.

- The most extreme way to reduce forwarding element states is to provision no rules at all. Hence, each forwarding element will have to query the controller one after another. The controller has to process each packet multiple times, once for each switch the packet traverses.

Furthermore, measures to reduce the controller maintained states are also applicable.

- As soon as the request packet enters the ICN-SDN and the controller determines the MsgIDs, all parameters for the response path are already known. Hence, the response forwarding path can be provisioned along with the request path. This procedure avoids the necessity for the controller to also process the response packets.
- The response path egress packet re-writing rules can also be pre-configured. By doing so no MsgID to request origin information needs to be maintained by the controller.
- In combination with SDN flow rule inactivity timers, the rule provisioning can be performed such that the controller does not have to maintain any state and soft state timers on the rules that are provisioned in the network.

To reduce the overall state on the controller and SDN switches, the forking approach can be fine-tuned to specific content types or namespaces, like mentioned in Section 4.3.1.1 “Forking cases”. Hence, as soon as the necessity for specific content identification and treatment does not exist any longer, the assignment of one MsgID per content request can be changed to a per requester MsgID, thereby reducing the required flow rules significantly.

Chapter 5

Implementation

In the previous chapter we introduced our basic as well as the advanced approach of operating an ICN over SDN. We implemented our basic approach and performed a rudimentary evaluation, which is attached in Appendix A. This implementation then evolved to the implementation allowing for advanced ICN packet forking. In the further course we are going to apply specifically the advanced approaches to the ICN scheme of NDN.

In Section 5.1 we provide an overview of the CCNx specifics, which are of importance for our approach. Subsequently we elaborate on how the CCNx nodes cooperate with the controller in Section 5.2. This is followed by the actual controller architecture in Section 5.3. Thereafter Section 5.4 provides a description of the mode of operation of our Trema [52] based *CCNx-SDN-Controller*. The process of determining and setting up the required flow rules is described in Section 5.5. Eventually this chapter is closed with a description of the name-prefix announcement mechanism, we build to allow for some flexibility in content locations and ease of measurement conduction in Section 5.6.

5.1 CCNx host specifics

In the basic approach it is assumed that each content request is issued from a different transport protocol port. MsgIDs are created for the combination of the requester IP address and the port. They uniquely identify a content request at a time. NDN on the other hand follows a different approach and therefore the mode of operation of the CCNx-SDN needs to be adjusted.

The forwarding daemon of the CCNx implementation uses a single port for its communication with external CCNx nodes. The routing daemon *ccnd* does not bother about the source of a packet in terms of the sending host (IP address and transport layer port). Packets are distinguished only by their type and name fields that are carried as the first pieces of information within the ICN protocol layer of each packet. New connections are

only initiated once a new host is contacted. The connections are reused for all content exchanges between each pair of CCNx nodes. However, even connections to new hosts still use the same UDP source port number, meaning that connections to multiple hosts are multiplexed to one and the same local port. Consequently all Interests of a single CCNx node arrive at the SDN edge with the same source port, which contradicts the assumptions made in the approach in Chapter 4 that related requests carry the same transport layer protocol port. Since related request are thus not specifically identifiable as such on the layers below ICN, each Interest has to be processed and mapped by the controller. Consequently the controller load is increased, but the ICN-SDN approach is still applicable.

Further, no special adaption is needed on the consumer side to allow CCNx to communicate with our CCNx-SDN. It is only necessary to add name routing entries to the CCNx nodes FIB that point to the SDN-IP. This might be done in a default route fashion, but can also be performed for specific namespaces only.

5.2 CCNx-SDN network integration

The *CCNx-SDN controller* is assigned an IP address, already referred to as the SDN-IP. This IP address is used by the CCNx daemons of the consumer nodes to pointer parts of the ICN namespace to. To successfully accomplish data transmissions between CCNx nodes and the SDN-IP, the *CCNx-SDN controller* application in our actual configuration also needs to be assigned a unique MAC address. The controller has to respond to Address Resolution Protocol (ARP) requests for the SDN-IP with its unique MAC address via an ARP response. Only afterwards will packets, destined to the SDN-IP, be forwarded towards the edge switch that forwarded the ARP response. This mechanism also works if the requester node is not directly attached to the OpenFlow switch, since ARP request for MAC addresses unknown to switches are flooded throughout a bridged network domain. The ARP response will update the MAC address tables of the intermediary switches as well as the ARP table of the requester node. This mechanism assures that subsequent packets are in turn forwarded right towards that ICN-SDN edge switch.

For the forwarding of ICN packets towards the controller, two different methods exist. One valid option is to install flow entries on each switch, prompting it to deliver all packets directed to the SDN-IP and the specific ICN port explicitly towards the controller. Contrary to this mechanism we opted for the implicit alternative, which is not to install any matching flow entry. By doing so, the default action defined in the OpenFlows specification is executed, and thus the packet is delivered towards the controller in this manner.

Further, for the proper operation of the ICN-SDN, the UDP port space utilized for MsgIDs has to be restricted. Two distinct UDP ports have to be reserved and thus excluded

from the MsgID generation. One is the default CCNx UDP port such that CCNx nodes are able to deliver their Interest packets towards the CCNx-SDN controller. Besides that, another port is required to be exclusively used for the exchange of name-prefix announcements.

5.3 CCNx-SDN controller architecture

The controller implementation is divided into different components as depicted in Figure 5.1. For each controlled switch, the Trema *Switch Manager* (not depicted) forks a *Switch Daemon*. Each of these *Switch Daemons* is subsequently responsible for the communication with its associated *OpenFlow Switch*. All packets delivered to the OpenFlow controller arrive at this specific *Switch Daemon*. The *Switch Daemon* forwards the packets to the *Packet_In Filter*, whose task is to filter out Link Layer Discovery Protocol (LLDP) packets and hand them over to the *Topology* component, while delivering all other packets to the *CCNx-SDN controller*. The *Topology* component is likewise generating and receiving these LLDP packets for the purpose of determining the network topology.

The *CCNx-SDN controller* processes the incoming packets and in the further course sends information back to the *OpenFlow Switches*. These Packet_Out messages contain the processing information. They are handed over to the *Switch Daemon* responsible for the particular switch. The *CCNx-SDN controller* further utilizes the *Topology* component to calculate paths through the network. When queried with ingress and egress Datapath_IDs the *Topology* component provides a list of Datapath_IDs and their associated ingress and egress ports, which form the requested path.

To perform its requested tasks the *CCNx-SDN controller* component relies on four different ICN related data structures. An overview of the data structures is given in the following, as well as being displayed in Figure 5.2.

Switch dependent MsgID store The switch dependent MsgID data structure is used for the maintenance of MsgIDs, which are unique per switch and thus used in conjunction with a switch-id. The controller maintains one of these data structures independently for each of its associated switches. These switch dependent MsgIDs are used whenever a switch has to perform additional actions other than just forwarding packets, for instance packets which need to be forked, or the switch acts as the ICN-SDN edge node. In these cases packets have to be rewritten to carry either ICN-SDN internal or external forwarding identifiers. The controller provisions these actions via flow rules. These switch dependent MsgIDs are used for the switch to identify which of the multiple provisioned actions to apply to each

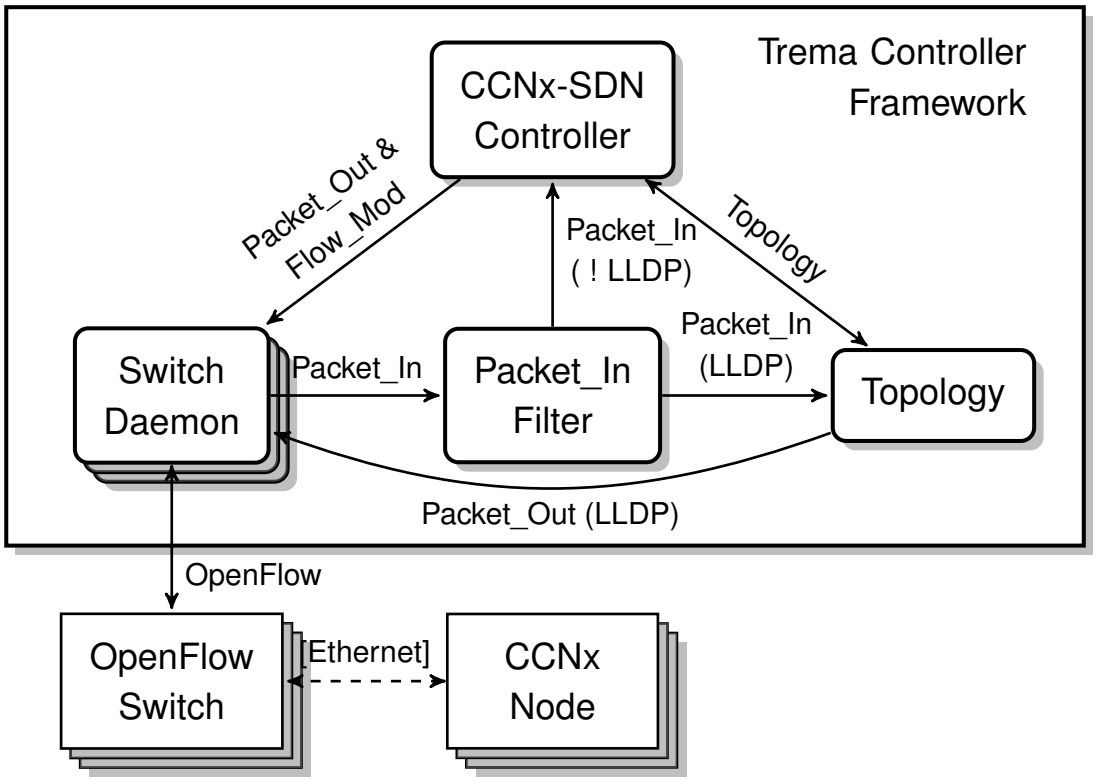


Figure 5.1: ICN-SDN implementation architecture

particular packet. The switch reads them out of the packet and looks up the action in the OpenFlow flow rule table.

The controller keeps track of MsgIDs that are actively in use in order to prevent erroneous multiple MsgID assignments. The actual actions that need to be performed reside only within the switches flow rule table.

Global MsgID store Whenever requests leave the ICN-SDN the external addressing scheme has to be applied to the packets. To associate each arriving content packet with its initial request, the source IP (SDN-IP) and port (Global MsgID) along with the destination IP of a request leaving the ICN-SDN is meant to uniquely map to specific content. This mapping is used to prevent the necessity to make the controller examining each incoming content packet. The IP and port information can be examined by the SDN switches themselves, who are then able to apply the previously provisioned rewriting rules. These global MsgIDs also need to be used only once at a certain point in time. Therefore the controller keeps track of the already assigned values. After the content transfer is completed this correlation

is invalidated and thus the three-tuple can be used to identify subsequent content request-response mappings.

Datapath store The *datapath store* is used to keep track of the OpenFlow Datapath to ICN-SDN MsgID association. Whenever a new switch connects to the controller, the controller assigns a switch-id. Hence, the *datapath store* is used to keep track of already assigned switch-ids. By utilizing the *Topology Manager*, the shortest paths between the nodes are calculated and rules for the associated switch-ids are installed such that each switch knows where to forward packets in order to reach any of the active switch-ids.

Forwarding Information Base (FIB) The FIB structure and maintenance functions of the *CCNx-SDN controller* are directly extracted from the original CCNx implementation. We just aligned the data structure that is returned as the result of a FIB lookup to meet our requirements. Hence, it consists of the MAC and the IP address as well as the UDP port of the content serving node along with the Datapath_ID and physical port number on which this nodes packet arrives at the ICN-SDN edge.

5.4 CCNx-SDN controller mode of operation

As soon as a switch establishes its connection with the controller, the switch is assigned a switch-id. Since the controller knows the exact topology of the network, it provisions all forwarding rules for the new switch. Via this measure new switches get to know how to forward packets directed to switch-ids other than their own. Subsequently also already existing switches will be provided with the necessary information about the new switch and how to forward packets containing its switch-id.

Figure 5.3 illustrates the packet processing work flow performed by the *CCNx-SDN controller* component. Whenever a Packet_In message is handed over to the *CCNx-SDN controller*, it checks if the actual packet is an ARP request referring to the SDN-IP. If the packet matches this criteria, the controller creates a corresponding Packet_Out message containing the ARP response which is subsequently handed over to the switch that yielded the Packet_In message. If the packet is no ARP request for the SDN-IP, the destination IP address is checked for the SDN-IP value.

When the destination IP differs from the SDN-IP, basic bridging mechanism are performed as defined by the *routing_switch* Trema application [53], which we took as a starting point for our *CCNx-SDN controller*. If the packet is directed to the SDN-IP, the destination port is evaluated. In case the port corresponds with the name-prefix an-

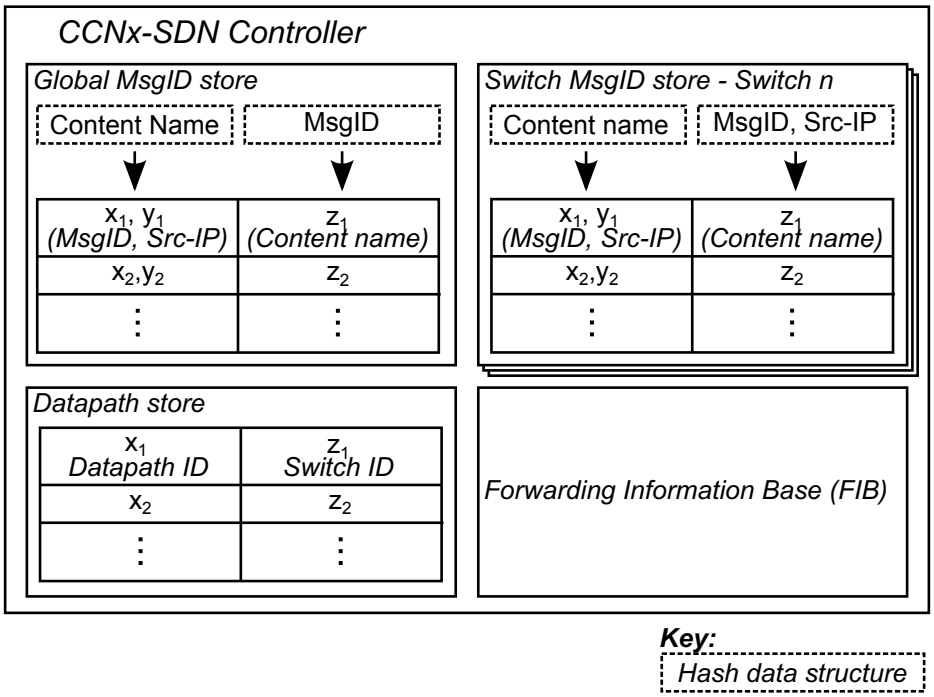


Figure 5.2: CCNx-SDN controller data structures

nouncement port, the enclosed prefix is either added to or removed from the controllers FIB.

Whenever the destination port equals the default CCNx port, the source IP and port information is used to acquire a host identifying, switch dependent MsgID. If still valid, an already assigned MsgID or otherwise a newly created one is returned. If the MsgID is newly created, a new rule, which will handle the content packets that the actual request will trigger, is installed. It matches the particular ingress node switch-id that the rule is installed on and the newly created MsgID. The matching packets are altered in a way such that they carry the ICN-SDN specific IP and port information in the source fields and the requesting nodes information in the destination fields.

Following, the CCNx mechanisms for FIB lookups is triggered to determine potential content origins. This information is subsequently used to build a packet dissemination tree for the forwarding and forking of packets.

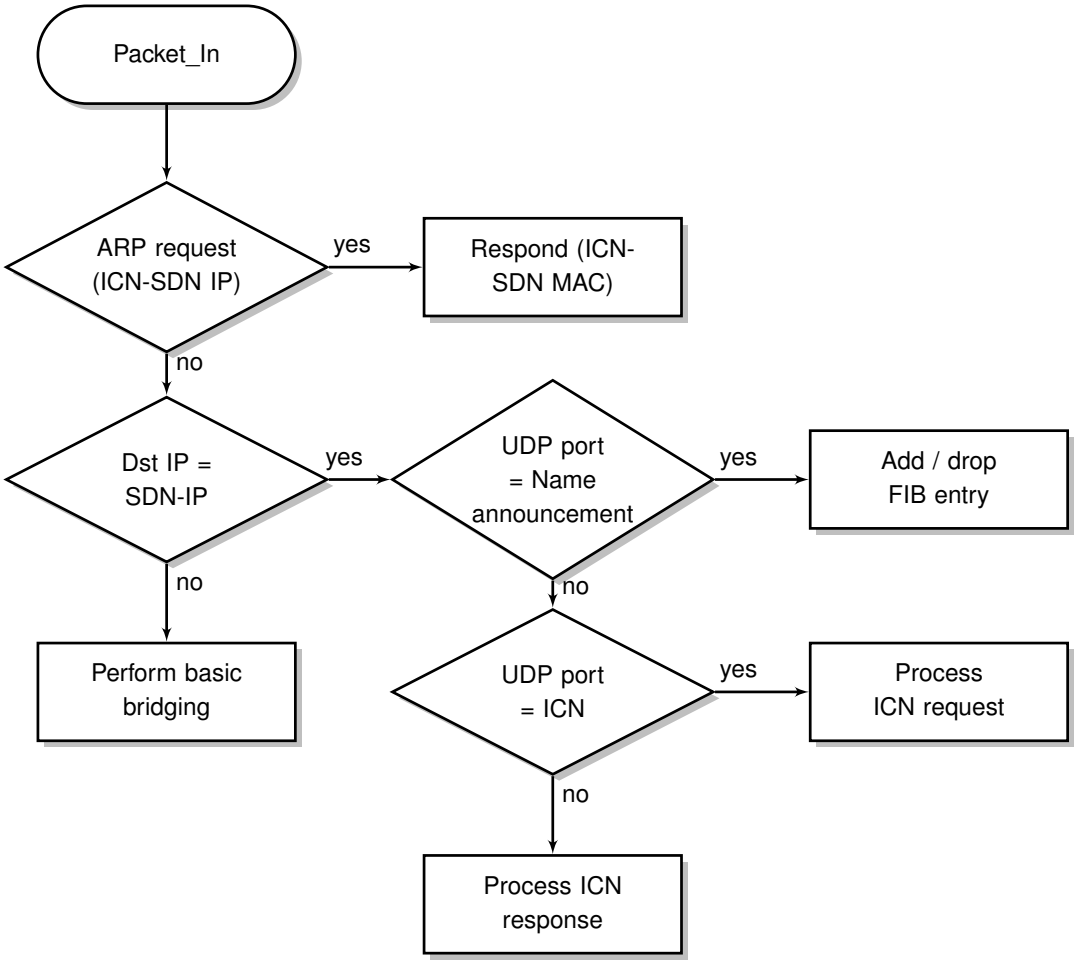


Figure 5.3: Controller packet processing work flow

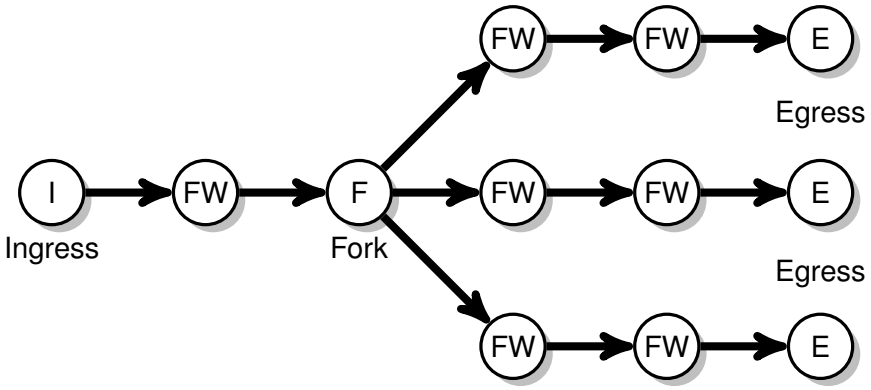
5.5 Flow rule setup

To form the dissemination tree and thus figure which roles the involved nodes have to perform, the following processing is applied.

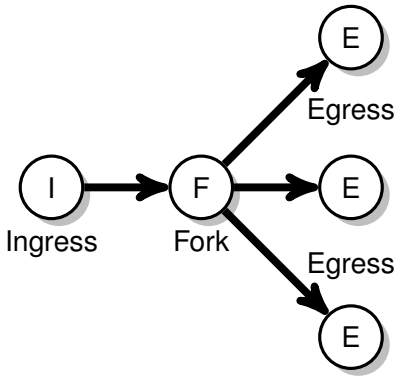
A single path from the requesting towards each content serving node is calculated via the support of the *Topology Manager*. Each returned path consists of switch-ids and their outgoing port numbers. The returned paths are subsequently added to a tree structure rooted at the ICN-SDN ingress node. An example of such dissemination tree is depicted in Figure 5.4a. Each node of the tree holds the information about all its successor nodes as well as the information if it acts as the egress for an adjacent content serving node.

The resulting tree is subsequently collapsed, only nodes that have to perform roles which require special actions (Ingress, Egress, Fork) remain in the tree. The ingress

node is simply identified as the root of the tree. Fork and Egress nodes on the other hand are identified via their amount of successor nodes. If these are greater than one, the node acts as a forking node. If the egress flag is explicitly set or the amount of successor nodes is zero, the node acts as an egress node. All intermediate nodes do not have to perform any special action other than forwarding the packet according to their general forwarding entries. Only the nodes that remain after the collapse are nodes that need additional rules. For the previously introduced dissemination tree in Figure 5.4a, the collapsed tree is illustrated in Figure 5.4b.



(a) Original dissemination tree



(b) Collapsed dissemination tree

Figure 5.4: Dissemination tree creation

The collapsed tree is subsequently traversed in post-order. For each of those switches in the dissemination tree, a unique MsgID is created. All actions that need to be performed with the packet that triggered the whole process are referenced via the switch-ids and the associated MsgIDs. The MsgID of each switch is handed over to the parent node within the tree such that previous nodes on the dissemination path can in-

stall rules to rewrite the MsgID of the packet to the values expected by the subsequent nodes (towards content origin). As soon as all these child node dependent MsgIDs are determined and provisioned, the rule provisioning for the actual node is straight forward. All particular rewrite and output actions are concatenated and subsequently deployed via a Flow_Mod message to the switch. In this way the request packet rewriting and forking rules are deployed from the egress towards the ingress node. This order of flow installation also prevents the packet from arriving at a switch that is maybe not yet provisioned, which would cause additional load on the controller, since these packets would trigger additional Packet_In events.

This method only describes the setup of the request forwarding flow rules. Nevertheless, the flow rule provisioning process for the content responses does work accordingly. Instead of being rooted at the ingress, the tree would be rooted at the egress node, which serves as the first hop into the ICN-SDN for content packets.

We are talking about the process of building a dissemination tree, which conveys the impression that this mechanism is only applicable in the forking case. However, the whole algorithm is generally applicable also if only a single content origin node exists. In that case the whole path collapses into two sequential nodes, ingress and egress, which can further be provisioned accordingly.

5.6 Learning and managing object locations

We ourselves defined a simple protocol that is used to quickly (de-)register name prefixes with the controller. By this mechanism content providers are able to notify the *CCNx-SDN controller* about their ability to provide content for specific namespaces. This information is then added to the *CCNx-SDN controller* FIB. We also opted for using the SDN-IP address for the name-prefix protocol instead of an additional controller IP address and the delivery of the packets via general bridging. Thereby, the advanced knowledge that the processing via the OpenFlow controller provides is used. The benefit is that whenever FIB entries have to be created, the Datapath_ID and Datapath port is already available in the Packet_In structure via which the edge switch delivers the packet to the controller. Just the prefix that is (de-)registered is appended as the payload of the name-prefix announcement packet. Further, using the SDN-IP also simplifies the configuration since only one IP address for name-prefix registration and FIB entries needs to be configured. Additionally there is no direct dependence on a specific controller. Whenever the controller of the SDN changes, due to failures, maintenance or the like, the controller that receives the name-prefix announcements changes as well.

For our prototype implementation we also refrained from including the listening port of the CCNx daemon on the prefix announcing node, since we stick to using the CCNx default port. However, it may be worth mentioning that we expect some pitfalls in case

of intermediate Network Address Translation (NAT) devices where some sort of port modification is performed. One solution for this issue would probably be to utilize the CCNx daemon itself to transport the name-prefix announcements. Doing so, the packet modifications would be applied in the same way that they are applied for all other CCNx packets. As a consequence, the port could still be read out of the Packet_In arriving at the controller. In this case, no explicit port needs to be reserved for the name-prefix registration procedure but a particular namespace is required.

Chapter 6

Evaluation

To analyse our ICN-SDN forking approach in conjunction with the scenario drawn out in Section 4.3.1, we setup and perform the measurements described in the following. This measures are performed to evaluate the analytically gained performance values described in Section 4.3 and check the applicability of the whole approach.

Following, we will first give an overview of the components of the measurement environment in Section 6.1. Subsequently, in Section 6.2 the measurement setup is detailed with the description of the network topologies used, the data that is gathered throughout the evaluation runs, how the environment is initialized and the procedure description of how the measurements are conducted. The corresponding results are then summarized and discussed in Section 6.4.

6.1 Measurement environment

We conduct the evaluation experimentations in a virtualized environment. The entire network topology with all its nodes – CCNx sources and sinks, SDN-controller and switches – are executed on a single evaluation computer. Therefore, the Mininet framework [54] in version 2.0.0 is utilized in combination with CCNx in version 0.7.2. Mininet utilizes lightweight network namespace isolation [55] such that all emulated nodes run atop the same kernel and thus share the same resources except for the separated network view. Each network namespace has its own configuration including (virtual or physical) network adapters, IP addresses, configured latencies etc. The network namespaces are then connected to one another to form the desired network topology. On the other hand, the file system and process namespaces are not separated, which requires the definition of unique unix-domain sockets per namespace instance. Otherwise the start up of additional *ccnd* instances is prevent. The *ccnd* also needs to be instructed on which IP addresses to listen for incoming connections, thereby the loopback IP address (127.0.0.1)

needs to be explicitly excluded. Cabral et al. meanwhile published a modified version of Mininet, called Mini-CCNx [56, 57], which is shipped with already CCNx enabled node templates. Mini-CCNx is available via [58].

By using these lightweight node emulation mechanisms, we are able to execute the evaluation environment on an Intel Core i5 quad core workstation at 3.2 GHz with 8 GB of RAM running Ubuntu 12.10 x64 with a Kernel version of 3.10.9. For the SDN switches we use the Open vSwitch [59] implementation in version 1.4.3. These switches are further managed by our Trema based *CCNx-SDN controller*.

6.2 Measurement setup

We performed only rudimentary evaluations of our basic approach and focus on the evaluation of the advanced approach. The results of the basic approach evaluation are attached in A. Hereinafter, we provide detailed information on the measurements performed with the advanced approach.

6.2.1 Data of interest

We compare our approach of Interest forking via the ICN-SDN controller to cases in which the pure CCNx is also performing Interest forking. Like previously introduced we are able to operate our controller in two different modes. The content transmission may be performed Bandwidth Optimized (BO) or Delay Optimized (DO). Both approaches differ in their resulting transmission times as well as their utilized network resources. Hence, we are going to evaluate both modes. Since we expect the data plane load to be reduced via the ICN-SDN BO approach that is able to stop superfluous replies, the imposed data plane load is collected once per each link. We will thereby be able to determine the effect of this mechanism. The ICN-SDN DO approach on the other hand is optimized for response times, introducing the least control delay possible. To measure this effect the per chunk response delays are collected. Thus, the time it takes from issuing a request till the content chunk arrives at the requesting node. Further, SDN utilizes the centralized controller, thereby introducing the control plane network. To evaluate the load our approach imposes on the control plane, the amount, type and size of the control plane packets is collected.

To gather the required data we are going to request multiple content chunks through the use of the CCNx included tool named *ccncatchunks2*. The general output of this tool already yields the per chunk transmission times required for our analysis. For the analysis of the amount of consumed network resources the network packet analyzer and capture program *Wireshark* [60] is used. On each link used in the content dissemination one node is selected to perform a packet capture, which is saved for later evaluation.

Additionally, in the ICN-SDN cases a packet capture is created for the loopback interface (lo), which is shared among all SDN switch namespaces for their communication with the controller. This further allows detailed control plane load analysis.

6.2.2 Evaluation topologies

We generated different network topologies, as depicted in Figure 6.1. For the creation of the topologies, we used the IGen topology generator [61]. Each of the topologies consists of 50 nodes of which the red nodes form the backbone of the network, while the gray nodes perform as edge nodes. The topologies 1 and 2 (Figure 6.1a, 6.1b) consist of edge nodes that are connected to two different core nodes, topologies 3 and 4 (Figure 6.1c, 6.1d) consist of three and topologies 5 and 6 (Figure 6.1e, 6.1f) consist of four core node connections per each edge node.

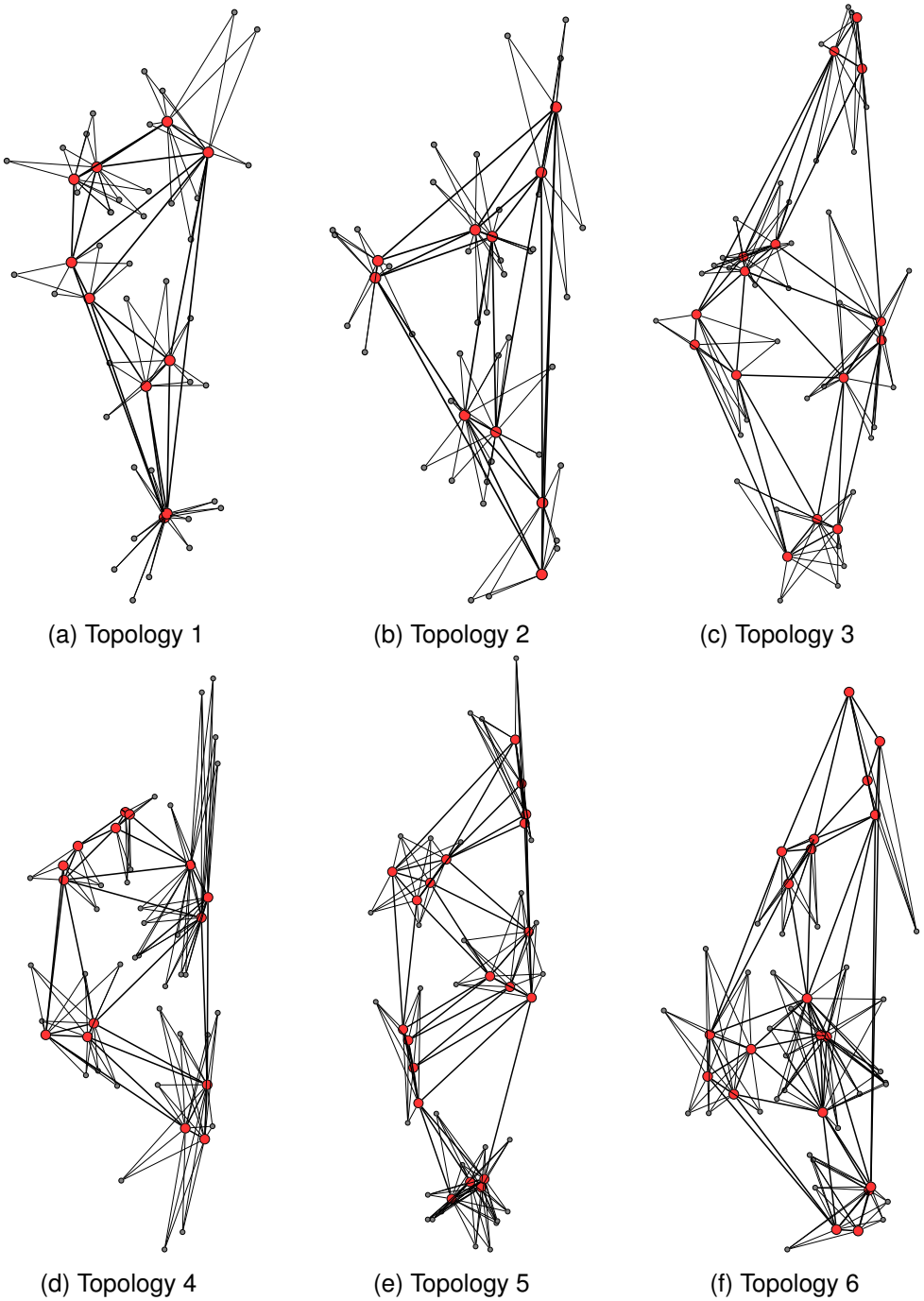
In all topologies the link delays are consistent. Core links are assigned 20 ms, while edge links are assigned 5 ms latency. This difference in link delays is introduced due to the greater distance we assume between core nodes compared to adjacent edge nodes. Further the control plane delay is set similarly for all SDN nodes to 5 ms. Thereby, the equal configuration of this value for all nodes is a result of the restrictions of the Mininet framework.

6.2.3 From generated topology to executable network

The topologies created via the IGen topology generator are processed and converted into Mininet scripts that consist of the topology information as well as additional instructions used to perform particular measurement runs.

The topology description is converted such that in the ICN case each node is mapped to a CCNx node while in the SDN case all nodes are generally converted into SDN nodes. To emulate the content origin being network proximity-wise far apart, the requester and origin nodes are placed on nodes that are maximum diameter hops away from each other.

For the two SDN cases the caches towards which the Interests will be forked are also determined in this pre-measurement phase. To account for the central view the controller has on the network and thus the knowledge it can base its decision making process on, we decided to place the caches (proximity-wise) in the same half of the network that also the requester resides in. The topology nodes, which are selected to perform the origin and cache functions in the SDN cases are each accompanied by an ICN node, which is directly connected to the SDN node. These ICN nodes run the CCNx software and performs the actual ICN processing and serve the content. An illustration is depicted in Figure 6.2. Figure 6.2a shows the resulting ICN and Figure 6.2b the SDN topology.

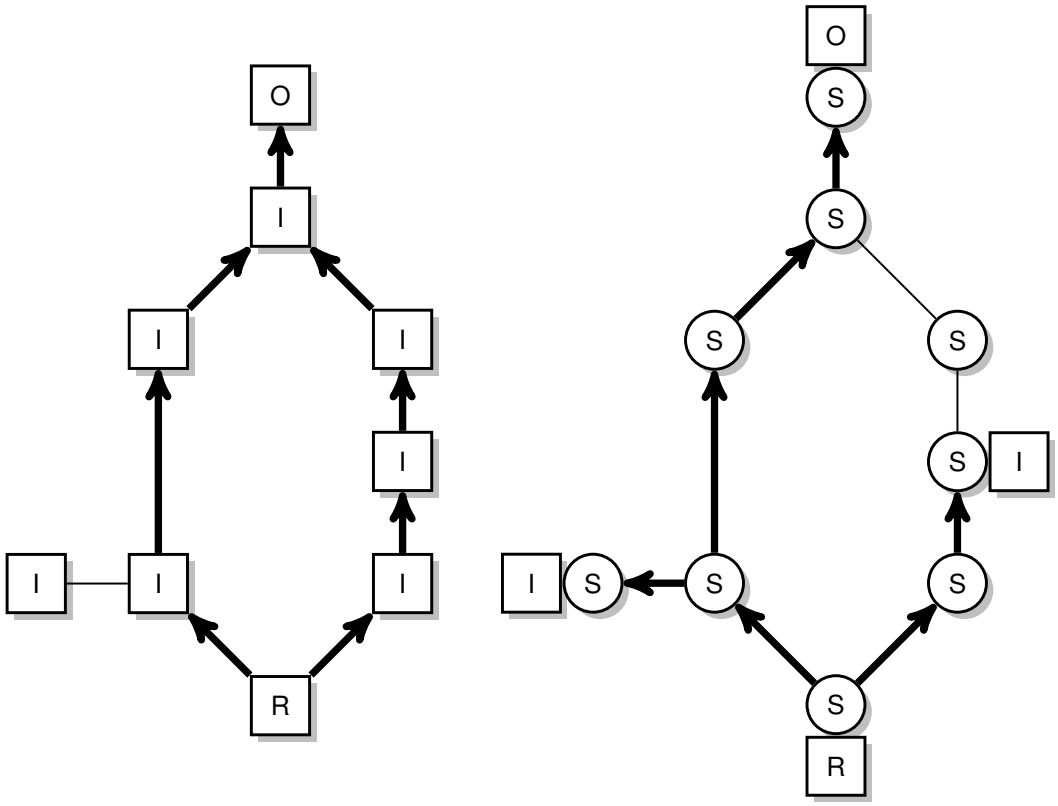


(a) Topology 1 (b) Topology 2 (c) Topology 3

(d) Topology 4 (e) Topology 5 (f) Topology 6

Figure 6.1: Topologies used for evaluation

They are all configured as dead-end caches, hence, they are not provided with additional



(a) ICN (b) SDN

Figure 6.2: Illustration of the same topology in SDN and ICN case with a fork factor of 2 and an exemplary cache selection (*I=ICN node, S=SDN forwarding node, R=Requesting ICN node, O=Origin ICN node*)

forwarding information they will not forward requests they are unable to reply to via their cache content. On the opposite the ICN case does not need this explicit cache selection, since the cache nodes in the ICN case are implicitly those nodes on the path towards the origin.

In the ICN case each of the CCNx nodes is equipped with multiple network interfaces that serve as point-to-point links, directly connecting nodes without intermediary switches or the like. Further, the nodes are configured with host routes which point to their direct adjacent neighbors. By doing so, we want to emulate a close to native deployment in which ICN packets are transported hop-by-hop. To ease the identification of nodes throughout the network and especially in the packet captures, each node is assigned only one IP address, which is bound to the loopback interface. Enabling the *IP_For-*

warding option in the Linux Kernel allows us to operate the interfaces between nodes in unnumbered mode. Hence, no IP address is explicitly assigned to each interface itself. Incoming packets are thus routed according to the routing table configuration, which delivers IP packets for the single node IP address towards the local loopack interface and thereby to the CCNx application.

Throughout the evaluation we explicitly leave aside the aspects of how the caches are populated with content since we focus on the effectiveness of the forking approach. However we do envision different possibilities, for instance by forking requester driven responses towards particular cache nodes or any other method. Nevertheless, for our measurements we need to populate the caches with content. Therefore, the CCNx nodes, which are subsequently utilized as caches, start loading content into their cache, themselves, via a local mechanism prior to each measurement run. This cache population is deterministic per one similarly configured ICN, SDN BO and SDN DO pass.

6.2.4 FIB population / routing

Furthermore, FIB entries have to be configured in the ICN case since we want to avoid the overhead of running an ICN routing protocol between the nodes. In this phase of the Mininet topology generation, further referenced as pre-measurement phase, a topology graph is build, which is used to calculate shortest paths between the direct adjacent nodes of the requester and the origin node. The FIB entries are then configured to create the branches on the requesting node towards its neighbors and from there on the shortest path towards the origin node. Of course this implies that our topologies themselves contain a certain node degree. According to the outcome of this calculation, the FIB entries of all on-path nodes from requester to origin are configured to be able to successfully forward incoming content requests for the measurement namespace. Further the calculated shortest paths are used to specifically start packet capture processes only on the nodes and their interfaces involved in the content transmission.

The caches selected by the controller are also determined in this pre-measurement phase and configured similarly in both SDN cases (BO and DO). The selected cache nodes are configured to register the namespace used for the measurement via the name-prefix registration mechanism previously described. Hence, at run time, in the measurement setup phase, the controller receives the information towards which ICN nodes to fork requests that belong to the particular namespace. The controller itself will then calculate the paths through the network via the use of the previously introduced *Topology Manager* component.

6.2.5 Parametrization

We execute comparative runs of ICN, SDN BO and DO with the same initial setup according to *(i)* origin and requester node placement within the topology and *(ii)* the pre-population of caches with a fraction of the chunks used for the measurement. The set of content used for the measurements thereby consists of 50 data chunks with a CCNx payload length of 1000 bytes. This data is accessible via the CCNx *Repository* application, which serves as a non-volatile chunk level cache.

The general content cache size per ICN node is larger than the amount of 50 content chunks. This is to assure that throughout the ICN measurements no cache eviction actions takes place. All content, which is not available when the content request arrives will be cached when the content arrives from an upstream node. Thereby, all pre-populated content chunks stay available in the cache throughout the measurement without being prematurely evicted.

The chunk size has to be limited, since the CCNx protocol layer is not considering the MTU of lower network layers. Content chunks that would result in packets bigger than the IP MTU are thus subject to IP fragmentation. IP fragmentation however is not supported in our environment. When IP performs fragmentation, the transport layer header is only included in the first IP fragment. Since our approach relies on matching transport layer header information to forwarding rules to perform the packet forwarding, this information has to be present in each and every packet. However, this is not the case when IP fragmentation is utilized.

The MTU of Ethernet amounts to 1500 bytes. This is further reduced by the IP and the UDP header. Additionally the CCNx header reduces the maximum payload size such that we decreased the chunk size to 1000 byte as stated above.

The parameter set we explicitly evaluate consists of *(i)* the amount of content pre-populated in the cache (Table 6.1a), *(ii)* the ratio of pre-populated content between ICN and SDN caches (Table 6.1b) and *(iii)* the fork factor (Table 6.1c). We analyze each of the parameter combination stated in Table 6.1 via the topologies depicted in Figure 6.1.

Pre-cached content per node The per node pre-cached content parameter determines the number of node-wise randomly chosen content chunks out of our evaluation content set, which are already available per cache node. Since our set of content chunks is quite limited and we want to avoid long cache initialization phases through random request generation and conduct the SDN BO and DO cases as comparable as possible, we chose to explicitly populate the caches with content prior to the actual measurement run. Furthermore, to evaluate different cache hit ratios the *per node pre-cached content* is introduced.

Ratio of pre-populated content ICN to SDN The SDN case always populates the

(a) Pre-cached content per node [# chunks]	(b) ICN to SDN cache ratio	(c) Fork factor
10	1/1	2
20	1/2	3
30	1/4	4

Table 6.1: Parameter space used for the evaluation

caches with "*per node pre-cached content*" number of chunks, while the amount of chunks per ICN cache is multiplied with this value.

We introduced this factor to account for the fact that in the SDN case we expect a cache population mechanism in place that will aggregate for instance certain namespaces deterministically on a group of nodes, thereby reducing the probability for the content to be overwritten. Since not every SDN node will have an ICN cache co-located with it, beyond that, the few caches deployed will likely be provided with higher caching capacity. Finally, content will not be stored multiple times throughout the network, in each traversed ICN node, occupying the cache space multiple times while evicting other objects from the cache.

Fork factor The fork factor defines the duplication coefficient. In the SDN case this parameter defines the number of caches that the requests are delivered to while in the ICN case not only the forking factor but also the path length defines the number of caches, which are queried when the request passes through.

Since in the ICN case the request issuing node performs the forking, the topologies have to allow for the defined amount of forks by providing enough connections towards a diverse set of core nodes. Therefore, they are created with different amounts of connections per edge node towards the core. Thus the topologies used to execute the measurements are dependent on the fork factor.

6.2.6 Procedure

The following steps are performed for each scenario (ICN, SDN BO and SDN DO) and every combination of the parameters described in Section 6.2.5.

1. The Mininet script is executed and the predefined topology is initialized.

   ICN: Each individual node is assigned one IP address, which is configured on its loopback interface. Furthermore, the IP host routes are added to the IP routing table. The network interfaces directly connect pairs of ICN nodes.

SDN: The OpenFlow switches as well as the ICN nodes are instantiated. The interfaces, which connect the ICN nodes to the OpenFlow switches are directly configured with an IP address.

2. The CCNx daemon is started on each ICN node.

3. A CCNx *Repository* daemon is started per each ICN node. This repositories contain all content chunks used in the evaluation measurement. The *Repository* automatically registers with the local CCNx daemon such that requests for the measurement namespace will be forwarded to the local *Repository*.

4. Each ICN node requests the content chunks individually determined for it in the pre-measurement phase. The local *Repository* receives the request and responds with the chunks. Thereby, the *ccnd* caches the chunks for future requests. The amount of chunks requested and thus cached is dependent on the *per node pre-cached content* parameter.

5. The *Repository* is stopped on all but the selected origin node.

6. The *ccnd* FIB is modified.

   ICN: The FIB of each but the origin and requester node is adjusted such that the measurement namespace follows the previously calculated shortest path. The origin node FIB does not need to be adjusted, since the *Repository* registers itself with the required namespace. The requester node is configured to perform the request forking multiple FIB entries.

   SDN: The FIB of the requester node is adjusted to forward requests for the measurement namespace to the SDN-IP.

7. The packet capturing processes are started.

8. The actual measurement takes place. Therefore, the total set of 50 content chunks is requested once from the selected requester node. The per chunk retrieval times are automatically recorded via the *ccncatchunks2* tool saved for later analysis.

9. Finally the packet captures are stopped, the capture files are saved and the entire Mininet environment is unloaded and shut down.

6.3 Measurement results

We ran multiple measurements that consisted of the three scenarios ICN forking, SDN BO and SDN DO with all combinations of the previously introduced parameters. Similar

parametrized runs are executed up to seven times in different topologies with different requester, origin and cache nodes. This procedure in total leads to about 180 measurements per each ICN, SDN BO and SDN DO run.

In the following we will take a close look at the measured data of transmission and processing times as well as the data plane and control plane load.

6.3.1 Transmission times

We deduced the average per chunk transmission times for individual measurement runs. These values are grouped according their respectively scrutinized parameters and used as the data basis for the following plots. These boxplots depict the median value as the thick black line. The solid box starts at 25% (first qurtile) and reaches up to 75% (third quartile) it thereby frames the 50% of the measured values that are called the inter-quartile. The dots that appear in some of the graphs are the statistical outliers. These outliers are measurement values that are more then 1.5 times the inter-quartile range away from the first or the third quartile. The whisker are thereby adjusted to stop at the last data point before the inter-quartile range is exceeded. Since boxplots contain this wide range of information in one plot, it is an adequate means to gain an overview of our measured data set.

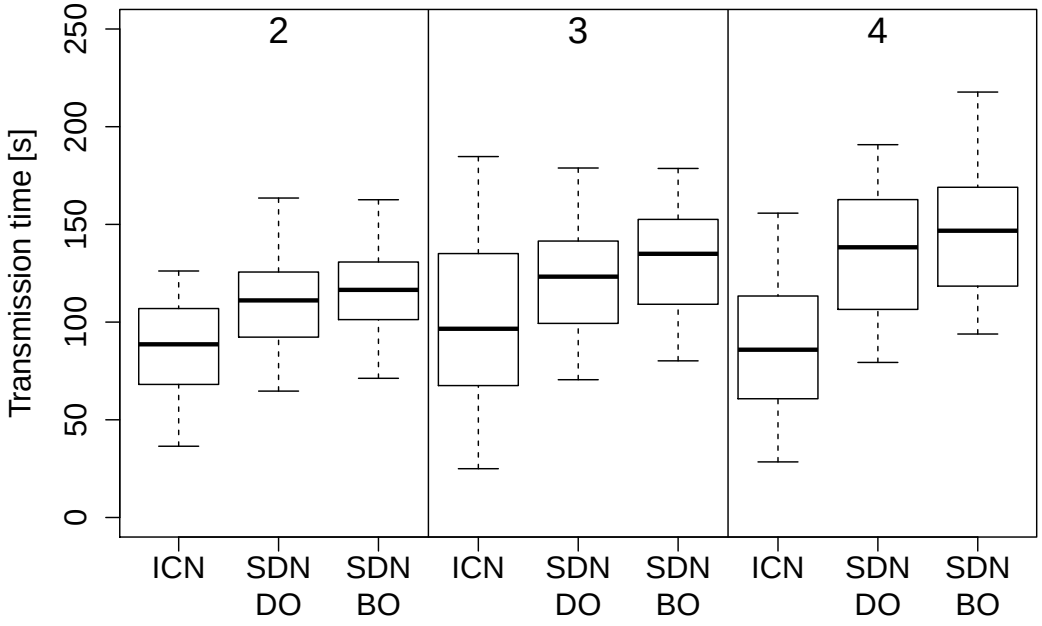


Figure 6.3: Effects of the forking factor on the avg. transmission times

Figure 6.3 depicts the influence of the forking factor on the transmission times. The

parameter space covers 2, 3 and 4 forks. The transmission times of the ICN case remain similar except the increase in the runs with 3 forks (center segment) where the transmission times are spread wider. We assume this spread is influenced by some noise in the measurements, since in the 4 fork runs the results are again more narrow.

Both the SDN cases DO and BO show the effect that by increasing the fork count the transmission times also increase. Furthermore, the dispersion increases. This behavior can be explained via the increased processing the ICN-SDN controller has to perform. For each fork the forwarding paths are calculated, which subsequently have to be merged in the transmission tree. More nodes have to be processed to create and subsequently collapse the tree. Finally, more egress, but probably also more forking nodes have to be provisioned. Hence, adding forks inevitably increases the controller processing and thereby also the transmission time in the SDN cases. Figure 6.4 displays the effects of

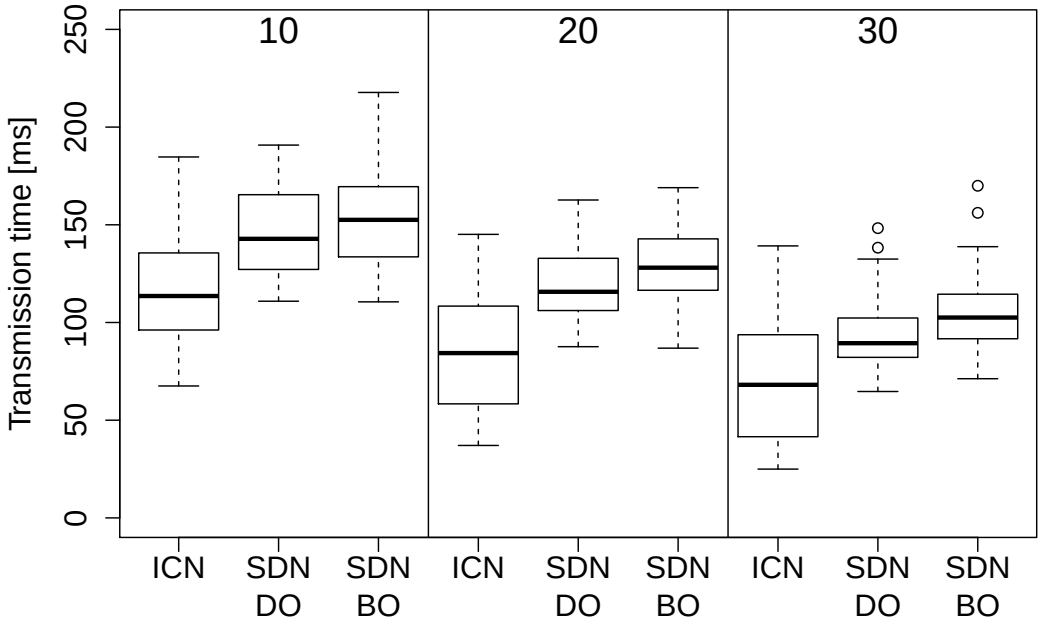


Figure 6.4: Effects of the number of pre-cached chunks per ICN cache on the avg. transmission times

varying amounts of pre-cached chunks per ICN cache on the transmission times. It is visible that the increase of pre-cached chunks per cache node has a positive influence on the transmission times. In all scenarios the transmission times decline. This effect manifests itself, since the amount of content located closer to the requesting node is increased. Hence, content needs to be transmitted on shorter paths, which leads to the reduced transmission delays. It is visible that this parameter does not provide an

essential change in the relation between the scenarios. All cases are influenced in the same manner.

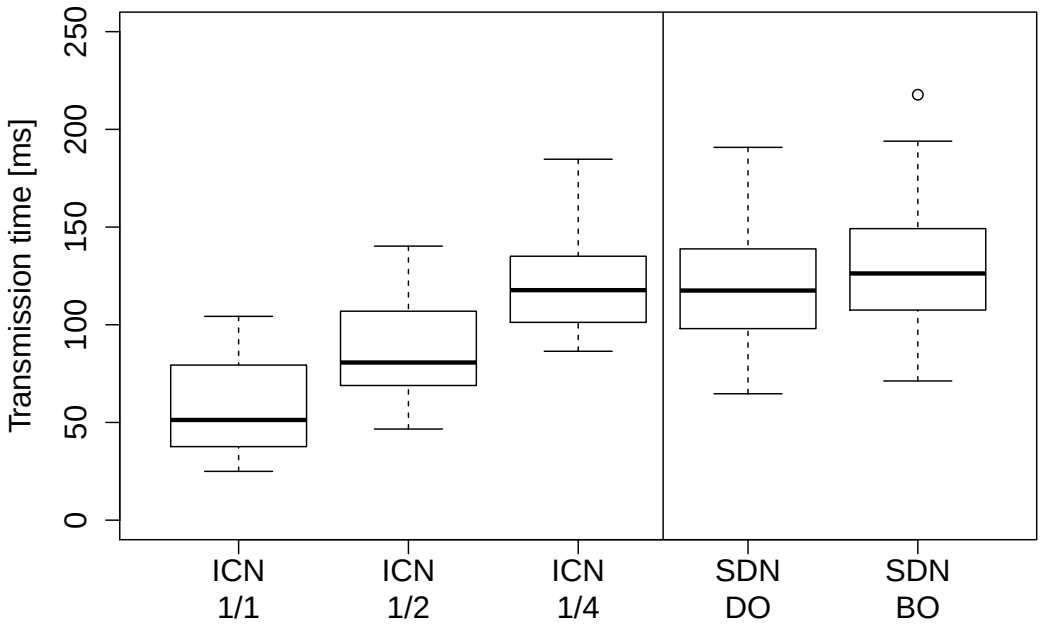


Figure 6.5: Effects of the ICN to SDN cache fill ratio on the avg. transmission times

Figure 6.5 illustrates the effects of varying ratios of pre-cached content chunks between the ICN and the SDN cases. Since this parameter effects the ICN measurements, only the ICN variations are shown. If the number of content chunks is the same in both cases, the ICN is in the average case able to respond roughly twice as fast as the SDN approach does. This advantage is reduced the further the relative amount of pre-cached content chunks is decreased. With a ratio of 1/4th of the content cached in the SDN case, the average transmission times of the ICN are about the same, whilst the distribution of the transmission times in the SDN cases are broader. The transmission times of the DO case are only slightly better than those of the BO case.

This behaviour is comprehensible since what happens is in general the reduction of the size of the ICN cache, which is somehow similar to the measurement depicted in Figure 6.3 with the difference that only the ICN cache size is varied while it stays fixed in the DO and BO case. Consequently the relative difference between ICN and the SDN cases decreases.

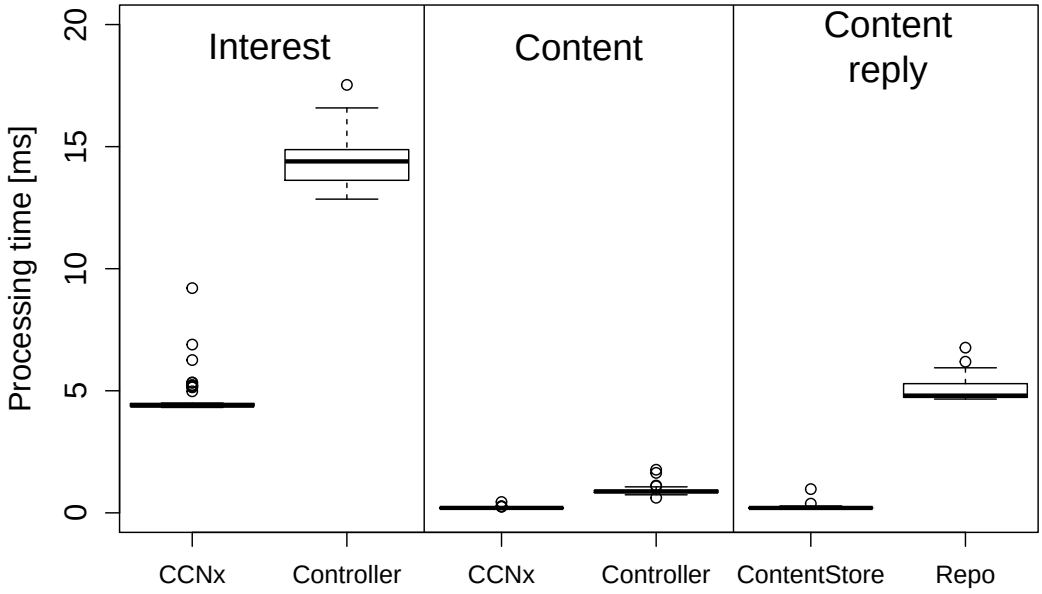


Figure 6.6: Packet processing times

6.3.2 Processing times

To gain a deeper understanding of the resulting transfer times, we performed additional time measurements. In Figure 6.6 the processing times for Interest as well as Content packets of native CCNx and the ICN-SDN controller are presented. Further, we depict the response times in case the requested content is available in the cache as well as when the requests have to be processed by the *Repository* to be satisfied.

Interest We measure the time it takes for an Interest to pass a CCNx node as well as the time it takes for an Interest to be forwarded towards an adjacent node, from entering the node until it leaves the node. Thereby, the usual ICN processing is carried out. A *Content Store* lookup is performed. Subsequent the PIT is consulted if request aggregation can be performed and afterwards the FIB is used to determine the next hop node. Finally, the Interest is emitted on the determined interface.

In the ICN-SDN case, the plotted values show the time from the reception of the Packet_In at the controller until the associated Packet_Out is transmitted. In this time frame, a content serving node is determined, the linear forwarding path is collapsed into the ingress and the egress node and the corresponding Flow_Mod for the packet rewriting are send, before the Packet_Out is issued.

It is visible that the processing times of our controller are nearly three times higher than the optimized CCNx processing. The results get even worse, considering that the processing time on the data plane is already included in the ICN case. The data is actually already forwarded, while in the SDN case the controller has just calculated where to forward the packet to. The actual data plane forwarding operations still have to be performed. On the other hand, this costly controller processing only has to be performed once for the entire path to be determined and set up. The illustrated ICN processing, on the other hand, is performed on each and every hop.

Content The processing of content responses in the CCNx case covers the time span from the packet entering an interface until it is forwarded towards a next hop node. Primarily a content name lookup in the PIT is performed to determine the output interface for the packet before subsequently dispatching it. The ICN-SDN controller on the other side also just needs to lookup the content name to determine the SDN egress port and the associated rewriting parameters before the corresponding Packet_Out is issued.

The processing times for content objects in the ICN-SDN controller are also higher than in the ICN case. Nevertheless, does this difference manifest itself in a different scale. 0.21 ms avg. in the ICN case versus 0.91 ms avg. in the SDN case.

Content reply To also get an impression of the response times of the *Content Store* compared to the ICN Repository, we issued requests for content that was either available in the cache or had to be fetched from the *Repository*. If the content is available in the *Content Store*, the CCNx daemon skips the PIT and FIB lookups because the cache lookup already yields the requested content. However, if the cache misses the content in addition to the *Content Store* lookup, the FIB is queried. Since the Interests needs to be forwarded towards the *Repository*, the registration of the forwarded Interest in the PIT is performed.

In case of the content being available in the *Content Store*, the delay until the content is transmitted out of the nodes interface is comparable to the content forwarding times, which is reasonable, since the content is available in memory and the *Content Store* lookup is in general the first step for the CCNx daemon to perform. However, getting a response from the *Repository* takes slightly longer than the Interest forwarding time, which is plausible since the whole cycle of *Content Store* check, PIT check and FIB lookup have to be performed to subsequently hand the request over to the local *Repository*. Additionally the content forwarding time is added on top because the response also has to pass the content processing.

[1] For Open vSwitch, according to [62]

Component	Multiplier	Time *[ms]*
Interest processing delay	h	4.70
Content processing delay	h	0.21

(a) ICN case

Component	Multiplier	Time *[ms]*
Interest processing delay	1	14.39
Content processing delay	1	0.91
Forwarding delay	h	< 0.05[1]
Control plane delay	2	5.00

(b) Bandwidth Optimized SDN case

Table 6.2: Average per chunk processing delay components (*h = number of hops*)

Table 6.2 shows the itemized delay components, which are introduced in the dissemination process. Table 6.2a depicts the average delays for the ICN case along with their multiplier. Since CCNx performs hop-by-hop forwarding, the Interest is processed by each hop, the same applies to the resulting content packets. Thus, the measured times have to be multiplied with the amount of traversed hops (h). Table 6.2b shows a different composition of the times. The Interest and content processing delays are only introduced once per request-response cycle (considering a non-forking case). The forwarding delay is introduced by each SDN switch to forward the packet towards the next hop. Additionally, the control plane delay has to be considered. In Table 6.2b we considered the general 5 ms control plane delay we defined, once for the Packet_In from the switch and once for the Packet_Out towards the switch.

The Interest processing delay of around 4.7 ms mean in the ICN case might quickly add up on longer forwarding paths since the ICN forwarding is performed hop-by-hop. The Open vSwitch forwarding delay is according to [62] below 50 µsec, which is less impacting on longer paths. figure 6.7 clearly depicts this correlation. Processing times for the ICN case rise quickly while the SDN BO increase only slowly.

To work out the actual transmission delays, the data plane connection latencies have to be taken into account as well. The times for the actual request and response propagation is not included. However, the paths between the caches in the SDN cases and the ICN case are most likely diverse and thus not directly related.

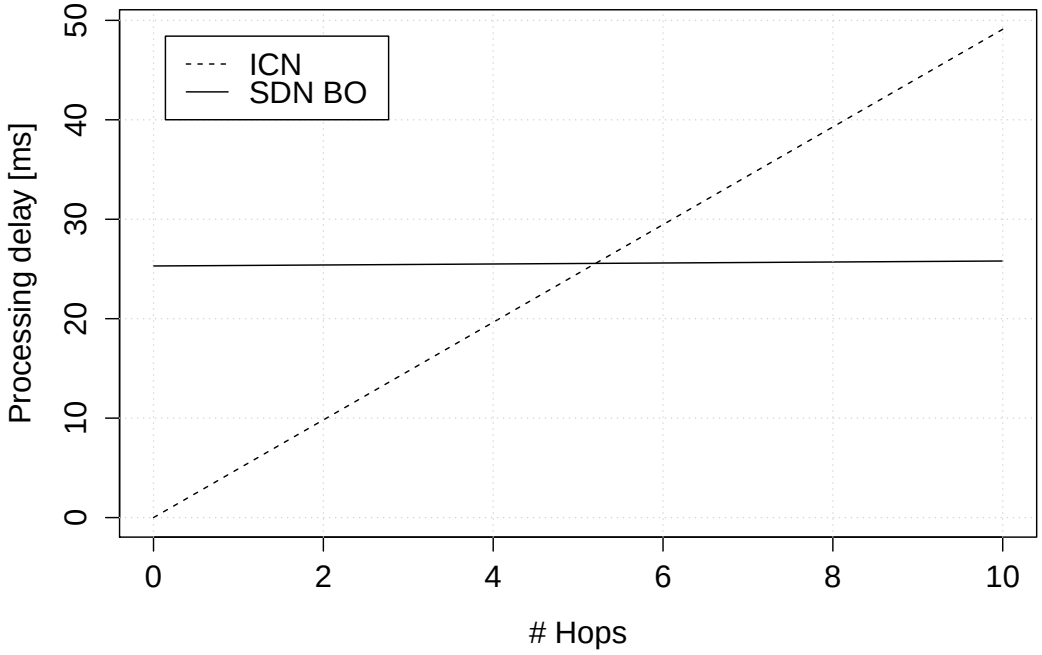


Figure 6.7: Hop count dependent processing delay

6.3.3 Data plane load

Again, we evaluate the influence of the parameters of Table 6.1, namely the ICN to SDN cache fill ratio, fork factor and the number of cached chunks. This time, we focus on the network load created by our advanced approach compared to the ICN forking case.

Therefore, the following graphs show the relative data plane load for the SDN BO and DO forking scenarios compared to the respective ICN scenarios. The data plane load of each set of runs, consisting of the ICN, BO and DO cases with similar cache pre-population and origin placement, are put into relation. The ICN data plane load is taken as the reference value and thus defines the 100% mark. The BO and DO cases are drawn according this value. Multiple of these sets are grouped into the correlating parameter category to subsequently draw the box plots shown in the following.

Figure 6.8 depicts the influence of the ICN to SDN cache fill ratio on the relative network load between the SDN BO and DO case and the ICN forking case. It is visible that the variance in the mean network load of the BO and DO case compared to the ICN case is reduced by decreasing the relative amount of ICN cached content. In the

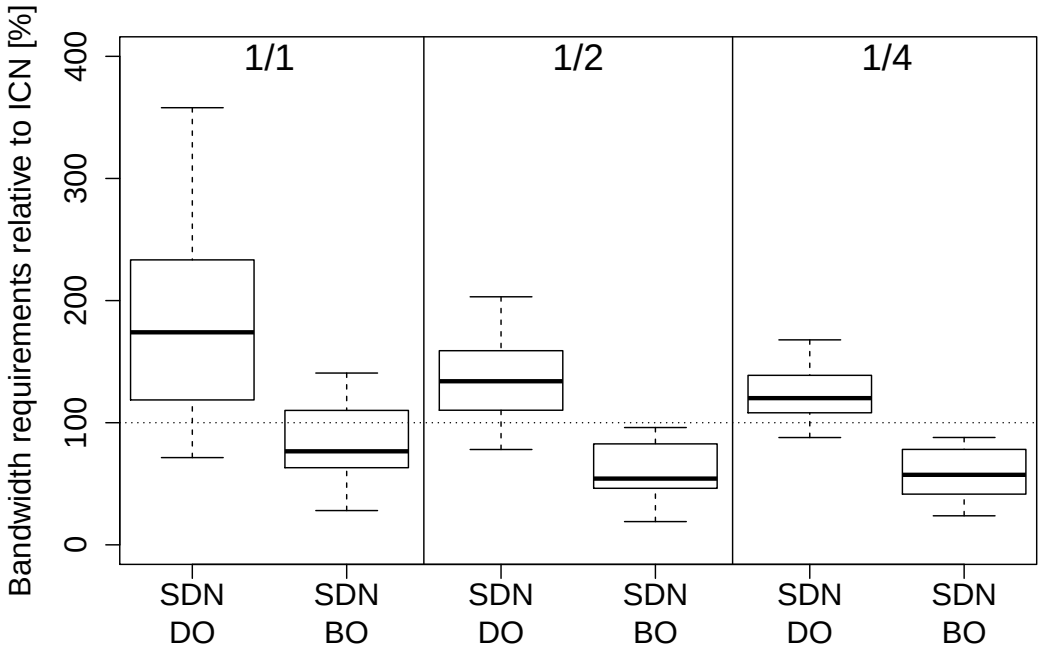


Figure 6.8: Effects of the ICN to SDN cache fill ratio on the network load

worst of the DO runs the network load is nearly four times higher, but in the best case also better than the network load in the ICN case. The results are broadly spread when utilizing similar cache pre-population amounts. This effect is reduced by changing the pre-population ratio. The values of the BO runs decline through the increased ratio. They are already completely below the ICN threshold starting at a ratio of 1/2.

The displayed values depict that the ICN to SDN cache fill ratio has a noticeable impact on the generated data plane network load. This behavior arises since each ICN cache node holds a smaller fraction of the requested content. Hence, the request in the ICN case has to travel longer distances until it reaches a copy of the requested content, which consequently creates a higher network load when the content is delivered all the way back. In the ICN case, each traversed node checks its local cache, whereas in the SDN case just the selected cache nodes, which might be some hops away, are utilized. This results in a general disadvantage for the SDN cases on the first sight. By tuning the ICN to SDN cache fill ratio, we reduce the per cache hit probability for the ICN and thus the packets have to traverse an increased amount of hops, which results in an increased network load.

Adjusting the cache size in the ICN and SDN cases individually is a valid way for the evaluation of our approach. The caches throughout the network can be populated more purposeful due to the centralized knowledge, which increases the network wide cache

hit probability. In the SDN cases, the content is cached in a node or small group of nodes explicitly dedicated to the actual namespace or content type while in the ICN case requested content is stored multiple times on the traversed ICN nodes. This uncoordinated caching in fact reduces the overall storage capacity for individual content and the lifetime of content within each of the caches.

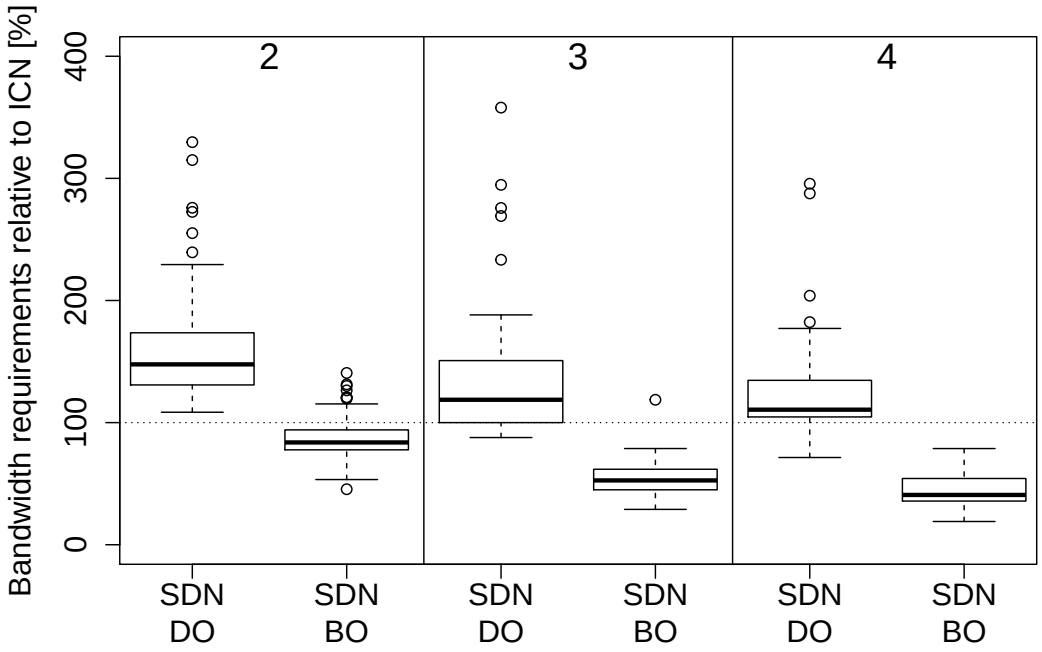


Figure 6.9: Effects of the forking factor (2, 3 & 4) on the network load

Figure 6.9 illustrates the effects that different numbers of forks have on the data plane load. While in the 2 fork scenario all runs of the DO case created more load than the ICN runs, even 25% of the DO runs in the 4 fork scenario created less network load. While in the 2 fork scenario already creating less data plane load in almost every run. The BO in fact still outperforms the ICN case in all runs for the 4 fork runs.

The resulting numbers can be explained via the diverse paths that the requests take in the ICN case. If a cache hit is achieved in a diverse part of a path, multiple replies are triggered, which are delivered up to the forking node. On the forking node, all except one reply are stopped due to the elimination of the PIT state by the first arriving reply. Other, if the caches are not populated with the requested content, the requests of the different forks merge latest at the last hop towards the origin node. Duplicated request packets will be ruled out by that node, when requests with the same nonce value arrive, the transmission of multiple responses is suppressed. If, however, in the SDN BO case multiple replies are triggered, all additional replies are stopped at the first SDN switch be-

fore entering the network core. Whereas, in case of the DO scenario, multiple responses will likely traverse the network. The rule adjustment that leads to the suppression of superfluous replies takes some time. During this time additional responses might already be forwarded.

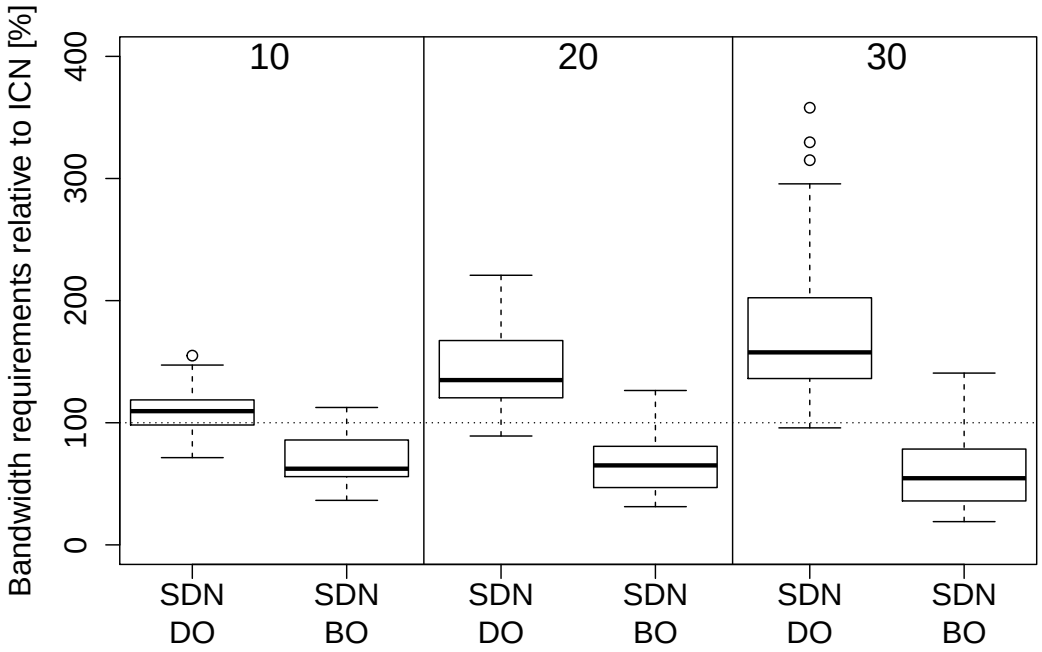


Figure 6.10: Effects of the number of pre-cached chunks per ICN cache on the network load

Figure 6.10 shows the effects that occur when the amount of pre-cached chunks per cache are varied. For the DO case the resulting data plane load relative to the ICN case increases while the BO case benefits from the rising amount of pre-cached content chunks per cache. In the 10 content chunks per cache case, the introduced network load of the DO runs is mostly higher than the mean ICN load. It even increases and spreads up to nearly 300% of the ICN load in the 30 chunks per cache case, leaving aside the outliers. The BO case shows a tendency towards declining data plane load while at the same time being increasingly distorted.

Through the increase in pre-cached content chunks per cache, the probability of content being available in multiple caches raises. This results in even more chunk transmissions in the case of DO. In the BO case the increase does not influence the network load. Only one response is always forwarded through the network. However, the probability of a nearby cache holding the content is increased, thereby the number of hops towards a cache hit is reduced as well as the overall network load.

6.3.4 Control plane load

Throughout the measurements we also monitored the imposed control plane load. Depicted in Figure 6.11 are the inter-quartile means of the measured values over the complete set of runs. We split up the control plane traffic into Packet_Ins used to transport Interest packets (*Pkt_In - Interest*) and content packets (*Pkt_In - Content*) towards the ICN-SDN controller as well as other OpenFlow messages (*OpenFlow w/o Pkt_In*). This class contains the ICN-SDN related Packet_Out, Flow_Mod and Flow_Remove messages.

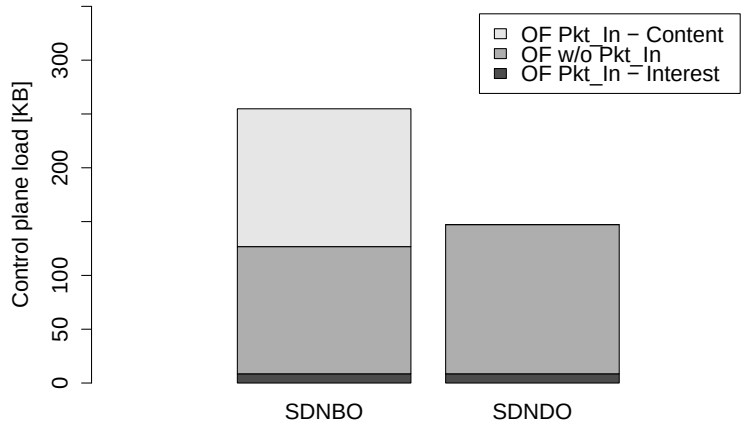


Figure 6.11: Control plane load derived via the inter-quartile mean over the entire measurement set

Taking a look at the SDN BO numbers, it is visible that the share of control plane load caused by Packet_Ins triggered through CCNx Interests is quite small, it does only account for 3.36% of the load. The Packet_Ins caused by content packets on the other hand amount to 50.26% of the control plane load. Finally, the remaining 46.36% are the share that allots to the remaining OpenFlow messages mentioned above. The amount of Interest driven Packet_Ins remains the same in both cases, which is plausible since we request 50 chunks of content in each run that all trigger one of those messages. The OpenFlow Packet_Out, Flow_Mod and Flow_Removed traffic amount is higher in the DO runs. This effect is caused by the Flow_Mod packets that carry the content ingress rule. They carry not only the request egress packet processing information but also the content ingress information for each request egress node. In the BO case, this information is only send to the specific node that is expected to forward the packet. All other nodes are provided with discard rules that do not need the additional packet rewriting information. Furthermore, the flow rules are configured with an idle timeout

and the instruction to notify the controller about the expiration of the rule. Setting up rules that carry the flow removed notification bit also results in additional control plane packets, since the controller is informed as soon as the rule expires. Finally, the Packet_In message triggered by content carrying packets are of course not present in the SDN DO case since the required flow rules for the content forwarding are already provisioned when the content arrives.

The results show that the volume of content packet triggered Packet_Ins cause 50% of the control plane traffic in the SDN BO case. This value is of course dependent on the fork count and the amount of cache hits per request. Each content packet that arrives at the SDN edge is delivered as a whole towards the controller. In general this behaviour can be optimized by only sending the first part of the packet, containing the name information towards the controller. This piece of information is sufficient for the controller to evaluate on how to further handle the packet. Hence, in the worst case, according to our parameters with four branches that might yield the data and a cache hit ratio of 30/50 chunks, the amount of content initiated Packet_Ins should theoretically be (30/50*4 =) 2.5 times the load of the control plane load created by the Interest Packet_Ins. This assumption is made, since the Interest packet also includes the requested name. The amount of data is thus sufficient to read the complete content name from the responses as well. However, this approach introduces a new challenge when it comes to ICN implementations that utilize variable length content names like CCNx does. The name can vary in size, there might not even be an upper boundary for the length of a name. Hence, the question arises on how much data per packet is needed for the controller to base its decisions on. Utilizing fixed length content names instantly resolves this issue because it is clearly defined where the content name ends.

6.4 Evaluation Summary

The presented results show that in terms of transmission times, the ICN forking outperforms the ICN-SDN forking approach in most of the constellations. It was only possible to achieve comparable transmission times by reducing the ratio of pre-cached content per node compared to the SDN cases. However, we mentioned that we expect less caches with higher capacity to be deployed in certain parts of the network, which in combination with the ability to work around the purely opportunistic on-path caching might even result in a higher cache hit probability in bigger networks.

According to the processing times of ICN packets, our ICN-SDN controller is also slower. However, the controller determines and provisions the rules for multiple forwarding elements at once, which reduces the per hop processing times since only forwarding rule matching has to be performed. ICN forwarding nodes in contrast have to perform more costly cache, PIT and FIB lookups at each hop. Nevertheless, we do ex-

pect that both the implementations of CCNx as well as our ICN-SDN controller still have potential for performance improvements since the actual implementations do not base on production-ready code.

The ICN-SDN approach is in term of the SDN BO instantiation able to compete with and in most cases even provide better results than the ICN forking. The SDN DO numbers are almost always worse then the results of the ICN forking case. Admittedly, this is what we expected from the design phase on since we aimed at the reduction of transmission delays, at the expense of the network bandwidth. However, the DO mode does not in general provide better transmission times compared to the ICN forking.

Chapter 7

Summary

We will now close by summarizing and concluding this work as well as providing an outlook for future research directions.

7.1 Conclusion

This work shows that the evolutionary path from IP to ICN is feasible via SDN. The introduced requirements where met, except the controller redundancy, which did not perceive sufficient attention throughout this work and thus requires further investigation. However, compared to earlier work on the integration of ICN and SDN does our approach not require changes to the network stack of involved ICN nodes or modifications to the ICN implementations. Hence, the common utilization of network protocols like IP, TCP and UDP is still possible.

The centralized view that the SDN controller has on the network provides certain benefits. It is able to spot bottleneck links and can subsequently re-rout forwarding paths, avoiding these links. Further, through the collection of additional information about the deployed ICN cache nodes it is also able to avoid steering requests towards these overloaded nodes. Via the capabilities of content namespaces identification and the ability to steer requests through the SDN network, the controller is even able to cache certain name spaces or content types on particular groups of ICN cache nodes. Further, did we introduce the capability of ICN packet forking within the SDN, to enable the network wide aggregation of ICN requests, as well as the parallel querying of multiple caches at once.

A theoretical analysis was conducted that compared CCNx packet forking with ICN-SDN packet forking. This analysis revealed that the forking of requests provides advantages according the data plane load. Less network bandwidth is consumed in the ICN-SDN case compared to CCNx forking. This advantage already manifests itself when utilized with two forks and increases with the number of forks. Beyond that, we roughly

estimated the amount of flow rules needed by our approach. Therefore, we defined different roles which are needed in the forwarding process, namely ingress, egress, fork and forwarding. The egress role was thereby identified as the most demanding role in terms of the number of flow rules. Two rules are necessary per each content transfer, one for the request and the other for the response forwarding. We considered the NEC ProgrammableFlow switch PF5240 as an example SDN device. Doing so, our approach would theoretically support 32K-80K concurrent content transfers per egress switch.

The evaluation results show that with our actual implementation we are not able to compete with ICN request forking in terms of response times, but the amount of data transferred on the data plane can be reduced. Further, the results revealed that the control plane load, dependent on the utilized ICN protocol, is quite high. Hence, it is a mechanism that will unlikely be performed for all requests within an ICN-SDN domain. It is advisable to not use this mechanisms for each and every content dissemination but for initial content requests in environments where the controller has no full knowledge of cached content. Thereby requests are forked until a cache, containing the required content is found, subsequently all requests can solely be directed towards one of the responding nodes without the utilization of the ICN request forking approach.

As a side effect, we where able to identify properties which are desirable or even recommended to be met, in case ICN is operated over SDN.

Name information Each of the packets that are delivered through the network should represent a self-contained ICN packets. Hence, the type of information – request or response – and the name of the content have to be present in each packet. Otherwise no intermediate node will be able to make appropriate decisions if necessary.

Fragmentation handling To comply with the first bullet point, a proper protocol layer has to be in place that chops data into MTU compliant fragments while preserving the requirement of each packet carrying the type and content name attributes.[1]

Label forwarding For the forwarding of ICN packets through the SDN network a fixed length tag or label attached to the packets is necessary. Since fully generic packet forwarding is according to [25] not be feasible in the near future, matchable fixed length identifiers are needed to perform cost efficient ICN-SDN forwarding.

7.2 Future work

One of the outstanding challenges that exist in conjunction with our approach is the consistent provisioning of OpenFlow rules. If the forwarding rules are not already send out

[1] See also [30].

to the upstream SDN element and committed to their TCAM, the packet may trigger multiple, unnecessary Packet_In messages at the controller and thus results in an increased load on the control plane. Quite a lot of work already exists that elaborate on the problem of network wide forwarding rule consistency, like [63, 64], to name just two. Hence, we leave this challenge aside for the time being.

Also the controller is expected to suffer a high burden, due to the load caused by the inspection of bulks of Packet_In messages. The content names extraction, as well as the path calculation and provisioning. Therefore it is at some point inevitable to come up with a controller partitioning and redundancy scheme that allows multiple controllers to operate as one logical entity. We have not yet payed much attention to the controller redundancy, but we keep this in mind as an important point for the applicability of the approach and will focus on this in our future research.

More research has to be performed on increasing scalability of the approach. The burden that is put on the controller needs to be reduced. A promising attempt might be the utilization of software based forwarding at the edge of the network as already mentioned in the SDIA proposal [25]. This would easily allow for whole packet generic matching and thus introduce some more flexibility in the matching and forwarding process. The European Telecommunications Standards Institute (ETSI) is actually working on Network Function Virtualisation (NFV) [65, 66], a generalized framework for the implementation of network functions via virtualization techniques on general purpose computer hardware. The goal of NFV is to allow for rapid and cost efficient deployment of network functions like for instance firewalls, Virtual Private Network (VPN) services and could thereby also provide benefits for the deployment of ICN services.

Bibliography

[1] M. Gritter and D. R. Cheriton, “An Architecture for Content Routing Support in the Internet,” in *Proc. USITS’01*. Berkeley, CA, USA: USENIX Association, 2001, pp. 4–4.

[2] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, “A Data-Oriented (and beyond) Network Architecture,” *SIGCOMM Computer Communications Review*, vol. 37, no. 4, pp. 181–192, 2007.

[3] L. Zhang, D. Estrin, J. Burke, V. Jacobson, and J. D. Thornton, “Named Data Networking (NDN) Project,” PARC, Tech.report ndn-0001, 2010.

[4] V. Jacobson, D. K. Smetters, J. D. Thornton, and M. F. Plass, “Networking Named Content,” in *Proc. of the 5th Int. Conf. on emerging Networking EXperiments and Technologies (ACM CoNEXT’09)*. New York, NY, USA: ACM, Dec. 2009, pp. 1–12.

[5] “The PSIRP Homepage,” http://www.psirp.org, 2012.

[6] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, “LIPSIN: Line Speed Publish/Subscribe Inter-networking,” in *Proc. of the ACM SIGCOMM 2009*. New York, NY, USA: ACM, 2009, pp. 195–206.

[7] B. Ahlgren *et al.*, “Second NetInf Architecture Description,” 4Ward EU FP7 Project, Tech.report D-6.2 v2.0, 2010.

[8] M. Vahlenkamp, “Information-Centric Networking - a related work survey,” HAW Hamburg, Tech. Rep., 2012. [Online]. Available: http://inet.cpt.haw-hamburg.de/teaching/ss-2012/master-projects/markus_vahlenkamp_aw2.pdf

[9] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlmann, “A Survey of Information-Centric Networking (Draft),” Dagstuhl Seminar Proceedings, Tech. Rep. 10492, 2011.

[10] D. Kutscher, S. Eum, K. Pentikousis, I. Psaras, D. Corujo, and D. Saucez, “ICN Research Challenges,” IETF, Internet-Draft – work in progress 00, February 2013.

[11] F. Hermans, E. Ngai, and P. Gunningberg, "Global source mobility in the content-centric networking architecture," in *Proceedings of the 1st ACM workshop on Emerging Name-Oriented Mobile Networking Design - Architecture, Algorithms, and Applications*, ser. NoM '12. New York, NY, USA: ACM, 2012, pp. 13–18. [Online]. Available: http://doi.acm.org/10.1145/2248361.2248366

[12] W. Wong and P. Nikander, "Secure Naming in Information-Centric Networks," in *Proc. of Re-Architecting the Internet Workshop (ReARCH '10)*. New York, NY, USA: ACM, 2010, pp. 12:1–12:6.

[13] T. Aura, "Cryptographically Generated Addresses (CGA)," IETF, RFC 3972, Mar. 2005.

[14] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker, "Naming Things with Hashes," IETF, RFC 6920, April 2013.

[15] "The Named Data Networking Homepage," http://www.named-data.net, 2013.

[16] PARC, "The CCNx Homepage," http://www.ccnx.org, 2012.

[17] L. Wang, A. K. M. M. Hoque, C. Yi, A. Alyyan, and B. Zhang, "OSPFN: An OSPF Based Routing Protocol for Named Data Networking," Tech. Rep., Jul. 2012. [Online]. Available: http://www.named-data.net/techreport/TR003-OSPFN.pdf

[18] "The CCNx Technical Documentation," http://www.ccnx.org/documentation/ccnx-technical-documentation-index, 2013.

[19] B. Davie, "Network Virtualization: Delivering on the Promises of SDN." Open Networking Summit, 2013.

[20] H. Khosravi and T. Anderson, "Requirements for Separation of IP Control and Forwarding," IETF, RFC 3654, Nov. 2003.

[21] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," *SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 69–74, Mar. 2008. [Online]. Available: http://doi.acm.org/10.1145/1355734.1355746

[22] Z. Wang, T. Tsou, J. Huang, X. Shi, and X. Yin, "Analysis of Comparisons between OpenFlow and ForCES," IETF, Internet-Draft – expired 01, Mar. 2012.

[23] ONF, "Software-Defined Networking: The New Norm for Networks," Open Networking Foundation, Tech. Rep., 2012.

[24] *The OpenFlow Switch Specification 1.3.1*, Open Network Foundation Std. [Online]. Available: https://www.OpenNetworking.org

[25] B. Raghavan, M. Casado, T. Koponen, S. Ratnasamy, A. Ghodsi, and S. Shenker, "Software-Defined Internet Architecture: Decoupling Architecture from Infrastructure," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*, ser. HotNets-XI. New York, NY, USA: ACM, 2012, pp. 43–48. [Online]. Available: http://doi.acm.org/10.1145/2390231.2390239

[26] D. Jen, M. Meisel, H. Yan, D. Massey, L. Wang, B. Zhang, and L. Zhang, "Towards a New Internet Routing Architecture: Arguments for Separating Edges from Transit Core," *HotNets-VII*, Oct. 2008.

[27] D. G. Andersen, H. Balakrishnan, N. Feamster, T. Koponen, D. Moon, and S. Shenker, "Accountable Internet Protocol (AIP)," in *Proceedings of the ACM SIGCOMM 2008 conference on Data communication*, ser. SIGCOMM '08. New York, NY, USA: ACM, 2008, pp. 339–350. [Online]. Available: http://doi.acm.org/10.1145/1402958.1402997

[28] N. Blefari-Melazzi, A. Detti, G. Morabito, S. Salsano, and L. Veltri, "Information Centric Networking over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed," *CoRR*, vol. abs/1301.5933, 2013.

[29] L. Veltri, G. Morabito, S. Salsano, N. Blefari-Melazzi, and A. Detti, "Supporting information-centric functionality in software defined networks," in *ICC*, 2012, pp. 6645–6650.

[30] A. Detti, N. Blefari Melazzi, S. Salsano, and M. Pomposini, "CONET: A Content Centric Inter-Networking Architecture," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*, ser. ICN '11. New York, NY, USA: ACM, 2011, pp. 50–55. [Online]. Available: http://doi.acm.org/10.1145/2018584.2018598

[31] T. Bates, E. Chen, and R. Chandra, "BGP Route Reflection: An Alternative to Full Mesh Internal BGP (IBGP)," IETF, RFC 4456, Apr. 2006.

[32] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, and N. Blefari-Melazzi, "Transport-layer issues in information centric networks," in *Proceedings of the second edition of the ICN workshop on Information-centric networking*, ser. ICN '12. New York, NY, USA: ACM, 2012, pp. 19–24. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342493

[33] S. Salsano, A. Detti, N. Blefari-Melazzi, and M. Cancellieri, “ICTP - Information Centric Transport Protocol for CONET ICN,” IETF, Internet-Draft – work in progress 01, Nov. 2012.

[34] P. Fransson and A. Jonsson, “End-to-end measurements on performance penalties of IPv4 options,” in *Global Telecommunications Conference, 2004. GLOBECOM ’04. IEEE*, vol. 3, 2004, pp. 1441–1447 Vol.3.

[35] D. Trossen and G. Parisis, “Designing and Realizing an Information-Centric Internet,” *Communications Magazine, IEEE*, vol. 50, no. 7, pp. 60–67, 2012.

[36] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, “LIPSIN: line speed publish/subscribe inter-networking,” in *Proceedings of the ACM SIGCOMM 2009 conference on Data communication*, ser. SIGCOMM ’09. New York, NY, USA: ACM, 2009, pp. 195–206. [Online]. Available: http://doi.acm.org/10.1145/1592568.1592592

[37] D. Syrivelis, G. Parisis, D. Trossen, P. Flegkas, V. Sourlas, T. Korakis, and L. Tassiulas, “Pursuing a Software Defined Information-Centric Network,” in *Proceedings of the 2012 European Workshop on Software Defined Networking*, ser. EWSDN ’12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 103–108. [Online]. Available: http://dx.doi.org/10.1109/EWSDN.2012.20

[38] B. J. Ko, V. Pappas, R. Raghavendra, Y. Song, R. B. Dilmaghani, K.-w. Lee, and D. Verma, “An Information-Centric Architecture for Data Center Networks,” in *Proceedings of the second edition of the ICN workshop on Information-centric networking*, ser. ICN ’12. New York, NY, USA: ACM, 2012, pp. 79–84. [Online]. Available: http://doi.acm.org/10.1145/2342488.2342506

[39] M. Wählisch, T. C. Schmidt, and M. Vahlenkamp, “Backscatter from the Data Plane – Threats to Stability and Security in Information-Centric Network Infrastructure,” *Computer Networks*, 2013. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2013.07.009

[40] ——, “Lessons from the Past: Why Data-driven States Harm Future Information-Centric Networking,” in *Proc. of IFIP Networking*. Piscataway, NJ, USA: IEEE Press, 2013.

[41] ——, “Bulk of Interest: Performance Measurement of Content-Centric Routing,” in *Proc. of ACM SIGCOMM, Poster Session*. New York: ACM, August 2012, pp. 99–100. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p99.pdf

[42] "The potaroo.net Homepage," http://bgp.potaroo.net, Nov. 2013.

[43] "The Domain Name Industry Brief," Apr. 2013. [Online]. Available: http://www.verisigninc.com/assets/domain-name-brief-april2013.pdf

[44] "The Domain Name Industry Brief – Q3 Highlights," Apr. 2013. [Online]. Available: http://www.verisigninc.com/assets/domain-name-brief-april2013.pdf

[45] http://googleblog.blogspot.it/2008/07/we-knew-web-was-big.html, 2008.

[46] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: Voice-over Content-centric Networks," in *Proceedings of the 2009 Workshop on Re-architecting the Internet*, ser. ReArch '09. New York, NY, USA: ACM, 2009, pp. 1–6. [Online]. Available: http://doi.acm.org/10.1145/1658978.1658980

[47] E. Rosen, A. Viswanathan, and R. Callon, "Multiprotocol Label Switching Architecture," IETF, RFC 3031, January 2001.

[48] S. Alexander and R. Droms, "DHCP Options and BOOTP Vendor Extensions," IETF, RFC 2132, March 1997.

[49] M. Vahlenkamp, F. Schneider, D. Kutscher, and J. Seedorf, "Enabling Information-Centric Networking in IP Networks Using SDN," in *Proc. of IEEE SDN4FNS*, Nov. 2013.

[50] Y. Rekhter, R. G. Moskowitz, D. Karrenberg, G. J. de Groot, and E. Lear, "Address Allocation for Private Internets," IETF, RFC 1918, February 1996.

[51] N. Corporation, "The NEC ProgrammableFlow PF5240 Switch Datasheet." [Online]. Available: http://www.nec.com/en/global/prod/pflow/images_documents/ProgrammableFlow_Switch_PF5240.pdf

[52] "The Trema Homepage," http://trema.github.io/trema, 2013.

[53] "The Trema-Apps Homepage," http://trema.github.io/trema/apps, 2013.

[54] "The Mininet Homepage," http://mininet.org, 2013.

[55] R. Rosen, "Resource management: Linux kernel Namespaces and cgroups," Haifux, Tech. Rep., May 2013. [Online]. Available: http://www.haifux.org/lectures/299/netLec7.pdf

[56] C. M. Cabral, C. E. Rothenberg, and M. F. Magalhães, "Mini-CCNx: Fast Prototyping for Named Data Networking," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '13. New York, NY, USA: ACM, 2013, pp. 33–34. [Online]. Available: http://doi.acm.org/10.1145/2491224.2491236

[57] ——, "Reproducing Real NDN Experiments Using mini-CCNx," in *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '13. New York, NY, USA: ACM, 2013, pp. 45–46. [Online]. Available: http://doi.acm.org/10.1145/2491224.2491242

[58] "The Mini-CCNX Homepage," https://github.com/carlosmscabral/mn-ccnx, 2013.

[59] "The Open vSwitch Homepage," http://openvswitch.org, 2013.

[60] "The Wireshark Homepage," https://www.wireshark.org, 2013.

[61] "The IGen Homepage," http://igen.sourceforge.net, 2013.

[62] C. Rotsos, N. Sarrar, S. Uhlig, R. Sherwood, and A. W. Moore, "OFLOPS: an open framework for openflow switch evaluation," in *Proceedings of the 13th international conference on Passive and Active Measurement*, ser. PAM'12. Berlin, Heidelberg: Springer-Verlag, 2012, pp. 85–95. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-28537-0_9

[63] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, "Consistent Updates for Software-De?ned Networks: Change You Can Believe In!" in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks*, ser. HotNets-X. New York, NY, USA: ACM, 2011, pp. 7:1–7:6. [Online]. Available: http://doi.acm.org/10.1145/2070562.2070569

[64] P. Perešíni, M. Kuźniar, N. Vasić, M. Canini, and D. Kostić, "OF.CPP: Consistent Packet Processing for OpenFlow," in *Proceedings of HotSDN'13*, Aug 2013.

[65] "The NFV Homepage," http://www.etsi.org/technologies-clusters/technologies/nfv, 2013.

[66] ETSI, *Network Functions Virtualisation – Introductory White Paper*, European Telecommunications Standards Institute, Oct. 2012. [Online]. Available: http://portal.etsi.org/NFV/NFV_White_Paper.pdf

Appendix A

Basic approach evaluation

To analyse the applicability of our basic approach and to get a first impression of the expectable performance values, we setup an emulated network environment to run experiments with our ICN-SDN approach in comparison to a simple bridged configuration also consists of multiple CCNx-aware hops.

A.1 Emulation setup & scenarios

The network as depicted in Figure A.1 is used for a quick evaluation of the basic approach. It is configured with individual link delays via *netem*, the Linux integrated network emulation functionality. Edge links connecting CCNx nodes to the network are configured with 10 ms delay while the overall delay introduced by the linear arrangement of switches also add up to 10 ms. The delay between each Open vSwitch and the controller depends on the executed scenario. When utilizing the *CCNx-SDN controller*, we apply a 5 ms delay to account for the centralization of the controller, whereas no delay is applied for the reference measurements, hence, to mimic usual standalone switches. All edge and core links carry a capacity of 10 Mbit/s each.

We use the CCNx included tools *ccnsendchunks2* and *ccncatchunks* to run the different comparative measurement scenarios. Via *ccnsendchunks2* we published fixed size files from one CCNx node in the network that we request via *ccncatchunks* on another CCNx node.

Within all scenarios H1 is used as the content producing node, while H3 is running the content consuming application. Scenarios that we evaluated include at first the *(i)* CCNx-SDN approach in which our Trema *CCNx-SDN controller* is employed. The CCNx routing tables default entry (ccnx:/) of all nodes is thus configured to point at the SDN-IP address *(ii)* 1 Hop Bridged in which all switches connect to the standard Open vSwitch controller, which is performing usual switching functionalities while H3 holds the direct association

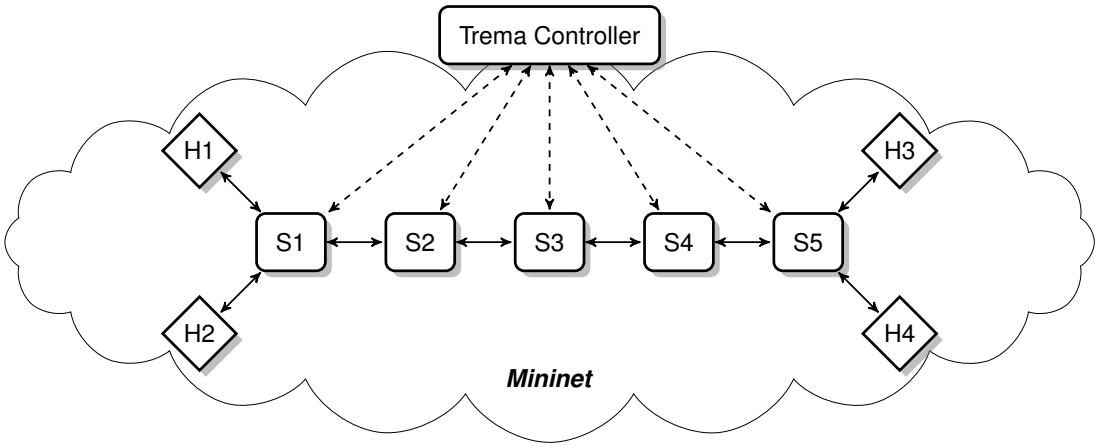


Figure A.1: Evaluation environment

between the namespace used for the measurement and the IP address of H1 *(iii)* 2 Hop Bridged where an additional CCNx hop is introduced into the path. Hence, the name entry for our measurement points to CCNx node H2, which in turn points eventually to H1, the content serving node.

The 1 Hop Bridged case represents the optimal scenario of non-SDN operation in which the content consumer is 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, additional hops that perform CCNx routing decisions will be required to reach a content serving node. Since the *CCNx-SDN controller* also performs these kind of elevated forwarding decisions, it is rather comparable with at least the 2 Hop Bridged case.

A.2 Measurements

Figure A.2 illustrates the measurement results of 15 iterations per each of the three scenarios. In the 1 Hop Bridged scenario, the transfer times reside between 5.77 and 6.34 seconds. Transfer times of our ICN-SDN approach though are spread less broad and vary by only ≈ 0.5 seconds in the range of 6 to 6.5 seconds. By introducing a second hop, the bridged scenario times increase significantly such that the quartile is around 8 and slightly above 8.5 seconds, which is a wider scattering than we have seen in both other cases. Also the maximum value is even above 9 seconds, resulting in a variation of over one second in our measurements. The times, of course, increased because of the additional 20 ms delay introduced by the two additional link traversals towards the node and back to the switch. It is quite plausible that by saving an additional CCNx hop,

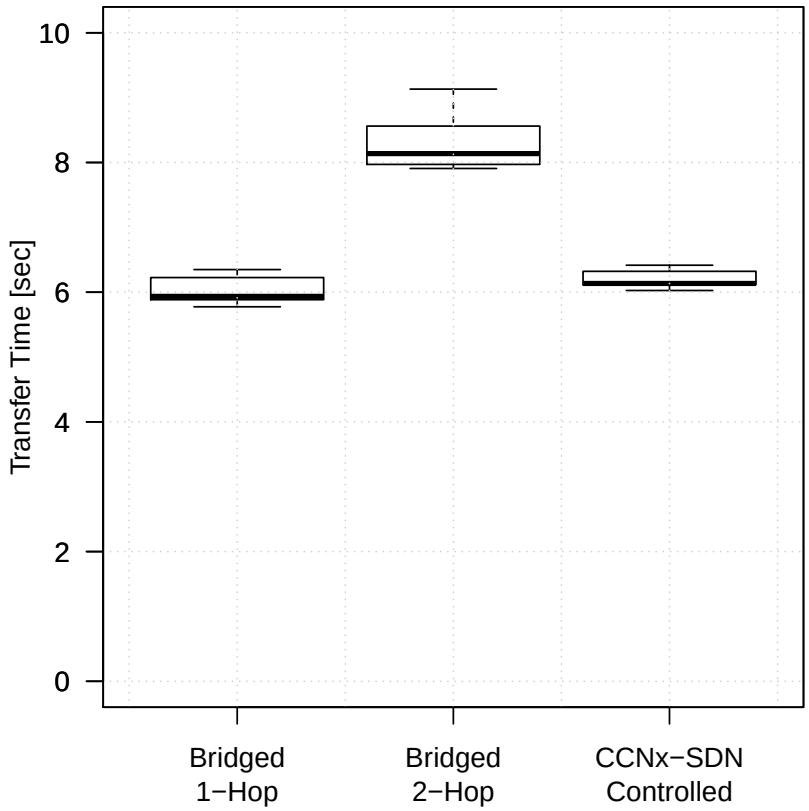


Figure A.2: 5 MiB content transfer time comparison

the delay can be reduced, but further on, one can also see that the variance in transfer times is lower in the ICN-SDN approach. Overall the results yield that the transfer times increase slightly, which is evident due to the detour Interest packets take through the controller, but they are even lower than they get when introducing an additional CCNx hop.

Our results show that the approach introduced in Section 4 is applicable to CCNx. Due to the detour introduced by handing up the interests to the controller, it is slightly slower than the direct communication between content requester and origin. Nevertheless, since we do not expect this 1 hop dissemination to be the default case, the SDN approach can improve the content transmission compared to the 2 Hop Bridged scenario. When even further enhanced, for instance through a centralized and cache aware request steering process in networks of greater complexity, we expect the effects to gain even more traction.

To check the measured values for plausibility, we add up the different delay components introduced in the network and compare them to the measured values.

$$\text{Delay}_{\text{(Host-to-SDN-Edge)}} = 10 \text{ ms} \tag{A.1}$$
$$\text{Delay}_{\text{(S1-to-S5)}} = 10 \text{ ms} \tag{A.2}$$
$$\text{Delay}_{\text{(Sx-to-C)}} = 5 \text{ ms} \tag{A.3}$$

$$\text{RTT}_{\text{(1Hop)}} = \text{Delay}_{\text{(S1-to-S5)}} + 2 * \text{Delay}_{\text{(Host-to-SDN-Edge)}} \tag{A.4}$$
$$= 10 \text{ ms} + 2 * 10 \text{ ms} = 30 \text{ ms} \tag{A.5}$$

$$\text{RTT}_{\text{(2Hop)}} = \text{RTT}_{\text{(1Hop)}} + 2 * \text{Delay}_{\text{(Host-to-SDN-Edge)}} \tag{A.6}$$
$$= 30 \text{ ms} + 2 * 10 \text{ ms} = 50 \text{ ms} \tag{A.7}$$

$$\text{RTT}_{\text{(SDN)}} = \text{RTT}_{\text{(1Hop)}} + 2 * \text{Delay}_{\text{(Sx-to-C)}} \tag{A.8}$$
$$= 30 \text{ ms} + 2 * 5 \text{ ms} = 40 \text{ ms} \tag{A.9}$$

$$\text{Filesize} = 5 \text{ MB} = 40 \text{ Mbit} \tag{A.10}$$
$$\text{\#-Chunks} \approx 5000 \tag{A.11}$$
$$\text{Link-BW} = 10 \text{ Mbit/s} \tag{A.12}$$
$$\text{Initial Transmission Window} = 80 \text{ Chunks} \tag{A.13}$$

$$\text{Transmission Time}_{\text{(Link)}} \ (\text{tt}_{\text{(Link)}}) = \frac{\text{Filesize}}{\text{Link-BW}} = \frac{40 \text{ Mbit}}{10 \text{ Mbit/s}} = 4 \text{ s} \tag{A.14}$$

$$\text{\#-Transmission Windows (\#-tw)} = \frac{\text{\#-Chunks}}{\text{Initial Transmission Window}} = \frac{5000}{80} = 62,5 \tag{A.15}$$

$$\text{Transmission Time}_{\text{(1H)}} \approx \text{\#-tw} * \text{RTT}_{\text{(1Hop)}} + \text{tt}_{\text{(Link)}} \tag{A.16}$$
$$\approx 62,5 * 30 \text{ ms} + 4 \text{ s} = 5.875 \text{ s} \tag{A.17}$$

$$\text{Transmission Time}_{\text{(2H)}} \approx \text{\#-tw} * \text{RTT}_{\text{(2Hop)}} + \text{tt}_{\text{(Link)}} \tag{A.18}$$
$$\approx 62,5 * 50 \text{ ms} + 4 \text{ s} = 7.125 \text{ s} \tag{A.19}$$

$$\text{Transmission Time}_{(\text{SDN})} \approx \#\text{-tw} * \text{RTT}_{(\text{SDN})} + \text{tt}_{(\text{Link})} \tag{A.20}$$

$$\approx 62,5 * 40 \text{ ms} + 4 \text{ s} = 6.5 \text{ s} \tag{A.21}$$

The above calculations show that the measured times are in line with the expected values according to this model. The SDN approach lies in between the 1 and the 2 hop cases and also the values are comparable – keeping in mind that they are estimates that for instance do not include processing times.

Since the CCNx included tool we use for the transmission, namely *ccncatchunks2*, issues multiple requests at a time to fully utilize network links, we additionally check the initial windows size, which we chose as 80 for out measurements.

$$\text{Min. Window size} = \frac{\text{Link-BW} * \text{RTT}}{\text{Chunksize}} \tag{A.22}$$

$$\text{Min. Window size}_{(\text{2Hop})} = \frac{10 \text{ Mbit/s} * 50 \text{ ms}}{1 \text{ kbyte}} = 62.5 \tag{A.23}$$

It is visible that the transmission window of 80 chunks is sufficient, Round Trip Time (RTT)-wise worst case of 2 Hop Bridging requires a transmission window of 63 chunks to fully utilize the link, leaving aside the processing delay.

Versicherung über Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §22(4) bzw.§24(4) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 2. Dezember 2013 Markus Vahlenkamp