TU Delft

Delft University of Technology

Resilience and Application Deployment in Software-Defined Networks

van Adrichem, Niels

DOI
10.4233/uuid:318d88af-e25e-4a7e-8d37-18770fe980c4

Publication date
2017

Document Version
Final published version

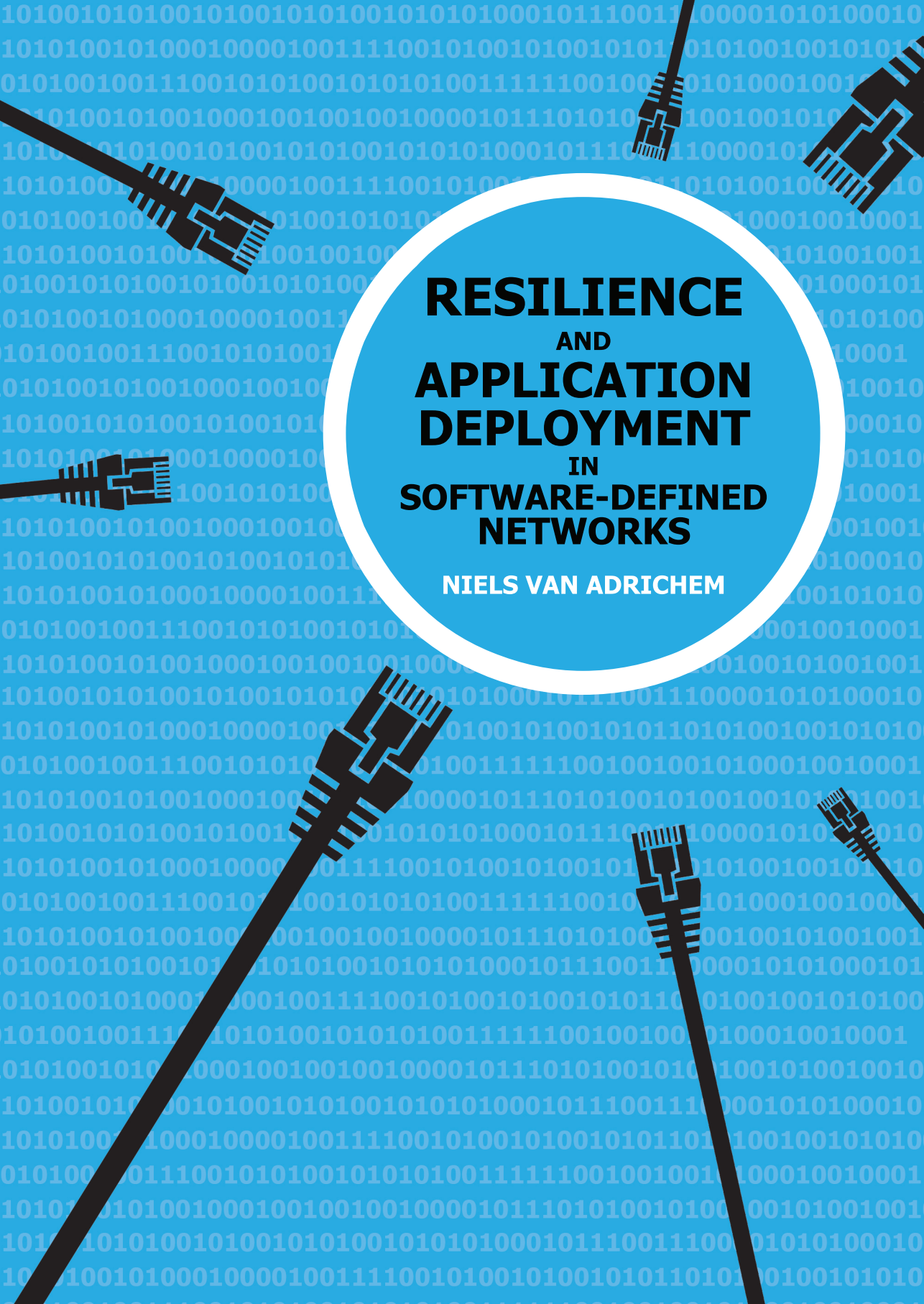
Citation (APA)
van Adrichem, N. (2017). *Resilience and Application Deployment in Software-Defined Networks*. [Dissertation (TU Delft), Delft University of Technology]. https://doi.org/10.4233/uuid:318d88af-e25e-4a7e-8d37-18770fe980c4

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

This work is downloaded from Delft University of Technology.
For technical reasons the number of authors shown on this cover page is limited to a maximum of 10.


RESILIENCE
AND
APPLICATION
DEPLOYMENT
IN
SOFTWARE-DEFINED
NETWORKS
NIELS VAN ADRICHEM

Resilience and Application Deployment in Software-Defined Networks

RESILIENCE AND APPLICATION DEPLOYMENT IN SOFTWARE-DEFINED NETWORKS

Proefschrift

ter verkrijging van de graad van doctor
aan de Technische Universiteit Delft,
op gezag van de Rector Magnificus prof. ir. K.C.A.M. Luyben,
voorzitter van het College voor Promoties,
in het openbaar te verdedigen op vrijdag 3 februari 2017 om 12:30 uur

door

Nicolaas Leonardus Maria VAN ADRICHEM

Master of Science in Computer Engineering,
Technische Universiteit Delft, Nederland,
geboren te Rotterdam, Nederland.

Dit proefschrift is goedgekeurd door de

promotor:	Prof. dr. ir. P. F. A. Van Mieghem
copromotor:	Dr. ir. F. A. Kuipers

Samenstelling promotiecommissie:

Rector Magnificus	voorzitter
Prof. dr. ir. P. F. A. Van Mieghem	Technische Universiteit Delft
Dr. ir. F. A. Kuipers	Technische Universiteit Delft
Onafhankelijke leden:	
Prof. dr. ir. N. H. G. Baken	Technische Universiteit Delft
Prof. O. Bonaventure	Université catholique de Louvain
Dr. P. Grosso	Universiteit van Amsterdam
Prof. dr. K. G. Langendoen	Technische Universiteit Delft

Title: Resilience and Application Deployment in Software-Defined Networks

Cover Design: Remy van Kats

Typeset by: The author using the LyX Document Processor

Printed by: Ipskamp Printing

Copyright © 2017 by N.L.M. van Adrichem

ISBN 978-94-6186-784-1

An electronic version of this dissertation is available at
http://repository.tudelft.nl/.

There is no place like 127.0.0.1

CONTENTS

Summary xi

Samenvatting xiii

1 Introduction 1
- 1.1 Dissertation Aim and Outline . . . 4

2 The Internet 7
- 2.1 Introduction . . . 8
- 2.2 Standardization procedure . . . 8
- 2.3 OSI and TCP/IP reference models . . . 9
- 2.4 Ethernet . . . 10
- 2.5 The Internet Protocol . . . 11
  - 2.5.1 From IPv4 to IPv6 . . . 14
- 2.6 Transmission Control and User Datagram Protocol . . . 14
- 2.7 Routing algorithms . . . 16
- 2.8 Routing Protocols . . . 17
  - 2.8.1 Distance-vector routing protocols . . . 18
  - 2.8.2 Link-state routing protocols . . . 19
  - 2.8.3 Path-vector routing protocols . . . 20
- 2.9 Software-Defined Networking . . . 20
- 2.10 Conclusion . . . 21

3 OpenNetMon: Network Monitoring in SDNs 23
- 3.1 Introduction . . . 24
- 3.2 Monitoring . . . 25
  - 3.2.1 Active vs. passive methods . . . 25
  - 3.2.2 Application-layer and network-layer measurements . . . 25
  - 3.2.3 Current measurement deployments . . . 26
- 3.3 Background and Related Work . . . 27
- 3.4 OpenNetMon . . . 29
  - 3.4.1 Polling for throughput . . . 29
  - 3.4.2 Packet loss and delay . . . 30
- 3.5 Experimental Evaluation . . . 31
- 3.6 More lessons learned . . . 34
- 3.7 Conclusion . . . 37

4 Fast Network Topology Failure Recovery in SDNs **39**
4.1 Introduction . . . 40
4.2 Failure detection mechanisms . . . 40
4.2.1 Bidirectional Forwarding Detection . . . 41
4.2.2 Liveliness monitoring with OpenFlow . . . 42
4.3 Proposal . . . 42
4.4 Experimental Evaluation . . . 44
4.4.1 Testbed environments . . . 45
4.4.2 Recovery and measurement techniques . . . 45
4.4.3 Experiments . . . 46
4.4.4 Results and analysis . . . 46
4.5 Related Work . . . 50
4.6 Conclusion . . . 51

5 All-to-all Network Topology Failure Protection in SDNs **53**
5.1 Introduction . . . 54
5.2 Problem Statement . . . 55
5.3 Per-failure precomputation . . . 57
5.3.1 Link-failure disjoint paths . . . 58
5.3.2 Node-failure disjoint paths . . . 59
5.3.3 Hybrid approach . . . 61
5.3.4 Routing Table Optimization . . . 62
5.4 Evaluation . . . 63
5.5 Software Implementation . . . 68
5.6 Related Work . . . 70
5.7 Conclusion . . . 73

6 Globally-Accesible Names in Named Data Networking **75**
6.1 Introduction . . . 76
6.2 Named Data Networking . . . 77
6.3 Recursive Name Aggregation . . . 78
6.4 Naming and discovery . . . 79
6.4.1 Inter-Domain Routing . . . 79
6.4.2 Intra-Domain Routing . . . 79
6.4.3 Local Area Network . . . 80
6.5 Mapping . . . 80
6.5.1 Location independence and detecting broken paths . . . 82
6.5.2 CCNx Renaming Implementation . . . 82
6.5.3 ContentObject Signature Validation . . . 84
6.5.4 CCNx DNS implementation . . . 84
6.5.5 Experimental Measurements . . . 84
6.6 Local Area Network Implementation . . . 85
6.6.1 Propagation . . . 86
6.7 Related Work . . . 87
6.8 Conclusion . . . 87

7 NDNFlow: Software-Defined Information-Centric Networking 89
7.1 Introduction . . . 90
7.2 Related Work . . . 90
7.3 NDNFlow . . . 91
7.4 Implementation . . . 93
7.4.1 OpenFlow controller implementation . . . 93
7.4.2 CCNx daemon implementation . . . 94
7.4.3 Protocol Implementation . . . 94
7.5 Experimental Evaluation . . . 95
7.5.1 Testbed environment . . . 95
7.5.2 Experiments and results . . . 96
7.6 Conclusion . . . 98

8 DNSSEC Misconfigurations 99
8.1 Introduction . . . 100
8.1.1 Motivation and problem definition . . . 100
8.1.2 Outline . . . 101
8.2 DNS and DNSSEC . . . 101
8.2.1 DNS . . . 102
8.2.2 DNSSEC . . . 103
8.2.3 Authentication of non-existing resources . . . 105
8.3 Related work . . . 107
8.4 Measurement tools . . . 107
8.5 Measurement Scenarios . . . 109
8.5.1 Top-down measurement scenario . . . 109
8.5.2 Bottom-up measurement scenario . . . 110
8.6 Results and evaluation: Top-down approach . . . 111
8.6.1 DNSSEC Implementation . . . 111
8.6.2 DNSSEC Misconfigurations . . . 112
8.6.3 Effects on availability . . . 115
8.7 Results and evaluation: Bottom-up approach . . . 116
8.7.1 DNSSEC Implementation . . . 116
8.7.2 DNSSEC Misconfigurations . . . 118
8.7.3 Analysis of domains without a DS . . . 118
8.7.4 Signature Validity . . . 119
8.8 Conclusion . . . 120

9 Conclusion 121
9.1 Future work . . . 124
9.2 Recommendations . . . 125

References 127

Acknowledgements 143

Biography 145

List of Publications 147

SUMMARY

In the past century, numerous iterations of automation have changed our society significantly. In that perspective, the professional and personal availability of computing devices interconnected through the Internet has changed the way we eat, live and treat each other. Today, the Internet is a service as crucial to our society as public access to electricity, gas and water supplies. Due to its successful adoption, the Internet now serves applications that were unthinkable at the time of its initial designs when social media, online global market places and video streaming were still far out of reasonable imaginary reach. Early research initiatives worked on realizing a global network of interconnected computers, an aim clearly realized by the successful implementation of the Internet and the fact that the infrastructure still suffices to provide connectivity to an unforeseen growth and change in usage. The research field of *future Internet* aims at long-term improvements of the Internet architecture, trying to improve the network infrastructure such that it will also facilitate future growth and applications.

In this dissertation, we have contributed to the field of future Internet by proposing, implementing and evaluating infrastructure improvements. Most of our work revolves around *Software-Defined Networking* (SDN), a network management architecture aiming at logical centralization and softwarization of network control through the separation of data plane and control plane functionality. In particular, we have assessed the feasibility and accuracy of network monitoring through SDN (see chapter 3), as well as contributed to the robustness and recovery of such networks under topology failure by speeding up failure detection and recovery (see chapter 4) and precomputation of network-wide per-failure protection paths (see chapter 5).

In addition to SDN, we have contributed to *Information-Centric Networking* (ICN), a network architecture optimizing content distribution by implementing network-layer forwarding techniques and cache-placement strategies based on content identifiers. We have contributed to this field by introducing a globally-accessible namespace maintaining a feasible global-routing-table size through separation and translation of context-related and location-aggregated name components (see chapter 6). Considering the same demand for centralization and softwarization of network control found in SDN applies to other network architectures, we have designed a protocol-agnostic SDN scheme enabling fine-grained control of application-specific forwarding schemes. With our prototype, we evaluate an implementation of such an SDN-controlled ICN, demonstrating correct functionality in both partial and fully upgraded networks (see chapter 7).

Besides working on future Internet topics, we have also taken a step aside and looked at more recent Internet architecture improvements. Specifically, we have performed measurements on the *Domain Name System*'s *Security Extensions* (DNSSEC). From these measurements we provide insight into the level of implementation and correctness of DNSSEC configuration. Through categorization of errors we explain their main causes and find the common denominators in misconfiguration (see chapter 8).

SAMENVATTING

In de afgelopen eeuw hebben verscheidene slagen van automatisering onze maatschappij significant veranderd. In het verlengde daarvan heeft de zakelijke en persoonlijke toegang tot computerapparaten die met elkaar verbonden zijn via het Internet onze manieren van eten, leven en omgang veranderd. Vandaag de dag is het Internet net zo cruciaal voor onze maatschappij als publieke toegang tot elektriciteits-, gas- en watervoorzieningen. Dankzij de succesvolle integratie van het Internet bedient het nu toepassingen die onvoorstelbaar waren ten tijden van de ontwerpfase toen social media, online globale marktplaatsen en video streaming sites nog ver buiten het redelijke verbeeldingsvermogen vielen. De eerste onderzoeksprojecten werkten aan het realiseren van een globaal netwerk van onderling verbonden computers, welk doel met de succesvolle uitrol van het Internet en het feit dat deze infrastructuur nog steeds voldoet aan een onvoorziene groei en verandering in gebruikspatronen duidelijk behaald is. Het onderzoeksgebied *future Internet* richt zich op lange termijn verbeteringen van de Internet architectuur en tracht hiermee de netwerk infrastructuur dusdanig te verbeteren dat het ook toekomstige groei en veranderingen kan ondersteunen.

In dit proefschrift hebben we bijgedragen aan dit veld door het voorstellen, implementeren en evalueren van infrastructurele verbeteringen. Het meeste van ons werk draait om *Software-Defined Networking* (SDN). Deze network management architectuur doelt op centralisatie en softwarisatie van netwerk controle door middel van een scheiding tussen de data en controle functies. In het bijzonder hebben we de haalbaarheid en nauwkeurigheid van netwerk monitoring door middel van SDNs beoordeeld (zie hoofdstuk 3), bijgedragen aan de robuustheid en herstelvermogen van dergelijke netwerken wanneer zich topologische fouten voordoen door het versnellen van foutdetectie en omleiding (zie hoofdstuk 4) en het voorberekenen van netwerk-brede fout-omleidingen (zie hoofdstuk 5).

Aanvullend op SDN hebben we bijgedragen aan *Information-Centric Networking* (ICN). Deze netwerk architectuur optimaliseert content distributie door het implementeren van netwerklaag forwarding technieken en caching strategiën op basis van content identificatoren. We hebben hier aan bijgedragen met het introduceren van een globaal toegankelijke naamruimte die een aanvaardbare globale forwarding tabel grootte onderhoudt middels scheiding en vertaling van context-gerelateerde en locatie-geaggregeerde naamcomponenten (zie hoofdstuk 6). Aangezien eenzelfde vraag naar de centralisatie en softwarisatie van netwerk controle uit SDN toepasbaar is op andere toekomstige Internet architecturen, hebben we een protocol-onafhankelijk SDN schema ontworpen die fijnmazige controle over toepassing-specifieke forwarding schema's mogelijk maakt. In ons prototype evalueren we een implementatie van een SDN gecontroleerd ICN en tonen we functionaliteit in zowel gedeeltijk als volledig geupgrade netwerken (zie hoofdstuk 7).

Naast het werk op toekomstige Internet architecturen, hebben we ook een stapje opzij genomen en gekeken naar meer recente verbeteringen aan de Internet architectuur. Met name hebben we metingen uitgevoerd aan de *Domain Name System's Security Extensions*. Deze meetresultaten bieden inzicht in de mate van implementatie en juistheid van diens configuratie. Door categorisatie van fouten leggen we hun voornaamste oorzaken bloot en vinden we de gemeenschappelijke delers in misconfiguratie (zie hoofdstuk 8).

1

INTRODUCTION

October 29, 1969, *University of California, Los Angeles*, "lo", the first characters sent over the first link of the ARPANET, a computer network that we now call *the first Internet* [1]. In fact, had the system not crashed after the first two characters, the first message sent would have read "login". However trivial sending such a message correctly appears to us today, it took until 1994 before reliable consumer access through dial-up connection became available. Today, access to reliable broadband access is common in most developed countries.

Since its inception, the Internet has dramatically changed how we live, communicate and perform many types of transactions. Even in the background, invisible to our phone and computer screens, the Internet plays an important role and has become a critical infrastructure in society supporting almost any communication-assisted transaction thinkable. All types of database management and transactional services, ranging from airline bookings, road traffic monitoring, bank and stock market transactions, distribution warehouse management, distribution of radio from studio to antenna and even the planning software that delivery express services use to instruct the postal carriers where to deliver the next package. Anywhere where a computer-like device communicates with another computer-like device, the communication network is in one way or another supported by what we call the Internet.

Despite its popularity, the underlying network architecture facilitating the Internet has not changed conceptually. In the meantime, the high availability of such a high-speed network has led to an unforeseen high usage. Where the first protocol implementation could facilitate at most 65,536 hosts [2][1], an estimated 1.3 billion IP addresses showed signs of activity in 2012 [5] and reports predict over 3 billion active users in 2015 [6][7]. Moreover, we use the Internet in ways unforeseen by its original specifications and design. The Internet Protocol (IP) was designed as a host-to-host message passing

[1] In fact, this is already the circa-1977 design adopting 16-bit destination addresses. An earlier design dating back to 1970 showed 256 possible hosts through 8-bit destination addresses [3], while an early RFC from 1971 refers to upgrading the Interface Message Processer (IMP) from 3 to 4 hosts "caused a rather large internal change" [4].

system where users can send one-time messages from one host to another in an almost telegraph-style manner.

By now, the usage of the Internet has outgrown the original specifications of IP. Besides the exponential growth in users, the way the Internet is used has also changed significantly. While IP still serves host-to-host messages, we tend to use it in terms of connections, sharing files and watching or listening to content distributed from one source to many viewers and listeners instead. These functionalities have been implemented as layers on top of IP[2]. The higher the layer, the more difficult it becomes to optimize the computer network for the functions described in that layer. In case of the highest layers, this is even prevented through legislation in many countries, forbidding network providers to inspect data that might contain privacy sensitive information.

However, the information stored in the higher layers is necessary to analyze content flows, optimize their respective paths and reduce bandwidth consumption through caching. Current Content Distribution Networks perform a fair job by keeping requests close to the requesters through a number of techniques such as propagating anycast IP addresses and applying geographical filters on top of the Domain Name System. However, these services need a multitude of complicated systems to guarantee functionality, such as load balancers, intrusion detection systems, denial of service mitigation systems and internal content replication services and respective policies. Implying high investments acquiring and maintaining these systems, content distribution networks are costly to deploy and hence are only available to large publishers or service subscribers.

Instead, the field of Information-Centric Networking (ICN) proposes an architectural change to the Internet architecture in which the use of pseudo-connections is replaced with a network layer that requests content directly, hence allowing the network to be optimized for content distribution instead. ICNs allow providers to optimize content streams and place network caches at any point in their network and allow any publisher from consumer to enterprise to benefit from these in-network caches. Most ICN implementations, such as Content-Centric Networking [9], Named Data Networking [10], the Publish/Subscribe Paradigm [11] and NetInf [12], however, are not yet in a production-ready state. Among others, problems exist regarding global naming and addressing conventions, growing routing table complexity and the transition procedure from the Internet Protocol to another.

Besides a large increase in the number of users and how users perceive and use the network, the administration and management has also become much more complicated. At its high tide, ARPANET hosted a little over 200 computers divided over 57 routers, a number that is easy to oversee and manage. Even then, a contemporary routing protocol was in place to discover network topology and compute shortest paths between hosts, thriving for optimal usage of the network.

Now that the Internet has grown and continues to expand, a variety of distributed routing protocols are in place finding appropriate paths to deliver connectivity to users. Some network providers' networks have grown so large, that they are employing different administrative domains within their networks to be able to handle the complexity. Still, they struggle to achieve a good overview of network usage and perform flexible planning of network resources. As such, network design and realization remains a very static pro-

[2] As described by the layered OSI model [8] described in chapter 2.

cess, whereas providers wish to introduce more control and flexibility to cost-effectively exploit their networks.

Traditionally, computer networking devices such as switches and routers both functionally handle network traffic and distributively derive a common network configuration. The field of Software-Defined Networking (SDN) proposes a system where the distributed nature of forwarding and routing protocols is split into two strictly distinct data and control planes. In these planes, switches revert to operating the data plane, while centralized network controllers operate the control plane. As such, switches are restricted to plainly forwarding packets, while controllers monitor data plane activities, compute correct network paths, configure switches accordingly and reconfigure them when necessary. The centralized network gives operators the opportunity to perform network configuration from a centralized perspective without the necessity to configure lots of distributed nodes whenever a small change in configuration needs to occur.

Computer networks are sensitive to failures, a property that also compromises the robustness of SDNs. Given that any device or link will statistically break at some point in time, protection from these failures is crucial. However, current routing protocols are "slow" in detecting failures and need time to converge to new functional paths between hosts. Faster failure detection protocols exist, such as BFD [13] and Ethernet OAM/CFM [14], but instead of offering network-wide coverage, those often operate on a host-to-host basis. Operating those from and to every node in the network implies a high configuration complexity and communication overhead and thus does not provide an architecturally scalable solution.

Finally, security and privacy concerns have recently been troubling the Internet. Although security- and privacy-improving solutions exist, those are often applied at the application level and thus only offer the implemented level of security and privacy for that particular application. Furthermore, the trustworthiness of security- and privacy-improving solutions in applications varies and is difficult to evaluate for end users. For optimal security and privacy protection the Internet needs to have mechanisms implemented from an architectural perspective, rather than leaving implementation to the application design process.

In general, we see a recurring pattern of problems that affect Internet architecture improvements. First of all, the current Internet architecture has had almost half a century to be improved to function as effectively as possible. This implies that although a new architecture may be fundamentally better, in practice it can achieve worse results since its execution has not been polished to be sped-up as intensively as the current architecture. Secondly, often unforeseen requirements complicate implementing proposed additions to the Internet architecture, a provider will not implement a new architecture if it cannot perform all, however custom-made or specific, of its current demands. Finally, once an Internet architecture is fully deployable, network operators tend to be conservative in learning and implementing new techniques in their production networks, hence delaying the Internet architecture's progress. The last is painfully proven by the implementation of IPv6 [15], the much-needed successor of the current de facto standard IPv4, which 17 years after its inception only has an 11 % adoption level [16][17].

1

1.1. DISSERTATION AIM AND OUTLINE

In this dissertation, we research, propose and evaluate network architectures optimizing the Internet to be capable to perform according to current-day and possible future specifications. To achieve this goal, we take promising existing candidate architectures, evaluate their current state, define areas that need improvement and improve accordingly to make selected architectures more suitable for actual deployment in computer networks. In particular, we look at the very recent emerging fields of Software-Defined Networking and Information-Centric Networking, contributing significant improvements to these fields. We furthermore evaluate already implemented Internet architecture improvements such as the Domain Name System Security Extensions. Figure 1.1 shows the relation between these topics and the chapters.

Chapter 2 presents a literature research summarizing the current Internet architecture and defines the primary concepts of Software-Defined Networking (SDN) and OpenFlow, a basic insight on which the remainder of this dissertation relies. One of the main advantages of large-scale SDN deployments is an increase in management flexibility through improved network monitoring tools and ultimately the possibility to deliver fine-grained on-demand Quality of Service to customers. As an important step towards that goal, chapter 3 proposes OpenNetMon, an approach and open-source software implementation monitoring online per-flow metrics such as throughput, delay and packet loss in SDNs through the OpenFlow protocol.

Chapter 4 introduces the field of network-topology robustness and proposes and by experiment evaluates our method to achieve protection against network-topology failures in SDNs. In chapter 5, we design an algorithm and network controller implementation deriving the corresponding network configuration and path computations.

Chapter 6 takes a step back from Software-Defined Networking and presents the field of Information-Centric Networking (ICNs), solving the problem of routing-table complexity in globally deployed ICNs while maintaining its location-agnostic nature through location mappings stored in the Domain Name System (DNS).

Chapter 7 combines the topics of ICN and SDN, showing that the transition procedure from a traditional communication network to an ICN can be significantly sped up and simplified through an SDN approach. By regarding the ICN as an application served by an already existing SDN, we implement an ICN through SDN techniques and show one can already achieve ICN's benefits without a fully upgraded network.

While the aforementioned chapters focused on improving on-going work, chapter 8 evaluates an already implemented Internet architecture improvement. The evaluation of an already available standard gives insight not only into the capability of Internet Service Providers to change and adapt to new standards, but also into the operational challenges that future architectural improvements should be able to cope with. Being a recently adopted Internet architecture improvement, we evaluate the Domain Name System Security Extensions (DNSSEC). DNSSEC offers a higher level of security through authentication of DNS, a critical part of the Internet infrastructure. We evaluate its level of (1) adoption, (2) correct implementation by its adopters and (3) show the consequences and their severeness when misconfigured.

Finally, chapter 9 concludes that it is inevitable for the Internet architecture to be redesigned and outlines our main findings, recommendations and future work.

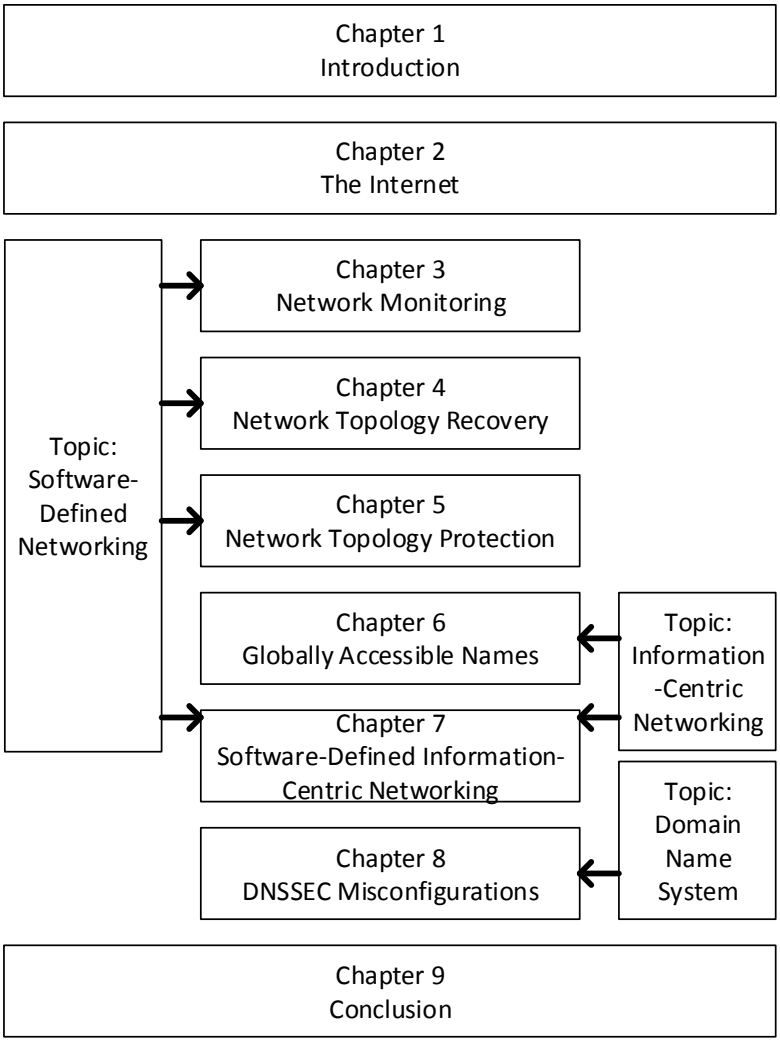


Figure 1.1: Dissertation outline, connection between main topics and core chapters, and overall connection between topics.

2

THE INTERNET

This dissertation discusses our work proposing and evaluating Internet architecture improvements. Most of our work focuses on current research trends in future Internet Architectures. To put this work into perspective, this chapter discusses the current state of the Internet architecture. More specifically, we (1) explain the organizational hierarchy and standardization procedure, (2) summarize the current state of the architecture according to the OSI Model and TCP/IP Reference model and its protocol implementations, and (3) the routing algorithms and protocols necessary to participate in the Internet. Additionally, we explain the primary concepts of Software-Defined Networking, a reoccurring topic in the rest of this dissertation. In chapter 8, we furthermore explain the Domain Name System, a system that, among others, resolves the domain name part of website URLs to their respective web server's IP addresses and is, hence, also a critical part of the Internet architecture.

2.1. INTRODUCTION

Since its inception in 1969, the Internet has undergone numerous standardization iterations. Standardization of its protocols provides compatibility between participating parties such as network operators and equipment manufacturers, while the process towards achieving new standards streamlines the adoption of new protocols or extensions. The history of protocols and candidates is large, especially dating to the ARPANET era, and even the current set of standards is too large to fully discuss within one book at any point of time. Nonetheless, excellent books have been written summarizing the history of ARPANET and the design of the Internet [18][19], overall explanation of concepts and protocols [20][21], as well as views on how it should have been designed [22]. Instead, in this chapter we focus on conceptually explaining the architectural design and protocols of the current state of the Internet, providing the necessary basis supporting the following chapters of this dissertation.

By definition, the Internet is a network of networks [23], in which independently administered networks are connected to form a larger interconnected network. As such there is little hierarchy, nor one specific ultimately responsible authority to rule this Inter-network. The only two organizations of central authority are the Internet Engineering Task Force (IETF) and the Corporation for Assigned Names and Numbers (ICANN) in conjunction with its department IANA (Interned Assigned Numbers Authority).

The IETF is a volunteer-run membership-based non-profit organization devoted to creating standards and evolving the Internet from a technical perspective (see section 2.2). The task of IANA involves allocation of unique Internet identifiers, such as IP addresses (see section 2.5), domain names (see chapter 8) and protocol assignment (see section 2.6), and is mainly an administrative task. ICANN, on the other hand, coordinates the maintenance and procedures involving the logistical infrastructure executing IANA's tasks.

First, the standardization process operated by the IETF is discussed in section 2.2. In section 2.3, we shortly explain the OSI model and its relation to the TCP/IP reference model used in the Internet, followed by an explanation of 3 of its implementation layers in sections 2.4, 2.5 and 2.6. Sections 2.7 and 2.8 respectively explain the algorithms and protocols necessary to find shortest paths and how to participate in routing discovery within a network. Finally, section 2.10 concludes this chapter.

2.2. STANDARDIZATION PROCEDURE

In essence, the development and promotion of Internet Standards is executed by the Internet Engineering Task Force (IETF). Within Working Groups volunteers, often financed through their employer, work on topics through open mailing lists and discussions at periodically organized IETF meetings to achieve rough consensus on new standards. Initially, a Working Group will work on an Internet Draft (I-D), a document presenting the work-in-progress working towards a proposal published for review in iterations of at most 6 months. As such, an I-D has no formal status yet. If and once mature enough, the Internet Engineering Steering Group (IESG) may be requested to approve publication of the I-D as a Request For Comments (RFC) document describing the proposed standard.

Although not all RFCs intend to describe standards, some are purely informational,

experimental or describe best current practices, the ones that are of the type Standards Track start as a Proposed Standard. Although errata occur occasionally, a Proposed Standard is considered sufficiently mature to allow global implementation on the Internet.

Finally, once implementation and widespread use evolve, the IESG promotes a Proposed Standard into an Internet Standard. At this point, the document is given a number in the STD series, while maintaining its number in the RFC series. Where an RFC document never changes, it may be both updated and obsoleted by follow-up RFC documents, which updates the referenced STD document once the updating or obsoleting document becomes an Internet Standard.

Although the review process is considered to be fairly informal and standardization generally occurs based on rough consensus, documentation of standards and processes is considerably punctual. For example, there are RFCs documenting the exact use of key words indicating requirement levels [24] and procedural processes regarding IETF itself [25].

However, not all widely used Proposed Standards make it into an Internet Standard. Besides the STD series, there is a Best Current Practice (BCP) series collecting documents that may be considered as official rules or stringent recommendations but are not standardized as such. For example, the just explained process of standardization is documented in BCP 9, currently containing, among others, RFC2026 [26] and RFC6410 [27]. Altogether, the derived and reviewed documents describe how parties and devices on the Internet must implement their protocols, hence maintaining interoperability. The Internet Architecture Board oversees the activities of the IETF as a whole.

In parallel, the Internet Research Task Force (IRTF) focuses on promoting longer-term Internet research and forms research groups to promote collaboration. The Software-Defined Networking Research Group (SDNRG) and Information-Centric Networking Research Group (ICNRG) in particular work on topics related to this dissertation and vice versa.

2.3. OSI AND TCP/IP REFERENCE MODELS

Communication functions are often modeled and characterized by to the Open Systems Interconnect (OSI) model [8], which divides the complexity of communication functions into 7 respective layers, represented in table 2.1a. In short, the different layers define the following functions:

1. Physical layer: Physical structure specification of transporting bits on a transmission medium, specifying among others the medium material, voltage, bitrate, bandwidth, etc.

2. Data Link layer: Specification of host-to-host data transfer on a physical transmission medium or local network.

3. Network layer: Routing of data across local networks.

4. Transport layer: Realizing reliability of end-to-end data transport.

5. Session layer: Representation of logical connections or sessions to the end points.

Table 2.1: The OSI Model and TCP/IP Reference Model.

(a) OSI Model.

Layer	Description
7	Application Layer
6	Presentation Layer
5	Session Layer
4	Transport Layer
3	Network Layer
2	Data Link Layer
1	Physical Layer

(b) TCP/IP Reference Model.

Layer	Description
7	Application Layer
6	
5	
4	Transport Layer
3	Internet Layer
2	Host-to-network
1	

6. Presentation layer: Solves (mis)representation of data between the two connected terminating operating systems, such as converting numbers stored as big-endian to little-endian or datetime conversion when necessary.

7. Application layer: Protocol definition of interaction between applications, an Internet browser and webserver, for example, communicate via the HTTP protocol, more and more often encrypted using SSL.

Each layer is responsible for a specific sets of functions, easing the process of designing the necessary protocols solving them. Furthermore, layering gives flexibility, in the sense that for example a Network layer implementation may function on multiple Data Link layer technologies without disturbing the implementation of the individual host-to-host networks. The OSI Model, however, mainly presents a theoretic model from which the more practical TCP/IP Reference model [28], shown in table 2.1b, and its implemented protocols slightly deviate. In practice, the TCP/IP Reference model combines the Physical and Data Link layers into one Host-to-network layer and does not describe the Session and Presentation layers since those functions are either solved in the Transport layer, the Application layer, or left out due to operating system compatibility.

The following sections explain how different protocols implement the particular layers of the TCP/IP Reference model. The Host-to-Network layer is implemented through Ethernet, explained in section 2.4. The Internet layer is implemented through the Internet Protocol, presented in section 2.5. The Transport Control Protocol and User Datagram Protocol are 2 common Transport layer protocols, summarized in section 2.6. The Application layer defines protocol specifications between servers and clients and contains a large range of application-specific protocols. The precise details of these specifications, however, are out of scope of this dissertation.

2.4. ETHERNET

The most often used technology to perform OSI layer 2, or host-to-host functionality from the TCP/IP reference model, is a set of technologies known as Ethernet [29]. Ethernet defines the protocol on a LAN that connecting computers must adhere to. Ethernet mainly performs the functions of medium access control necessary in a shared broad-

cast medium. It specifies the use of a 6-byte addressing scheme, where network interfaces are statically preconfigured with a universally unique identifier, so interfaces can distinguish dataframes intended for their use or not. Furthermore, it specifies the use of Carrier Sense Multiple Access with Collision Detection (CSMA/CD) to perform channel access control. CSMA/CD implies that hosts wait for the transmission channel to be clear before they send to decrease the probability of collissions. However, collisions still infrequently occur since propagation time prevents the detection of recently started competing transmissions, Ethernet detects such collisions and reattempts transmission after a short random waiting time to prevent the same collission from reoccurring. Each Ethernet frame furthermore contains a short frame check sequence (FCS) containing a short error-detecting code to determine the valid transmission of data frames.

Besides specifying Data link properties, Ethernet is subdivided in multiple subspecifications denoting the physical specification of OSI layer 1. Initially, Ethernet 10BASE5 and 10BASE2 specified 10 Mb/s bitrates using coaxial cables on which all computers on a LAN literally connect to the shared broadcast medium formed by the wire of at most 500 respectively 185 meters long. Later, 10BASE-T and its follow-ups 100BASE-TX (Fast Ethernet) and 1000BASE-T (Gigabit Ethernet) specified 10, 100 and 1000 Mb/s bitrates using twisted-pair cables, referred to as cat 5, cat 5e and cat 6 cables based on their specification. Besides the change in the carrying physical layer, the twisted-pair cables are connected between exactly 2 interfaces, and the broadcast domain is extended using so-called hubs and switches repeating incoming messages to the other connected computers. Where hubs broadcast incoming frames to all outgoing ports immediately, switches implement transmission queues implementing CSMA/CD and often have learning capabilities reducing the probability of collisions by only forwarding frames to the interface where it learned that a certain destination host resides.

Equal specifications exist for optical fiber connections and, more recently, the use of coaxial-like cables has been reintroduced in the form of shorter twinaxial cables used for switch-to-server connections in datacenter environments, allowing speeds up to 40 and 100 Gb/s. Furthermore, Ethernet specifies many application-specific extensions, for example for use in provider-to-provider networks and extensions to enable fast fault detection [14]. Additionally, a technique known as Virtual LAN [30] uses additional (VLAN) tags to isolate traffic from separated networks using a shared physical LAN.

2.5. THE INTERNET PROTOCOL

The Internet Protocol (IP) is, as suggested by its name, an unmissable protocol in the functionality of the Internet. To interconnect networks, IP describes a packet header and forwarding scheme to which all participating networking nodes adhere, and is therefore of utmost importance. Where Ethernet performed OSI layer 2 (host-to-host) functionality of communication within a broadcast medium or Local Area Network (LAN), IP offers OSI layer 3 or network-to-network functionality of forwarding packets across these (sub)networks. In IPv4, to date the most used version of IP, a global address space of 4 bytes decimally written in the form 0.0.0.0 to 255.255.255.255 is distributed in address spaces among these subnetworks. An institute may for example reserve an address space at its RIR and receive the address space 131.160.0.0 to 131.160.255.255, notated as 131.160.0.0/16 or 131.160.0.0 netmask 255.255.0.0. In the previous notations the bit-field

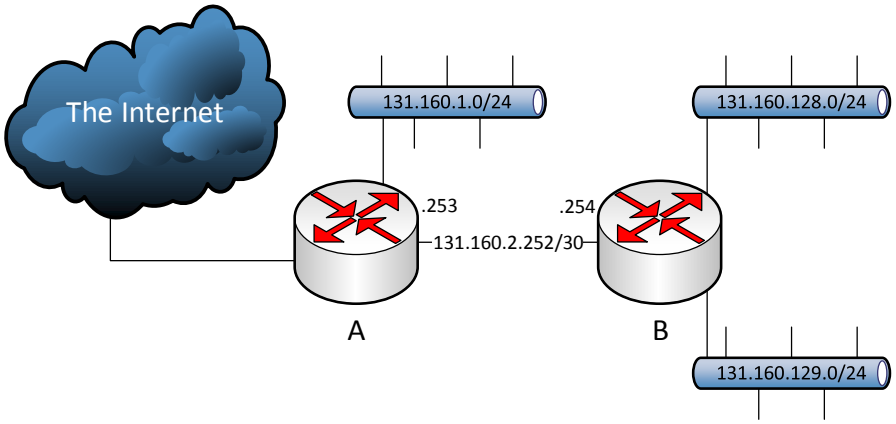


Figure 2.1: Example network topology in which router A is designated to serve the whole subnet 131.160.0.0/16, of which it has recursively designated the subnet 131.160.128/17 to router B. The link towards "the Internet" serves as a final gateway via which packets to other destinations can be transmitted.

/16 or netmask 255.255.0.0 indicate which bits of the aforementioned address space (often called a subnet or subnet address) identifies the specific subnetwork, the remaining bits identify elements such as clients or recursive subnets within that network. For example, the reserved address space may be used to put client machines into it directly, or may be divided into further subnets such as 131.160.0-255.0/24 when the network design needs more fine-grained subnets of 256 addresses each instead of one subnet of 65.536 addresses.

Within a subnet, devices may send packets to one another using their local subnet addresses, whose IP addresses are resolved to their respective MAC addresses using the Address Resolution Protocol (ARP) [31]. Packets that are intended for other subnets are sent to a local router called the gateway. Routers interconnect the different subnetworks, forming a globally connected network of subnetworks. Figure 2.1 depicts a simple example two-router network of such subnetworks. Each packet that is sent from one device to another contains an IP header denoting the source and destination IP address of that packet. Routers inspect the destination address of the packet and based on the match between the destination and its local forwarding table forward it to the appropriate next-hop router or send it to the device directly if connected to one of its local subnetworks. Table 2.2 shows the forwarding table of router A, table 2.3 shows the forwarding table of router B. The appropriate forwarding action for each packet is selected based on the longest-prefix matching IP address, meaning that from the lines with a matching network address, those with the longest number of bits is selected to execute the most precise available action. To execute longest-prefix matching, the forwarding table is sequentially ordered by netmask, metric and network address. The metric is an additional property available to set a preference between lines that match with the same netmask and network address. In practice this can be used to prefer wired networks over

Table 2.2: Forwarding table of router A from figure 2.1.

Destination	Netmask	Action
131.160.2.252	255.255.255.252	Local
131.160.1.0	255.255.255.0	Local
131.160.128.0	255.255.128.0	Via 131.160.2.254
131.160.0.0	255.255.0.0	Drop
0.0.0.0	0.0.0.0	Via ¨Internet¨

Table 2.3: Forwarding table of router B from figure 2.1.

Destination	Netmask	Action
131.160.2.252	255.255.255.252	Local
131.160.128.0	255.255.255.0	Local
131.160.129.0	255.255.255.0	Local
131.160.128.0	255.255.128.0	Drop
0.0.0.0	0.0.0.0	Via 131.160.2.253

WiFi networks on laptops, WiFi networks over 3G or 4G networks on mobile devices, or in general to switch from a primary forwarding rule to a secondary forwarding rule in case the primary rule ceases to be correct (for example due to removal or failure of its respective network interface). The contents of the forwarding table is sometimes manually configured, but often filled by a routing protocol (see section 2.8) that performs topology discovery and uses routing algorithms (see section 2.7) to compute a correct, mostly cost-optimized, configuration.

In principal, all addresses are globally routable and unique, with the exception of reserved address spaces solely intended for local or private use. For example, the range 169.254.0.0/16 is reserved for local subnet use (host-to-host) only and is not routable. Furthermore, the ranges 10.0.0.0/8, 172.16.0.0/12 and 192.168.0.0/16 are reserved ranges intended for private use, which are often distributed into multiple subnets to achieve routing within a private network but are not globally routable in the sense that one cannot directly send a packet to or from it from or to the publicly accessible network. Instead, a technique called Network Address Translation (NAT) [32] is often implemented in routers connecting private subnets to the public Internet to give machines on these networks outbound access.

NAT on-the-fly renames the respective outgoing packet's source address and transport protocol port number (see section 2.6) to a unique selected quadruple of one of the router's public IP addresses and transport protocol port numbers, maintains a list of active translations, and also renames returning packets' destination IP address and transport protocol port number to its original value, and forwards the resulting packet accordingly. Inbound access to a machine in a private subnet can be achieved by creating so called Port Forwarding entries where a public IP address and corresponding port number is statically mapped to a private IP address and port number on which the same renaming procedure is applied in reverse order.

2.5.1. FROM IPV4 TO IPV6

After 3 versions of protocols on the ARPANET, there initially was only one version of IP which is known as IPv4 [33]. However, due to the successful implementation of the Internet its 4-byte address specification soon became too small, and in 1998 a second version known as IPv6 containing 16-byte addresses was proposed [15][1]. Undue to great effort to migrate to IPv6, IPv4 currently remains the most used protocol of the two.

Besides the increase in address space, IPv4 and IPv6 are architecturally very similar. Other architectural differences from IPv4 to IPv6 include:

- Instead of the variable size IPv4 header that grew with the attachment of optional properties, IPv6 has a fixed-size header containing a minimal amount of properties, including source and destination addresses, to speed up header inspection and packet forwarding.
- Optional parameters are attached to IPv6 using Extension Headers, for example there exist specific extension headers to encode IPsec encrypted streams and to encapsulate fragmented parts of packets.
- The IPv6 header contains a flow label to enable per-flow matching without the need to perform Transport layer header decoding.
- Where IPv4 allowed per-hop fragmentation of packets that exceeded the Maximum Transmission Unit, resulting into a maximum packet size, IPv6 prohibits this behavior and obliges to send a message to the transmitting source to adapt the the packet size of this and future packets of the particular flow.
- IPv6 has better support for anycast routing.

A variety of transitioning protocols exist to exchange data between parties that independently support IPv6, but are disconnected from an IPv6 perspective in the sense that no IPv6-native path exists between the disconnected islands. Popular protocols include 6in4 [34], 6over4 [35], 6to4 [36] and TEREDO [37] that share the property that they encapsulate IPv6 data in IPv4 (or even UDP in IPv4) datagrams, creating IPv6 over IPv4 tunnels spanning IPv6 incapable parts of a path. To access IPv4-only destinations from an IPv6-only source, other protocols such asNAT64 [38] and DS-Lite [39] exist. The use of these protocols is known to introduce a higher delay and packet loss in long-term packet traces [40].

2.6. TRANSMISSION CONTROL AND USER DATAGRAM PROTOCOL

The Transmission Control Protocol (TCP) [41] implements the transport layer described in OSI layer 4. In practice, it delivers a reliable transport channel over the IP network that is prone to packet loss. To provide reliability, TCP divides a data stream into sequentially numbered segments whose arrival is acknowledged by its destination. TCP implements

[1] IPv6 is numbered 6 due to the reservation of header version number 5 for the Internet Stream Protocol, an experimental protocol that has not been introduced to public use.

a variety of retransmission strategies to recover from packet loss during data transmission, as well as a congestion control mechanism that will try to maximize the throughput of packets without inferring too much packet loss and unfairness towards competing data streams.

Besides delivering a reliable transport channel, TCP is most well known for offering so-called port numbers to aid in multiplexing multiple transport channels between nodes. Each data stream is set up towards a specific port number, the destination port, identifying the service one tries to connect to. Just like IP addresses and ASNs, the reservation of port numbers to applications is maintained by the IANA. Popular port numbers include port 80 for unencrypted HTTP access, port 443 for SSL-encrypted HTTP access, port 21 for FTP, port 22 for SSH, etc., etc. Each connection is set up from a source port, which number is often chosen randomly. Together, the quadruple of source and destination IP addresses and port numbers make up a unique identifier of a transport channel. Hence, by varying the source port a source can set up multiple data streams to the same application or service on a server. Just as with IP addresses, the returned data segments and acknowledgments exchange the source and destination port numbers to identify a returning segment belonging to the original data stream.

Additionally, a protocol of flags is used to set up, maintain and terminate transport channels. A transport channel is initiated by the source through sending a SYN flag, indicating the set up of a new connection resetting the range of sequence number to start with the attached random sequence number (logically 0). The destination replies with both a SYN and ACK flag, mutually initiating the session indicating its starting sequence number, as well as acknowledging the receipt of the previous segment. Finally, the initiator will send a segment with an ACK flag acknowledging receipt of the destination's previous segmenting, finishing the initialization phase.

Throughout the active phase of the transport channel the ACK flags together with acknowledgment numbers are used to indicate which sequence numbers have been received and are expected from the opposite party. Finally, after all data has been exchanged, the transport channel will be terminated through a sequence of FIN, FIN+ACK, ACK messages.

Although the described procedure is quite useful when larger amounts of data need to be exchanged or when transport channels need to exist over longer periods of time, the sequence of setting up and tearing down the transport channel provides too much overhead when small segments need to be transmitted incidentally. The User Datagram Protocol (UDP) [42] offers an alternative that only provides multiplexing using port numbers. The transmitting application itself needs to monitor whether it gets a reply to its sent frames and whether it needs to retransmit the whole query. UDP is often used in streaming scenarios, such as video or phone conferencing, where retransmission of packets is insufficient due to the retransmitted packet arriving too late to be usable. Instead, coding or prediction techniques (such as interpolation) are used to recreate feasible value for the missing packet. Additionally, the Realtime Transport Protocol (RTP) [43] is often used on top of UDP, defining a standardized format for streaming data and detecting loss of frames. Finally, projects like MultiPath TCP (MPTCP) [44][45] combine multiple subflows of TCP transport channels to achieve higher throughput in networks where multiple paths are available.

Algorithm 2.1 Shortest path relaxation

Relax(u, v, ℓ, d, π)

1: **if** $d_v > d_u + \ell_{uv}$
2: set d_v to $d_u + \ell_{uv}$
3: set π_v to u

2.7. ROUTING ALGORITHMS

Section 2.5 explained that he Internet Protocol (IP) forwards packets between networks based on forwarding tables stored in routers. To compute the correct configuration of these forwarding tables, routers implement routing protocols (see section 2.8) to perform topology discovery which execute routing algorithms, which depend on shortest-path finding algorithms to compute the shortest, or other means of cost-optimized, paths from and to each node in the network.

Where section 2.8 also discusses the implications of network size and routing internal and external to administrative boundaries, in this section we only use the representation of a network through a graph $G(N, L)$ of a set N of $|N|$ nodes and a set L of $|L|$ links interconnecting nodes in the network. Each link $l_{uv} \in L$ connects nodes u and v, and is characterized by a link weight ℓ_{uv}. From a programming perspective, the values l_{uv} and ℓ_{uv} can be stored in an adjacency lists or adjacency matrix to access their values. A shortest path implies that no other path from any node u to v has a sum of per-link weights lower than the already computed path[2].

Throughout most algorithms, shortest paths within a graph are found through searching for and executing the relaxation depicted in algorithm 2.1. The list d stores the found distance from a starting point to all other nodes $n \in N$ and is initialized to ∞ for all nodes besides the starting point which is set to 0. The list π stores the last node through which a node n can be reached and starts uninitialized, this value is used to retrieve the shortest path once execution finishes. The procedure of scanning for pairs where the relaxation requirement holds and then executes is first described by Ford in 1956 [46] as a linear programming solution, however, no specific scanning order or proof of finish was specified besides that "Eventually no such pairs can be found". Later, in 1958, Bellman showed that a solution for a network containing no negative cycles is guaranteed to be found after $|N| + 1$ iterations over all $|L|$ links using a similar description of the relaxation method [47]. In parallel, Moore published a similar technique specifically searching through electrical switching hardware to set up telephone traffic circuits [48], an approach close to the Routing Information Protocol presented in section 2.8. Both Schrijver [49] and Bertsekas [50] give thorough overviews on the shortest-path problem and its solutions, in the remainder of this section we discuss the most common solutions.

[2] Note that multiple shortest paths from any node A to B may exist, and randomly choosing one fulfills this criterion.

Label-correcting algorithms The method of performing $|N|+1$ iterations over all $|L|$ links of a network is generally known as the Bellman-Ford algorithm and is the first of a set of algorithms generally known as label-correcting algorithms. This set of algorithms are called label-correcting algorithms because the cost towards nodes during computation is not guaranteed to be final and is prone to correction in successive iterations of $|N|+1$. Often, this algorithm is implemented with heuristic approaches to decrease the amount of necessary link evaluations, improvements include the use of a queue in which nodes are placed when their distance to source may change [51], a similar technique using a two-queue algorithm [52], or reordering the scan sequence by topological sorting [53]. Although the previous heuristics generally improve execution time, their theoretical complexity remains $O(|N|.|L|)$. Finally, the Floyd-Warshall algorithm provides an all-to-all shortest-path finding solution [54][55].

Label-setting algorithms Opposed to the family of label-correcting algorithms, label-setting algorithms have the property that definite labels are already set during execution of the algorithm. It is clear which values are definitive, even if the algorithm has not finished completely. Shortly after the introduction of the label-correcting methods propose by Bellman, Ford and Moore, in 1959, Dijkstra published a, by now renowned, label-setting shortest-paths finding algorithm [56]. Instead of executing all possible relaxations iteratively, Dijkstra's algorithm effectively builds a shortest-paths tree from the source node whose distance d is set to 0, all other distances are set to ∞. At every iteration, the algorithm takes an unvisited node with minimal temporary distance indicating the value of this node is minimal and thus definite. All the chosen node's unvisited neighbors are labeled or relabeled using the relaxation formulation from algorithm 2.1. After this procedure, the node is considered visited and the procedure is repeated for the next unvisited node of minimal temporary distance. The procedure finishes when the destination node becomes a minimal-value node. A one-to-all shortest-paths tree can be found by continuing the procedure until all nodes have been visited. After its initial publication, a minimal priority queue [57] and later a Fibonacci heap [58] have been implemented to reduce the running time and computational complexity of the algorithm.

2.8. ROUTING PROTOCOLS

Section 2.7 presented routing algorithms to find shortest paths in networks represented by a graph stored as an adjacency list or matrix. This section discusses the concept of routing protocols, which through practical implementations on routers and software such as Zebra or Quagga [59] to perform topology discovery in networks, building aforementioned adjacency lists and matrices, compute their network's respective shortest path and hence derive the necessary configuration of routing tables on routers.

To distribute the complexity of topology discovery and path computation, the Internet is built up from multiple routing scopes. The most explicit differentiation between routing scopes is interdomain and intradomain routing. As discussed in the introduction of this section, the Internet is a network of networks, built up from interconnected networks. Each such interconnected network is called an Autonomous System (AS) and is given a unique AS Number (ASN), which reservations are - just like IP addresses, ad-

ministered by the IANA. An AS generally consists of one logical network, possibly subdivided in multiple subnets and further internal routing scopes, under the administration of one institute such as an Internet Service Provider (ISP), or any other type of company administering its own network.

Within these domains, the AS is self-sufficient to apply routing protocols of its choice, to the point that even static configuration is allowed. Historically, the Routing Information Protocol [60], a protocol of the family of distance-vector routing protocols, was used to perform intradomain routing, which type of routing protocols are discussed in subsection 2.8.1. Although still infrequently applied, most used intradomain routing protocols are of the family of link-state routing protocols, discussed in subsection 2.8.2. Finally, for an AS to interconnect with other ASes, and hence access the Internet, it must participate in interdomain routing. For this, the AS must propagate a summary of its routing information to its directly-linked peer ASes using the Border Gateway Protocol [61], a protocol of the family of path-vector routing protocols explained in subsection 2.8.3.

2.8.1. DISTANCE-VECTOR ROUTING PROTOCOLS

Distance-vector routing protocols, and the Routing Information Protocol (RIP) [60] in particular, compute the shortest paths between nodes in a very distributed manner. Periodically, all hosts broadcast their presence to all adjacent nodes, including the subnets they serve at a cost of 0. The adjacent nodes denote in their routing tables which subnets can be reached through which neighbors at the received cost incremented with 1. In all next cycles, all nodes include the "remote" entries in their routing table in their broadcast packet, extending the range through which nodes can be found with 1 step until the diameter of the network has been reached. Since every node increases the cost of incoming route advertisements, every routing entry contains the cost in the form of the number of hops or links towards the destination subnet. Hence, nodes will choose the shortest-path - in terms of hop count - when multiple paths towards a destination exist. Although RIP does not implement this precise feature, each hop-count may be replaced with routing metrics indicating the cost of a link, often representing the delay or inverse of the available bandwidth.

The previous description shows the distance-vector routing protocol family is both elegant and simple, though knows a number of drawbacks one should be aware of when implementing it. Convergence of changes may take a long time, at most the diameter multiplied with the duration of a period. Instead, one could implement the protocol in such a way that changes are propagated immediately.

When a subnet stops to be reachable from one node, for example due to a link or node failure, a bouncing effect may occur when propagating the change in available routes. When a node A is notified that it cannot reach subnets at node B anymore, it will propagate this to all its adjacent nodes among which node C. If C, however, in the mean time propagates to node A that it may still reach node B, A will assume a new route and shortest path to A via C. However, C still thinks it may access node B through A, hence a loop between A and C has occurred. Since the cost of a route is incremented at each node, the nodes will send each other increasingly expensive routing updates until another path becomes cheaper and convergence occurs. However, the behavior is un-

stable and it may take long before convergence occurs.

The problem of the bouncing effect becomes even larger when node A is not reachable at all anymore, for example due to a network split or a complete failure of node B. The bouncing effect now only converges when the maximum countable hop count, which is considered to be logically infinite, has been reached and is hence known as the count-to-infinity problem. Both the bouncing-effect and count-to-infinity problem can be solved in multiple fashions. A first solution is to implement a waiting time for changed next hops in routing, meaning that removed routes or decreased or increased routing costs via the same adjacent node are processed immediately, while changes towards another node are stalled until they appear stable. Furthermore, one could implement a so-called split horizon, where nodes exclude the routes via an adjacent node in the propagated information to that adjacent node. Both would have solved the posed example where node A and C bidirectionally loop traffic towards B, though problems may still rarely occur in more complex situations.

Finally, another problem of RIP, as a popular implementation of a distance-vector routing protocol is that it has only 4 bits reserved in its header specification to denote the hop count. Although this decreases the count-to-infinity problem to at most 16 iterations, this also maximizes the diameter of a network to 15 hops. The maximal storable value of $2^4 - 1$ represents a logically infinite value and thus unreachable destination, limiting the maximum size of the network. This problem was later solved in Cisco's proprietary Interior Gateway Routing Protocol (IGRP) [62].

2.8.2. LINK-STATE ROUTING PROTOCOLS

Instead of implementing distance-vector protocols, most intradomain networks use link-state routing protocols such as Open Shortest Path First (OSPF) [63] or Intermediate System to Intermediate System (IS-IS) [64]. Where nodes implementing a distance-vector routing protocol share their local routing tables and compute shortest paths in a distributed way, link-state routing protocols only propagate topological properties such as the existence of a link between two nodes, which are forwarded to flood to the whole scope of the network. Each node in the network scope collect and forward the propagated properties and, using the collected information, assembles an overview of the full network. The network state is stored in local adjacency matrices or lists, on which the individual nodes will run a routing algorithm (see section 2.7) to determine its local routing table configuration.

The big advantage of using link-state routing protocols is that those do not suffer from the problems introduced by distance-vector routing protocols. The only requirement is that the routing table entries configured by the individual nodes are compliant with those of the adjacent nodes to which packets will be forwarded. In shortest paths, compliance is achieved by the fact that every node will always forward packets to a node closer to their destination, hence loops nor paths that are not shortest will not occur in a stable environment. In general, propagation of changes will occur much faster since only the insertion, removal or update of a link is propagated without further computation while being flooded. Hence, after an update all nodes can locally compute their local configuration immediately, satisfying the updated topology quickly.

2.8.3. PATH-VECTOR ROUTING PROTOCOLS

Despite that link-state routing protocols solve most of the problems present in distance-vector routing protocols, they pose one major flaw, scalability. When a network grows extremely large, the memory required to store all topology information grows and computation time increases. Distance-vector routing protocols, on the other hand gives little to no information about the path taken when traffic is forwarded to an adjacent node. This problem is solved by the family of path-vector routing protocols, of which BGP [61], the routing protocol used in interdomain routing, is a member.

Path-vector routing protocols are an extension of distance-vector routing protocols, in the sense that path computation is executed by the propagation of routing advertisements. In addition to forwarding its routing table with increased cost, path-vector routing protocols add a list of nodes traversed to the destination, to which a node appends itself when forwarding a routing entry to its adjacent nodes. Hence, every routing entry contains a list of nodes to be traversed when a packet is forwarded to its next hop. In BGP, this list exists of the ASNs traversed. The list of nodes on each shortest path solves the looping problem introducing the bouncing effect and count-to-infinity in distance-vector routing protocol, a loop is detected by a node when its own node identifier is already in the node list, or another node identifier appears twice.

Furthermore, it gives a node the ability to choose between two next-hops not only based on hop count or the sum of routing metrics, but also on the nodes to be visited. As such, an ISP may decide to not use forwards traveling through ISPs that it deems unreliable and instead rely on other (possibly non-shortest) paths. In the case of political relations, one could avoid routes containing paths through politically sensitive areas.

2.9. SOFTWARE-DEFINED NETWORKING

Principally, routers and switches are devices configured to drop or forward packets from and to specific interfaces based on a configuration, a functional layer called the *data* plane. Traditionally, the decisional functions of a network determining whether and where packets may be routed towards, also known as the *control plane*, have been distributed along many (often all) participating forwarding nodes. The routing protocols described in subsection 2.8 require all participating routers to operate both the control and data plane functions. Besides complicating the design of said forwarding devices, it also complicates path computation. In order for a path computation scheme to work properly, all nodes must agree on the path to be taken for a subflow of packets. Hence, a change in computation policies or the introduction of more complicated path computation schemes is problematic. Furthermore, individual nodes do not have a fine-grained overview of all link, node and flow metrics, nor the computation capability to perform complex Quality of Service (QoS) computations on them if they would. Overall, it is very difficult to "just reroute" a specific subflow of packets from one path to another due to the fact that all nodes compute their forwarding rules individually based on a network-covering adjacency matrix.

Software-Defined Networking (SDN) solves the presented network management issue by decoupling the control plane from the forwarding plane and placing the control plane functions on a logically centralized network controller. Switches and routers concentrate at data plane functionality, forwarding packets according to a configuration de-

termined by their master network controller. The network controller operating the control plane functions connects to the switches, runs topology discovery and computes network-wide configuration parameters. The network controller instructs its connected switches' data plane to operate according to its derived configuration and monitors network health and decides to recompute and reroute when necessary. Functionality that would be complicated and suffering high delays in prior described distributed routing protocols due to the flooding and exchange of routing information messages throughout the network.

Although multiple (some even centralized) network monitoring and configuration protocols such as SNMP [65], NetConf [66], ForCes [67] and numerous other vendor-specific protocols exist, OpenFlow has become the unwritten de-facto interface protocol for SDNs [68]. Initially introduced to enable researchers to reprogram switching logic in production networks for experimental purposes, it evolved to an enterprise SDN solution with the wide availability of supported network controllers such as Floodlight [69], POX [70], NOX [71], OpenDayLight [72], Ryu [73], ONOS [74] and the adoption of OpenFlow by various switch manufacturers.

Besides a single-domain single-authority network, hypervisors such as FlowVisor [75] support administering shared multi-tennant network environments, guaranteeing isolation of flow data and competing configuration parameters. Multi-domain environments, however, are often stuck reproducing exterior routing protocols or negotiating the use of another multi-domain path computation protocol such as as the Path Computation Element Protocol [76]. Steadily, SDNs and OpenFlow are gaining ground in production networks, virtualized OpenFlow-capable switches such as Open vSwitch [77] are often found in datacenter networks. A practical large-scale implementation of an OpenFlow-based SDN is found in B4, Google's OpenFlow-based global SDN network [78]. In chapters 3, 4, 5 and 7 we will apply the OpenFlow protocol in various ways to contribute to the field of SDN.

2.10. CONCLUSION

In this chapter, we have summarized the most important aspects of the current Internet architecture, providing the necessary basic insight into the current state of the Internet to understand the following chapters. The most important messages from this chapter are the facts that the Internet (1) is not controlled by a single ultimately responsible authority, (2) derives new standards based on rough consensus, and (3) relies on correct understanding and implementation of written protocols to which all parties must adhere.

In particular, correct participation in the Internet relies on switching hardware executing both data plane operations (forwarding packets) and control plane operations (deciding where packets should be forwarded to) through routing protocols. A historical fact that we often challenge in the subsequent chapters.

3

OpenNetMon: Network Monitoring in SDNs

The primary concepts of Software-Defined Networking (SDN) and OpenFlow were introduced in section 2.9 of the previous chapter. In general, SDN and OpenFlow allow for better network control and flexibility in the pursuit of operating networks as efficiently as possible. Among others, OpenFlow provides interfaces to implement fine-grained Quality of Service (QoS) through the use of buffers and queues. However, it lacks the monitoring necessary to determine whether end-to-end QoS parameters are actually met and derive the parameters necessary to perform QoS path computations. In this chapter, we derive and evaluate monitoring techniques applicable to SDNs and present OpenNetMon, an approach and open-source software implementation to monitor per-flow metrics - especially throughput, delay and packet loss - in OpenFlow networks. While classic monitoring techniques often rely on packet sampling or only give interface statistics, OpenNetMon polls edge switches to retrieve per-flow counters and derives per-flow throughput and delay accordingly. OpenNetMon furthermore injects monitoring packets to derive experienced delay. Polling occurs at an adaptive rate that increases when flow rates differ between samples and decreases when flows stabilize, hence minimizing the number of queries. Finally, we verify throughput, delay and packet loss measurements for bursty scenarios by streaming variable bit-rate video streams in our experiment testbed.

This chapter is based on a published paper [79].

3.1. INTRODUCTION

Recently, Software-Defined Networking (SDN) has attracted the interest of both research and industry. As SDN offers interfaces to implement fine-grained network management, monitoring and control, it is considered a key element to implement QoS and network optimization algorithms. As such, SDN has received a lot of attention from an academic perspective, enabling researchers to perform experiments which were previously difficult or too expensive to perform. Additionally, industry is already adopting vendor-independent network management protocols such as OpenFlow to configure their networks.

A key requirement for network management in order to reach QoS agreements and traffic engineering is accurate traffic monitoring. In the past decade, network monitoring has been an active field of research, particularly because it is difficult to retrieve online and accurate measurements in IP networks due to the large number and volume of traffic flows and the complexity of deploying a measurement infrastructure [80]. Many flow-based measurement techniques consume too much resources (bandwidth, CPU) due to the fine-grained monitoring demands, while other monitoring solutions require large investments in hardware deployment and configuration management. Instead, Internet Service Providers (ISPs) over-provision their network capacity to meet QoS constraints [81][82]. Nonetheless, over-provisioning conflicts with operating a network as efficient as possible and does not facilitate fine-grained Traffic Engineering (TE). TE in turn, needs granular real-time monitoring information to compute the most efficient routing decisions.

Where recent SDN proposals - specifically OpenFlow [68] - introduce programming interfaces to enable controllers to execute fine-grained TE, no complete OpenFlow -based monitoring proposal has yet been implemented. We claim that the absence of an online and accurate monitoring system prevents the development of envisioned TE-capable OpenFlow controllers. Given the fact that OpenFlow presents interfaces that enable controllers to query for statistics and inject packets into the network, we have designed and implemented such a granular monitoring system capable of providing TE controllers with the online monitoring measurements they need. In this chapter, we present *OpenNetMon*, a POX OpenFlow controller module enabling accurate monitoring of per-flow throughput, packet loss and delay metrics. *OpenNetMon* is capable of monitoring online per-flow throughput, delay and packet loss in order to aid TE. *OpenNetMon* is published as open-source software at our GitHub repository [83].

The remainder of this chapter is structured as follows: In section 3.2, we first discuss existing measuring methods and monitoring techniques used by ISPs. Section 3.3 summarizes OpenFlow and its specific options that our implementation uses, as well as previous work in the field of measuring traffic in OpenFlow networks. Our proposal *OpenNetMon* is presented in section 3.4 and experimentally evaluated in section 3.5. Section 3.6 discusses implementation specific details regarding the design of our network controller components. Finally, section 3.7 concludes this chapter.

3.2. MONITORING

Traditionally, many different monitoring techniques are used in computer networks. The main type of measurement methods those techniques rely on and the trade-offs they bring are discussed in the following two subsections. Traditionally, every measurement technique requires a separate hardware installation or software configuration, making it a tedious and expensive task to implement. However, OpenFlow provides the interfaces necessary to implement most of the discussed methods without the need of customization. Additionally, subsection 3.2.3 summarizes several techniques ISPs use to monitor their networks.

3.2.1. ACTIVE VS. PASSIVE METHODS

Network measurement methods are roughly divided into two groups, passive and active methods. Passive measurement methods measure network traffic by observation, without injecting additional traffic in the form of probe packets. The advantage of passive measurements is that they do not generate additional network overhead, and thus do not influence network performance. Unfortunately, passive measurements rely on installing in-network traffic monitors, which is not feasible for all networks and requires large investments.

Active measurements on the other hand inject additional packets into the network, monitoring their behavior. For example, the popular application *ping* uses ICMP packets to reliably determine end-to-end connection status and compute a path's round-trip time.

Both active and passive measurement schemes are useful to monitor network traffic and to collect statistics. However, one needs to carefully decide which type of measurement to use. On the one hand, active measurements introduce additional network load affecting the network and therefore influence the accuracy of the measurements themselves. On the other hand, passive measurements require synchronization between observation beacons placed within the network, complicating the monitoring process.

3.2.2. APPLICATION-LAYER AND NETWORK-LAYER MEASUREMENTS

Often network measurements are performed on different OSI layers. Where measurements on the application layer are preferred to accurately measure application performance, ISPs often do not have access to end-user devices and therefore use network layer measurements. Network layer measurements use infrastructure components (such as routers and switches) to obtain statistics. This approach is not considered adequate, as the measurement granularity is often limited to port-based counters. It lacks the ability to differentiate between different applications and traffic flows. In our proposal in section 3.4 we use the fact that OpenFlow-enabled switches and routers keep per-flow statistics to determine edge-to-edge network performance, which is the closest a network provider may get without having its users implement monitoring tools at the application level.

3.2.3. CURRENT MEASUREMENT DEPLOYMENTS

The *Simple Network Management Protocol* (SNMP) [65] is one of the most used protocols to monitor network status. Among others, SNMP can be used to request per-interface port-counters and overall node statistics from a switch. Being developed in 1988, it is implemented in most network devices. Monitoring using SNMP is achieved by regularly polling the switch, though switch efficiency may degrade with frequent polling due to CPU overhead. Although vendors are free to implement their own SNMP counters, most switches are limited to counters that aggregate traffic for the whole switch and each of its interfaces, disabling insight into flow-level statistics necessary for fine-grained Traffic Engineering. Therefore, we do not consider SNMP to be suitable for flow-based monitoring.

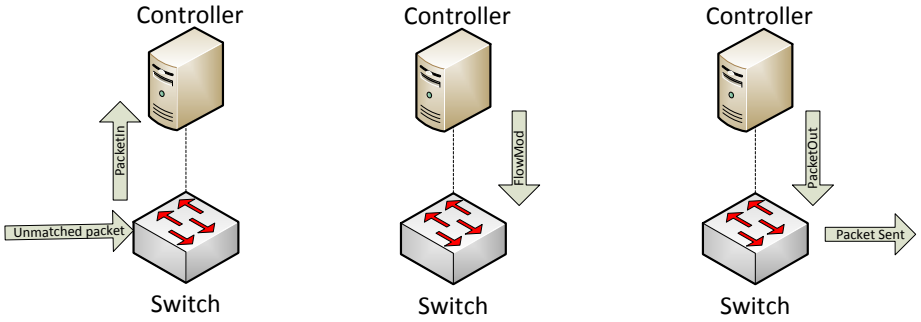
NetFlow [84] presents flow-based monitoring by collecting per-flow packet and byte counters, where a flow is defined by a preconfigured set of fields. Each switch or router independently captures and preprocesses these counters into flow exports which may be monitored through CLI or periodical exports to a central flow collector server. The main disadvantages of NetFlow are the fact that keeping statistics for a very fine-grained flow definition can be very costly, and that flow exports only occur when flows end, are idle for a considerate amount of time (in the order of seconds), or are very long-lived (order of minutes to hours).

To solve aforementioned issues, "sampled" NetFlow [84] collects samples of traffic and estimates overall flow statistics based on these samples, which is considered sufficiently accurate for long-term statistics. Sampled NetFlow uses a 1-out-of-n random sampling, meaning it stores every n-th packet, and assumes the collected packets to be representative for all traffic passing through the collector. Every configurable time interval, the router sends the collected flow statistics to a centralized unit for further aggregation. One of the major problems of packet-sampling is the fact that small flows are underrepresented, if noticed at all. Additionally, multiple monitoring nodes along a path may sample exactly the same packet and therewith over-represent a certain traffic group, decreasing accuracy. *cSamp* [85] solves these problems by using flow sampling instead of packet sampling and deploys hash-based coordination to prevent duplicate sampling of packets.

sFlow [86] provides an alternative protocol-agnostic packet sampling technique. Instead of sending preprocessed flow statistics implying switch CPU utilization impacting its scalability, sFlow sends records of the sampled data to its collector for further processing. The action of sampling and transmitting can be built into the switch's ASIC and is hence more scalable due to a lower load on the switch CPU.

Skitter [87], a CAIDA project that analyzed the Internet topology and performance using active probing, used geo-graphically distributed beacons to perform traceroutes at a large scale. Its probe packets contain timestamps to compute RTT and estimate delays between measurement beacons. Where Skitter is suitable to generate a rough estimate of overall network delay, it does not calculate per-flow delays, as not all paths are traversed unless a very high density of beacons is installed. Furthermore, this method introduces additional inaccuracy due to the addition and subtraction of previously existing uncertainty margins.

Measuring packet delay using passive measurements is a little bit more complex. *IP-*



(a) The first packet of a new connection arrives.

(b) The installation of forwarding rules.

(c) Retransmitting the captured packet.

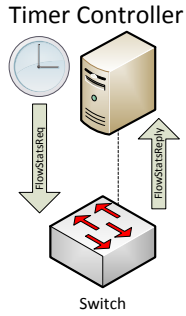
Figure 3.1: The three-step installation procedure of a new flow.

MON [88] presents a solution that captures the header of each TCP/IP packet, timestamps it and sends it to a central server for further analysis. Multiple monitoring units need to be installed to retrieve network-wide statistics. Where the technique is very accurate (in the order of microseconds), additional network overhead is generated due to the necessary communication with the central server. Furthermore, accuracy is dependent on accurate synchronization of the clocks of the monitoring units.

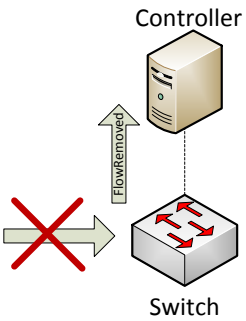
3.3. BACKGROUND AND RELATED WORK

Although SDN is not restricted to OpenFlow, other control plane decoupling mechanisms existed before OpenFlow, OpenFlow is often considered the standard communication protocol to configure and monitor switches in SDNs. OpenFlow-capable switches connect to a central controller, such as POX [70] or Floodlight [69]. The controller can both preconfigure the switch with forwarding rules, as well reactively respond to requests from switches which are sent when a packet matching none of the existing rules enters the network. Besides managing the forwarding plane, the OpenFlow protocol is also capable of requesting per-flow counter statistics and injecting packets into the network, a feature that we use in our proposal presented in section 3.4.

More specifically, OpenFlow-capable switches send a *PacketIn* message to the controller when a new, currently unmatched connection or packet arrives. The controller responds with installing a path using one or more Flow Table Modification messages (*FlowMod*) and instructs the switch to resend the packet using a *PacketOut* message. The *FlowMod* message indicates idle and hard timeout durations and whether the controller should be notified of such a removal with a *FlowRemoved* message. Figure 3.1 gives a schematic overview of the message exchange during flow setup. Using the *PacketIn* and *FlowRemoved* messages a controller can determine which active flows exist. Furthermore, the *FlowRemoved* message contains the duration, packet and byte count of the recently removed flow enabling the controller to keep statistics on past flows. Our pro-



(a) While a flow is active, the controller can - f.e. using a timer or other event - query the switch to retrieve flow specific statistics.



(b) The end of a flow is announced by sending a *FlowRemoved* packet to the controller.

Figure 3.2: While a flow is active, the controller and switch can exchange messages concerning the flow's state.

posal in section 3.4 uses this information in combination with periodically queried Flow Statistics Request (*StatsRequest*) messages, as shown in figure 3.2, to obtain information of running flows and regularly injects packets into the network to monitor end-to-end path delay.

OpenFlow's openness to switch and per-flow statistics has already been picked up by recent research proposals. OpenTM [89], for example, estimates a Traffic Matrix (TM) by keeping track of statistics for each flow. The application queries switches on regular intervals and stores statistics in order to derive the TM. The paper presents experiments on several polling algorithms and compares them for accuracy. Where polling solely all paths' last switches gives the most accurate results, other polling schemes, such as selecting a switch round robin, by the least load, or (non-) uniform random selection give only slightly less accurate results with at most 2.3 % deviation from the most accurate last-switch selection scheme. From the alternative polling schemes, the non-uniform random selection with a preference to switches in the end of the path behaves most accurate compared to last-switch polling, followed by the uniform random selection and round-robin selection of switches, while the least-loaded switch ends last still having an accuracy of approximately +0.4 Mbps over 86 Mbps. However, since OpenTM is limited to generating TMs for offline use and does not capture packet loss and delay, we consider it incomplete for online monitoring of flows.

OpenSAFE [90] focuses on distributing traffic to monitoring applications. It uses the fact that every new flow request passes through the network's OpenFlow controller. The controller forwards the creation of new flows to multiple traffic monitoring systems, which record the traffic and analyze it with an Intrusion Detection System (IDS). OpenSAFE, however, requires hardware investments to perform the actual monitoring, while we introduce a mechanism that reuses existing OpenFlow commands to retrieve the aforementioned metrics.

Others suggest to design a new protocol, parallel to OpenFlow, in order to achieve monitoring in SDNs. OpenSketch [91], for example, proposes such an SDN-based monitoring architecture. A new SDN protocol, however, requires an upgrade or replacement of all network nodes, a large investment ISPs will be reluctant to make. Furthermore, standardization of a new protocol has shown to be a long and tedious task. Since OpenFlow is already gaining popularity in datacenter environments and is increasingly being implemented in commodity switches, a solution using OpenFlow requires less investment from ISPs to implement and does not require standardization by a larger community. Therefore, we consider an OpenFlow-compatible monitoring solution, such as our solution OpenNetMon, more likely to succeed.

3.4. OPENNETMON

In this section, we present our monitoring solution *OpenNetMon*, written as a module for the OpenFlow controller POX [70]. *OpenNetMon* continuously monitors all flows between predefined link-destination pairs on throughput, packet loss and delay. We believe such a granular and real-time monitoring system to be essential for Traffic Engineering (TE) purposes.

In the following subsections, we will first discuss how our implementation monitors throughput and how we determine the right polling algorithm and frequency, followed by our implementation to measure packet loss and delay. Where one might argue that measuring throughput in OpenFlow SDNs is not new, albeit that we implement it specifically for monitoring instead of Traffic Matrix generation, we are the first to combine it with active per-flow measurements on packet loss and delay.

3.4.1. POLLING FOR THROUGHPUT

To determine throughput for each flow, *OpenNetMon* regularly queries switches to retrieve Flow Statistics using the messages described in section 3.3. With each query, our module receives the number of bytes sent and the duration of each flow, enabling it to calculate the effective throughput for each flow. Since each flow between any node pair may get different paths assigned by the controller, *OpenNetMon* polls on regular intervals for every distinct assigned path between every node pair that is designated to be monitored.

Even though polling each path's switch randomly or in round robin is considered most efficient and still sufficiently accurate [89], we poll each path's last switch. First, the round-robin switch selection becomes more complex in larger networks with multiple flows. When more flows exist, non-edge switches will be polled more frequently degrading efficiency. Furthermore, non-edge switches typically have a higher number of flows to maintain, making the query for flow statistics more expensive. Second, to compute the packet loss in subsection 3.4.2, we periodically query and compare the packet counters from the first and last switch of each path. As this query also returns the byte and duration counters necessary for throughput computation, we decided to combine these queries and solely sample each path's last switch for means of throughput computation.

3.4.2. Packet loss and delay

Per-flow packet loss can be estimated by polling each switch's port statistics, assuming a linear relation to link packet loss and the throughput rate of each flow. However, this linear relation to flow throughput does not hold when traffic gets queued based on QoS parameters or prioritization. Instead, we calculate per-flow packet loss by polling flow statistics from the first and last switch of each path. By subtracting the increase of the packet counter of the destination switch from the increase of the source switch packet counter, we obtain an accurate measurement[1] of the packet loss over the past sample.

Path delay, however, is more difficult to measure. Measuring delay in a non-evasive, passive manner - meaning that no additional packets are sent through the network - is infeasible in OpenFlow due to the fact that it is impossible to have switches tag samples of packets with timestamps, nor is it possible to let switches duplicate and send predictable samples of packets to the controller to have their inter-arrival times compared. Therefore, we use OpenFlow's capabilities to inject packets into the network. At every monitored path, we regularly inject packets at the first switch, such that that probe packet travels exactly the same path, and have the last switch send it back to the controller. The controller estimates the complete path delay by calculating the difference between the packet's departure and arrival times, subtracting the estimated latency from the switch-to-controller delays. The switch-to-controller delay is estimated by determining its RTT by injecting packets that are immediately returned to the controller, dividing the RTT by two to account for the bidirectionality of the answer giving $t_{delay} = \left(t_{arrival} - t_{sent} - \frac{1}{2}\left(RTT_{s1} + RTT_{s2}\right)\right)$.

The experiments on delay in section 3.5 show that using the control plane to inject and retrieve probe packets, using OpenFlow *PacketIn* and *PacketOut* messages, yields inaccurate results introduced by software scheduling in the switches' control planes. To ensure measurement accuracy, we additionally connect the controller to the switches' data planes through a separate VLAN. This VLAN is exclusively used to transport probe packets from the controller to all switch data planes directly. This method ensures that we omit the switches their control plane software, resulting in a higher accuracy.

To have the measurement accuracy and the packet overhead match the size of each flow, we inject packets for each path with a rate relative to the underlying sum of flow throughput. Meaning, the higher the number of packets per second of all flows from node A to B over a certain path C, the more packets we send to accurately determine packet loss. On average, we send one monitoring packet every measuring round. Although this gives overhead at first sight, the monitoring packet is an arbitrary small Ethernet frame of 72 bytes (minimum frame size including preamble) that is forwarded along the path based on a MAC address pair identifying its path and has a packet identifier as payload. Compared to a default MTU of 1500 (which is even larger in jumbo frames), resulting in frames of 1526 bytes without 802.1Q VLAN tagging, we believe that such a small overhead is a reasonable penalty for the gained knowledge.

[1] Given no fragmentation occurs within the scope of the OpenFlow network.

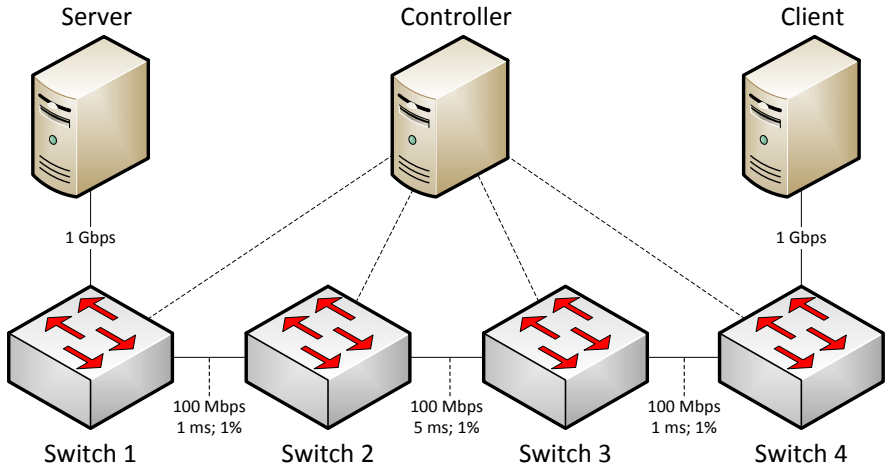


Figure 3.3: Experiment testbed topology. The measured traffic flows from Server to Client.

3.5. EXPERIMENTAL EVALUATION

In this section we evaluate our implementation of OpenNetMon by experiments on a physical testbed. Our testbed consists of two Intel Xeon Quad Core servers running stock Ubuntu Server 12.04.2 LTS with 1 Gbps NICs connected to four Intel Xeon Quad Core servers running stock Ubuntu Server 13.04 functioning as OpenFlow-compatible switches using Open vSwitch. The network is controlled by an identical server running the POX OpenFlow controller as shown in figure 3.3. All hosts are connected to their switch using 1 Gbps Ethernet connections, thus we assume plenty of bandwidth locally. Inter-switch connections, however, are limited to 100 Mbps. The delay between switches 1-2 and 3-4 equals 1 ms, while the delay between switches 2-3 equals 5 ms to emulate a WAN connection. Furthermore, the packet loss between all switches equals 1 %, resulting in an average packet loss a little less than 3 %. Delay and packet loss is introduced using NetEm [92]. Using this topology we intend to imitate a small private WAN, controlled by a single OpenFlow controller.

Throughout, we use a video stream to model traffic. Due to its bursty nature of traffic, we have chosen a H.264 encoded movie that is streamed from server to client. Figure 3.4 shows the throughput between our server and client measured by our implementation of OpenNetMon compared to Tcpstat. Furthermore, figure 3.5 shows packet loss compared to the configured packet loss. Finally, figure 3.6 presents the delay measured in our network.

The measurements shown in figure 3.4 represent the throughput measurements performed by Tcpstat and OpenNetMon, on average they only differ with 16 KB/s (1.2 %), which shows that most of the transmitted traffic is taken into account by the measurements. The average absolute difference, however, showed to be 17.8 % which appears

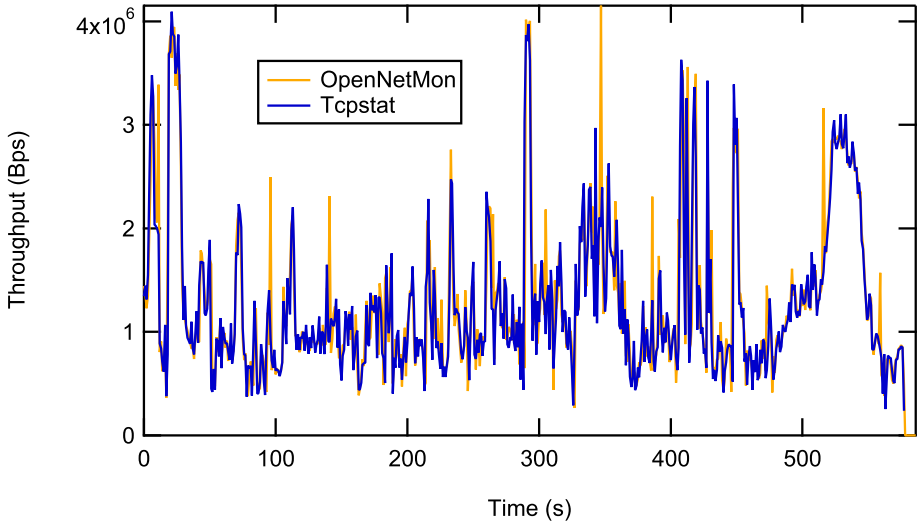


Figure 3.4: Bandwidth measurements of the flow between the client and server hosts, performed by both the OpenNetMon monitoring module and Tcpstat on the receiving node.

to be quite a significant inaccuracy at first sight. This inaccuracy is mainly introduced by a lack of synchronization between the two measurement setups. Due to the fact that we were unable to synchronize the start of the minimal 1-second buckets, traffic that is categorized in one bucket in one measurement is categorized in two adjacent buckets in the other. In combination with the highly bursty nature of our traffic, this leads to the elevated deviation. However, the high accuracy of the average shows an appropriate level of preciseness from OpenNetMon's measurements. In fact, we selected highly deviating traffic to prove our implementation in a worst-case measurement scenario, therefore, we claim our results are more reliable for a scenario with traffic of less bursty nature.

The throughput measurements in figure 3.4 furthermore show incidental spikes, followed or preceded by sudden drops. The spikes are introduced due to the fact that the switches' flow counter update frequency and OpenNetMon's polling frequency match too closely, due to which binning problems occur. In short, it occurs that our system requests the counter statistics shortly before the counter has been updated in one round, while it is already updated in the adjacent round. The difference is evened out in the average on the long run, as the misread value is equally decreased or increased in the adjacent bin. However, both bins have values that are equally but oppositely deviating from the expected value, contributing to the high standard deviation.

The described binning problem cannot be solved by either decreasing or increasing the polling frequency, in the best case the error margin is smaller but still existent. Instead, both ends need to implement update and polling frequencies based on the system clock, opposed to using the popular *sleep* function which introduces a slight drift due to delay introduced by the operating system scheduler and the polling and updating process consuming time to execute. Using the system clock to time update and polling en-

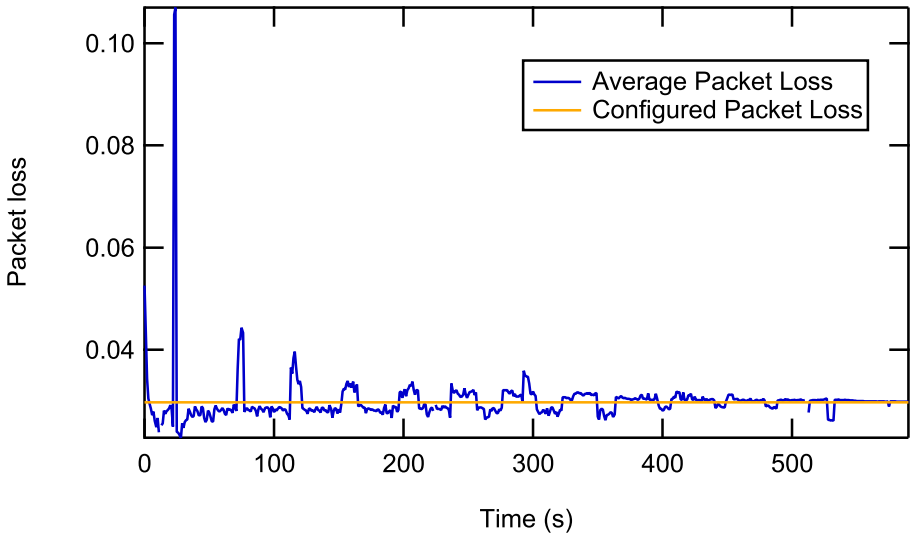


Figure 3.5: Packet loss measurements of the flow between the client and server hosts performed by the OpenNetMon monitoring module, compared to the configured values using NetEm.

sures synchronization between the two systems' sampling bins. Furthermore, the switch needs to implement a system to mutually exclude[2] access to the counter, guaranteeing a flow counter cannot be read until all its properties are updated and vice versa. Another, ideal, solution is to extend OpenFlow to allow flow counter updates to be sent to the controller at a configurable interval by subscription. However, since this requires updating both the OpenFlow specification and switch firmware, we do not consider it feasible within a short time frame.

As packet loss within one time sample may not represent overall packet loss behavior, figure 3.5 shows the running average of packet loss as calculated by computing the difference between the packet counters of the first and last switch on a path. Although the running packet loss is not very accurate, the measurements give a good enough estimation to detect service degration. For more accurate flow packet loss estimates one can reside to interpolation from port counter statistics.

Figure 3.6 shows delay measurements (1) as experienced by the end-user application to verify measurement results, computed by a stream's RTT, (2) measured by OpenNetMon using a separate VLAN connection to the data plane to send and retrieve probe packets and (3) measured by OpenNetMon using the control plane to send and retrieve probe packets. The figure shows that using the OpenFlow control plane to send and retrieve timing related probe packets introduces a large deviation in measurements, furthermore, the measured average is far below the expected value of 7 ms introduced by the addition of link delays as presented in figure 3.3. The measurements using exclusively data plane operations, however, resemble the delay experienced by the end-user

[2] Generally known as "mutex locks".

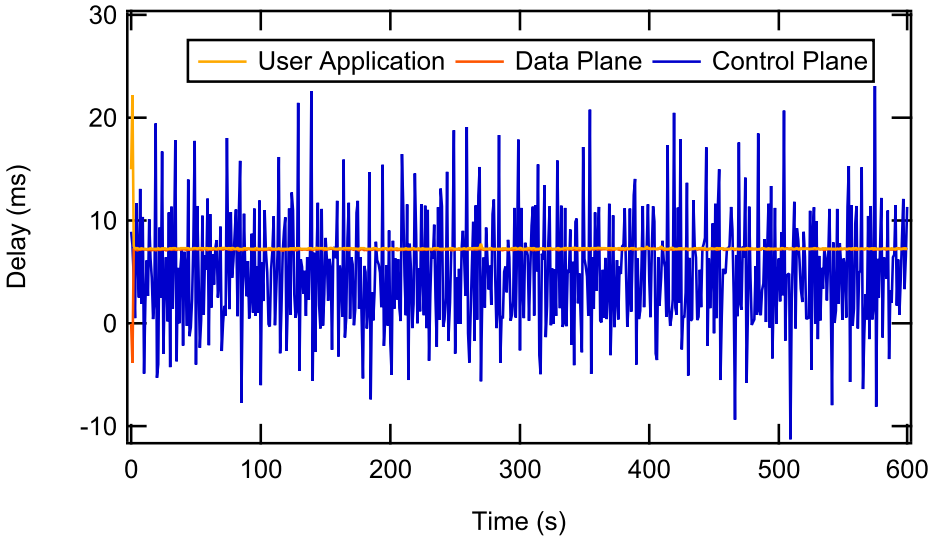


Figure 3.6: Delay measurements on the path from server to client, as measured by (a) the user application, (b) OpenNetMon connected to the data plane using a separate VLAN and (c) OpenNetMon using the OpenFlow control plane.

application so closely that a difference between the two is hardly identifiable.

These experiences are confirmed by the category plot in figure 3.7, showing an average of 4.91 ms with a 95 % confidence interval of 11.0 ms for the control plane based measurements. Where the average value already differs more than 30 % with the expected value, a confidence interval 1.5 times larger than the expected value is infeasible for practical use. The data-plane based measurements, however, do show an accurate estimation of 7.16 ± 0.104, which matches closely to the slightly larger end-user application experience of 7.31 ± 0.059 ms. The application delay is slightly larger due to the link delays from switch to end-hosts that cannot be monitored by OpenNetMon.

These results show that the control plane is *unsuitable* to use as a medium for time-accurate delay measurements, as response times introduced by the software fluctuate too much. However, we were able to obtain accurate results by connecting the controller to the data plane using a VLAN configured exclusively to forward probe packets from controller to the network. To prevent short-lived changes in delay from affecting network configuration, a low-pass filter can be implemented to smooth the results. An example of smoothing round-trip time measurements is implemented by TCP to compute the retransmission timer [93].

3.6. MORE LESSONS LEARNED

The implementation of OpenNetMon is published open source and can be found at our GitHub web page [83]. Our main intention to share it as open source is to enable other researchers and industry to perform experiments with it, use it as an approach to gain

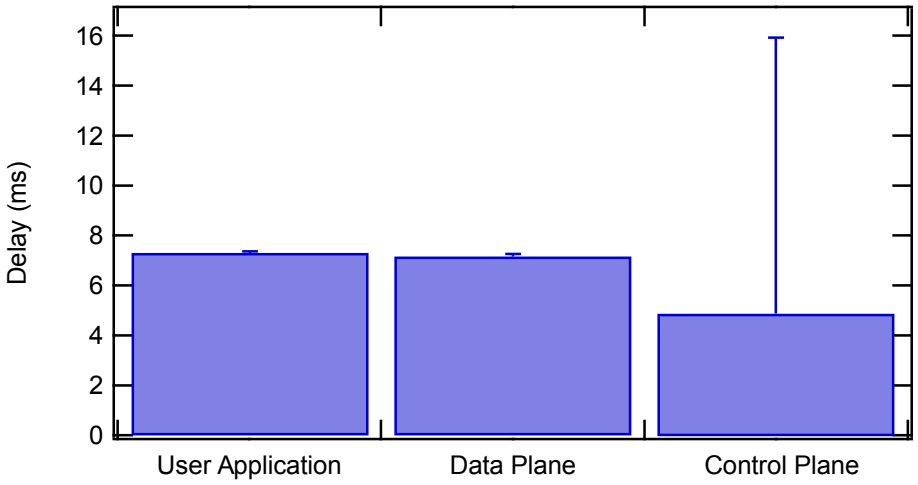


Figure 3.7: Category plot showing the averages and 95 % confidence intervals of the measurements from figure 3.6.

input parameters for fine-grained Traffic Engineering, and - when applicable - extend it to their use. While considered as a single module, technically OpenNetMon consists of two module components implemented in the POX OpenFlow controller. The *forwarding* component is responsible for the reservation and installation of paths, while the *monitoring* component is responsible for the actual monitoring. Both components rely on the POX Discovery module to learn network topology and updates.

Like some of the components shipped with POX, the forwarding component learns the location of nodes within the network and configures paths between those nodes by installing per-flow forwarding rules on the switches. However, we have implemented some of the specific details different from the other POX forwarding modules on which we will elaborate further. One could refer to this as a small guide to building one's own forwarding module.

1. OpenNetMon does not precalculate paths, it computes them online when they are needed. In a multipath environment (e.g. see [94]) not all flows from node A to B necessarily follow the same path, by means of load-balancing or Traffic Engineering it might be preferred to use multiple distinct paths between any two nodes. In order to support monitoring multipath networks, we decided to implement a forwarding module which may compute and choose from multiple paths from any node A to B. Especially to support online fine-grained Traffic Engineering, which may compute paths based on multiple metrics using the SAMCRA [95] routing algorithm, we decided to implement this using online path calculations.

2. We install per-flow forwarding rules on all necessary switches immediately. We found that the modules shipped with many controllers configured paths switch-

by-switch. Meaning that once an unmatched packet is received, the controller configures specific forwarding rules on that switch, resends that packet, and then receives an identical packet from the next switch on the path. This process iterates until all switches are configured. Our forwarding module, however, installs the appropriate forwarding rules on all switches along the path from node A to B, then resends the original packet from the last switch on the path to the destination instead.

3. We flood broadcast messages and unicast messages with an unknown destination on all edge ports of all switches immediately. We found that in the default modules shipped with POX, packets which were classified to be flooded, either due to their broadcast or multicast nature or due to the fact that their destination MAC address location was still unknown, were flooded switch-by-switch equally to the approach mentioned in the previous item. In the end, each switch in the spanning-tree contacts the controller with an identical packet while the action of that packet remains the same. Furthermore, if in the meantime the destination of a previously unknown unicast message was learned, this resulted in the forwarding module installing an invalid path from that specific switch to the destination switch. To reduce communication overhead when a packet arrives that needs to be flooded, our implementation contacts all switches and floods on all edge ports.

4. We only "learn" MAC addresses on edge ports to prevent learning invalid switch-port locations for hosts.

The forwarding component sends an event to the monitoring component when a new flow, with possibly a new distinct path, has been installed. Upon this action, the monitoring component will add the edge switches to the list that is periodically iterated by the adaptive timer. At each timer interval the monitoring component requests flow-counters from all flow destination and source switches. The flow-counters contain the packet counter, byte counter and duration of each flow. By storing statistics from the previous round, the delta of those counters is determined to calculate per-flow throughput and packet loss.

3.7. CONCLUSION

In this chapter, we have presented OpenNetMon, a POX OpenFlow controller module monitoring per-flow QoS metrics to enable fine-grained Traffic Engineering. By polling flow source and destination switches at an adaptive rate, we obtain accurate results while minimizing the network and switch CPU overhead. The per-flow throughput and packet loss is derived from the queried flow counters. Delay, on the contrary, is measured by injecting probe packets directly into switch data planes, traveling the same paths, meaning nodes, links and buffers, and thus determining a realistic end-to-end delay for each flow. We have published the implemented Python-code of our proposal as open source to enable further research and collaboration in the area of QoS in Software-Defined Networks.

We have performed experiments on a hardware testbed simulating a small inter-office network, while loading it with traffic of highly bursty nature. The experimental measurements verify the accuracy of the measured throughput and delay for monitoring, while the packet loss gives a good estimate of possible service degration.

Based on the work in [96], we further suggest to remove the overhead introduced by microflows, by categorizing them into one greater stream until recognized as an elephant flow. This prevents potential overloading of the controller by insignificant but possibly numerous flows. In future work, we intend to use OpenNetMon as an input generator for a responsive real-time QoS controller that recomputes and redistributes paths. OpenNetMon's measurements may further be used to perform multiple constraint routing, compute link-independent multiple paths based on actual usage allowing further exploitation of available bandwidth, or as a network health monitoring tool in general. Thanks to the open source publication of its implementation, OpenNetMon has already been used and extended by others to detect network failure and support recovery [97].

4

Fast Network Topology Failure Recovery in SDNs

Software-Defined Networking and its implementation OpenFlow facilitate managing networks and enable dynamic network configuration. Recovering from network failures in a timely manner, however, remains nontrivial. The process of (a) detecting a failure, (b) communicating it to the controller and (c) recomputing the new shortest paths results in an unacceptably long recovery time. In chapter 5 we derive methods to find and configure failure-disjoint paths prior to a failure, hence omitting path computation delay; in the present chapter we demonstrate that current solutions, employing reactive restoration or proactive protection, indeed suffer long delays. We introduce a failover scheme with per-link Bidirectional Forwarding Detection sessions and preconfigured primary and secondary paths computed by an OpenFlow controller. Our implementation reduces the recovery time by an order of magnitude compared to related work, which is confirmed by experimental evaluation in a variety of topologies. Furthermore, the recovery time is shown to be constant irrespective of path length and network size.

This chapter is based on a published paper [98].

4.1. INTRODUCTION

Recently, Software-Defined Networking (SDN) has received much attention, because it allows networking devices to exclusively focus on data plane functions and not control plane functionality. Instead, a central entity, often referred to as the controller, performs the control plane functionality, i.e. it monitors the network, computes forwarding rules and configures the networking nodes' data planes.

One of the benefits of the central controller introduced in SDN is its possibility to monitor the network for performance and functionality and reprogram when necessary. Where the controller can monitor overall network health as granular as observing per-flow characteristics, such as throughput, delay and packet loss [79], the most basic task is to maintain end-to-end connectivity between nodes. Hence, when a link breaks, the controller needs to reconfigure the network to restore or maintain end-to-end connectivity for all paths. However, the time-to-restoration of a broken path, besides the detection time, includes delay introduced by the propagation time of notifying the event to the controller, path re-computation, and reconfiguration of the network by the controller. As a result, controller-initiated path restoration may take over 100 ms to complete, which is considered too long for provider networks where at most 50 ms is tolerable [99].

In this chapter, we introduce a fast (sub 50 ms) failover scheme relying on link-failure detection and combining primary and backup paths, configured by a central OpenFlow [68] controller. We implement per-link failure detection in SDNs using Bidirectional Forwarding Detection (BFD) [13], a protocol that detects failures by detecting packet loss in frequent streams of control messages.

In section 4.2, we discuss several failure detection mechanisms and analyze how network properties and BFD session configuration influence restoration time. In section 4.3, we introduce and discuss the details of our proposed failover mechanism. Section 4.4 presents our experiments, after which the results are discussed and analyzed to verify our failover mechanism. Related work is discussed in section 4.5. Finally, section 4.6 concludes this chapter.

4.2. FAILURE DETECTION MECHANISMS

In this section, we introduce different failure detection mechanisms and discuss their suitability for fast recovery. In subsection 4.2.1 we analyze and minimize the elements that contribute to failure detection, while subsection 4.2.2 introduces OpenFlow's Fast Failover functionality.

Before a switch can initiate path recovery, a failure must be detected. Depending on the network interface, requirements for link failure detection are defined for each network layer. Current OpenFlow implementations are mostly based on Ethernet networks. However, Ethernet was not designed with high requirements on availability and failure detection. In Ethernet, the physical layer sends heartbeats with a period of 16 ± 8 ms over the link when no session is active. If the interface does not receive a response on the heartbeats within a set interval of $50 - 150$ ms, the link is presumed disconnected [100]. Therefore, Ethernet cannot meet the sub 50 ms requirement, which is confirmed by our experiments in section 4.4, hence failure detection must be performed by higher network protocols.

On the data-link layer multiple failure detection protocols exist, such as the Spanning Tree Protocol (STP) and Rapid STP [101], which are designed to maintain the distribution tree in the network by updating port status information. These protocols, however, can be classified as slow, as detection windows are in the order of seconds. Instead, in our proposal and implementation we will use BFD [13], which has proven to be capable of detecting failures within the required sub 50 ms detection window.

4.2.1. BIDIRECTIONAL FORWARDING DETECTION

The Bidirectional Forwarding Detection (BFD) protocol implements a *control* and *echo* message mechanism to monitor liveliness of links or paths between preconfigured end-points. Each node transmits control messages with the current state of the monitored link or path. A node receiving a control message, replies with an echo message containing its respective session status. A session is built up with a three-way handshake, after which frequent control messages confirm absence of a failure between the session end-points. The protocol is designed to be protocol agnostic, meaning that it can be used over any transport protocol to deliver or improve failure detection on that path. While it is technically possible to deploy BFD directly on Ethernet, MPLS or IP, Open vSwitch [77], a popular OpenFlow switch implementation also used in our experiments, implements BFD using a UDP/IP stream.

The failure detection time T_{det} of BFD depends on the transmit interval T_i and the detection time multiplier M, T_i defines the frequency of the control messages, while M defines the number of lost control packets before a session end-point is considered unreachable. Hence, the worst-case failure detection time equals $T_{det} = (M+1) \cdot T_i$. Typically, a multiplier of $M = 3$ is considered appropriate to prevent small packet loss from triggering false positives. The transmit interval T_i is lower-bounded by the round-trip-time (RTT) of the link or path. Furthermore, BFD intentionally introduces a 0 to 25% time jitter to prevent packet synchronization with other systems on the network.

The minimal BFD transmit interval is given in equations (4.1) and (4.2), where $T_{i,min}$ is the minimal required BFD transmit interval, T_{RTT} is the round-trip time, T_{Trans} is the transmission delay, T_{Prop} is the time required to travel a link in which we include delay introduced by routing table look-up, L is the number of links in the path and T_{Proc} is the processing time consumed by the BFD end-points.

$$T_{i,min} = 1.25 \cdot T_{RTT} \tag{4.1}$$

$$T_{i,min} = 1.25 \cdot 2 \cdot (T_{Trans} + L \cdot T_{Prop} + T_{Proc}) \tag{4.2}$$

It is difficult to optimize for session RTT by improving T_{Trans} and T_{Proc} as those values are configuration independent. However, by using link monitoring ($L = 1$), the interval time is minimized and smaller failure detection times are possible. A great improvement compared to per-path monitoring, where the number of links L is upper-bound by the diameter of the network.

An upper-bound for T_{RTT} can easily be determined with packet analysis, where we assume that the process of processing and forwarding is equal for each hop. On high link loads, the round-trip-time (RTT) can vary much and might cause BFD to produce false

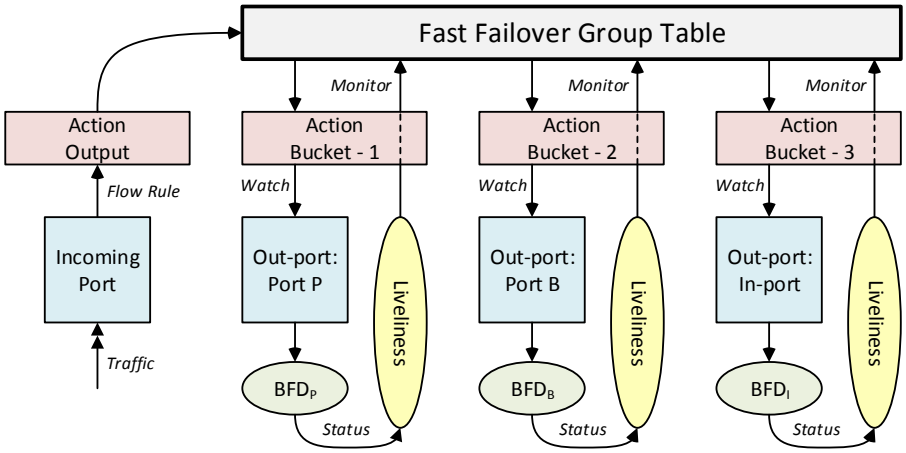


Figure 4.1: OpenFlow Fast Failover Group Table.

positives. During the development of TCP [41], a similar problem was identified in [102], where the re-transmission interval of lost packets is computed by $\beta \cdot T_{RTT}$. The constant β accounts for the variation of inter-arrival times, which we implement in equation (4.3).

$$T_{i,min} = 1.25 \cdot \beta \cdot T_{RTT} \tag{4.3}$$

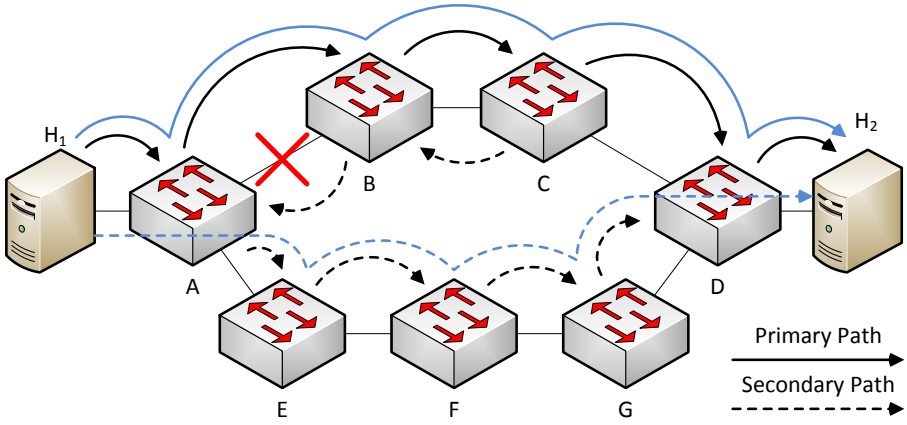
A fixed and conservative value $\beta = 2$ is recommended [103].

4.2.2. LIVELINESS MONITORING WITH OPENFLOW

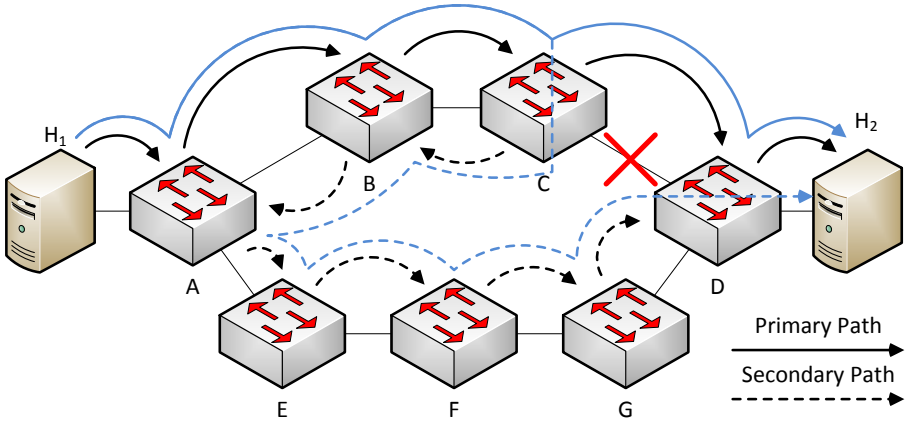
From OpenFlow protocol version 1.1 [104] onwards Group Table functionality is supported. Group Tables extend OpenFlow configuration rules allowing advanced forwarding and monitoring at switch level. In particular, the Fast Failover Group Table can be configured to monitor the status of ports and interfaces and to switch forwarding actions accordingly, independent of the controller. Open vSwitch implements the Fast Failover Group Table where the status of the ports is determined by the link status of the corresponding interfaces. Ideally, the status of BFD should be interpreted by the Group Table as suggested in figure 4.1 and implemented in section 4.4.

4.3. PROPOSAL

We propose to divide the recovery process into two steps. The first step consists of a fast switch-initiated recovery based on preconfigured forwarding rules guaranteeing end-to-end connectivity. The second step involves the controller calculating and configuring new optimal paths. Switches initiate their backup path after detecting a link failure, meaning that each switch receives a preconfigured backup path in terms of Fast Failover rules. Link loss is detected by configuring per-link - instead of per-path - BFD sessions.



(a) Topology with broken link resulting in usage of a secondary path.



(b) Topology with broken link resulting in usage of a backup path by crankback.

Figure 4.2: A topology showing its primary and secondary paths including two backup scenarios in case of specific link-failures. Where the first scenario (a) uses a disjoint backup path, the second scenario (b) relies on crankback routing.

Using per-link BFD sessions introduces several advantages:

1. A lower detection time due to a decreased session round-trip-time (RTT) and thus lower transmit interval.

2. Decreased message complexity and thus network overhead as the number of running BFD sessions is limited to 1 per link, instead of the multiplication of all end-to-end sessions and their intermediate links.

3. Removal of false positives. As each session spans a single link, false positives due to network congestion can be easily removed by prioritizing the small stream of control packets.

In situations where a switch has no feasible backup path, it will return packets to the previous switch by crankback routing. As the incoming-port is part of the packet-matching filter of OpenFlow, preceding switches have separate rules to process returned packets and forward them across their backup path. This implies a recursive returning of packets to preceding switches until they can be forwarded over a feasible link- or node-disjoint path. Figure 4.2 shows an example network with primary and backup paths, as well as two failover situations.

By instructing all switches up front with a failover scenario, switches can initiate a failover scenario independent of the SDN controller or connectivity polling techniques. The OpenFlow controller computes primary and secondary paths from every intermediate switch on the path to the destination to supply the necessary redundancy and preconfigures switches accordingly. Although the preconfigured backup-path may not be optimal at the time of activation, as in subfigure 4.2b, it is a functional path that is applied with low delay. Additionally, once the controller is informed of the malfunction, it can reconfigure the network to replace the current backup path by a more suitable path without traffic interruption as performed in [105].

The transmit interval of BFD is upper-bounded by the RTT between the session endpoints. As we are configuring per-link sessions, the transmit interval decreases greatly. For example, in our experimental testbeds we have a RTT below 0.5 ms, thus allowing a BFD transmit interval of 1 ms. Although this might appear as a large overhead, per-link sessions limit the number of traversing BFD sessions to 1 per link. In comparison, per-path sessions imply $O(|N| \times |N|)$ shortest paths to travel each link. Even though most of them may be forwarded to their endpoint without inspection, each node has to maintain $O(|N|)$ active sessions.

A BFD control packet consists of at most 24 bytes with authentication, encapsulation in a UDP, IPv4 and Ethernet datagram results in $24+8+20+38 = 90$ bytes $= 720$ bits. Sent once every 1 ms, this results in an overhead of 0.067 % and 0.0067 % in, respectively, 1 and 10 Gbps connections.

4.4. EXPERIMENTAL EVALUATION

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

4.4.1. TESTBED ENVIRONMENTS

We have performed our experiments on two types of physical testbeds, being a testbed of *software* switches and a testbed of *hardware* switches.

Our software switch based testbed consists of 24 physical, general-purpose, servers enhanced with multiple network interfaces and software to run as networking nodes. Each server contains a 64 bit Quad-Core Intel Xeon CPU running at 3.00GHz with 4.00 GB of main memory and has 6 independent 1 Gbps networking interfaces installed and can hence perform the role of a 6-degree networking node. Links between nodes are realized using physical Ethernet connections, hence deleting any measurement inaccuracies introduced by possible intermediate switches or virtual overlays. OpenFlow switch capability is realized using the Open vSwitch [77] software implementation, installed on the Ubuntu 13.10 operating system running GNU/Linux kernel version 3.11.0-12-generic.

The hardware switch based testbed we used was graciously made available to us by SURFnet, the National Research and Education Network of the Netherlands. The testbed consists of 5 Pica8 P3920 switches, running firmware release 2.0.4. The switches run in Open vSwitch mode to deliver OpenFlow functionality, meaning that they implement the same interfaces as defined by Open vSwitch.

4.4.2. RECOVERY AND MEASUREMENT TECHNIQUES

We use two complementary techniques to implement our proposal: (1) We use OpenFlow's Fast Failover Group Tables to quickly select a preconfigured backup path in case of link failure. The Fast Failover Group Tables continuously monitor a set of ports, while incoming packets destined for a specific Group Table are forwarded to the first active and alive port from its set of monitored ports. Therefore, when link functionality of the primary output port fails, the Group Table will automatically turn to the secondary set of output actions. After restoring link functionality, the Group Tables revert to the primary path. (2) Link-failure itself is detected using the BFD protocol. We chose a BFD transmit interval conform equation (4.3), with $\beta = 2$ to account for irregularities by queuing. Although the RTT allowed smaller transmit intervals, the implementation of BFD forced a minimum window of 1 ms. We use a detection multiplier of $M = 3$ lost control messages to prevent false positives.

At the time of writing, both BFD and Group Table functionality are implemented in the most recent snapshots from Open vSwitch' development repository [106]. Although both BFD status reports and Fast Failover functionality operate correctly, we found the Fast Failover functionality did not include the reported BFD status to initiate the backup sequence and only acts on (administrative) link down events. To resolve the former problem, we wrote a small patch for Open vSwitch to include the reported interface BFD status [107]. Furthermore, the patch includes minor changes to allow BFD transmit intervals smaller than 100 ms.

In order to simulate traffic on the network we use the *pktgen* [108] packet generator, which can generate packets with a fixed delay and sequence numbers. We use the missing sequence numbers of captured packets to determine the start and recovery time of the failure. *Pktgen* operates from kernel space, therefore high timing accuracy is expected and was confirmed at an interval accuracy of 0.005 ms. In order to get 0.1 ms timing accuracy, *pktgen* is configured to transmit packets at a 0.05 ms interval.

4.4.3. EXPERIMENTS

This subsection describes the performed experiments. We run baseline experiments using link-failure detection by regular Loss-of-Signal on both testbeds. Additionally, we run experiments using link-failure detection by per-link BFD sessions on the software switch testbed to show improvement. In general, links are disconnected using the Linux *ifdown* command. To prevent the administrative interface change to influence the experiment, the *ifdown* command is issued from the switch opposite to the switch performing recovery.

The experiments are executed as follows. We start our packet generator and capturer at $t = 0$. At $t_{failure}$, a failure is introduced in the primary path. The packet capture program records missing sequence numbers, while the failover mechanism autonomously detects the failure and converts to the backup path. At $t_{recovery}$, connectivity is restored, which is detected by a continuation of arriving packets, the recording of missing packets is stopped and the recovery time is computed.

BASIC FUNCTIONALITY

In our first experiment we measure and compare the time needed to regain connectivity in a simple network to prove basic functionality. The network is depicted in figure 4.3a and consists of two hosts named H_1 and H_2, which connect via paths $A \rightarrow B \rightarrow C$ (primary) and $A \rightarrow C$ (backup). In this experiment, we stream data from H_1 to H_2 and deliberately break the primary path by disconnecting link $A \leftrightarrow B$ and measure the time needed to regain connectivity.

CRANKBACK

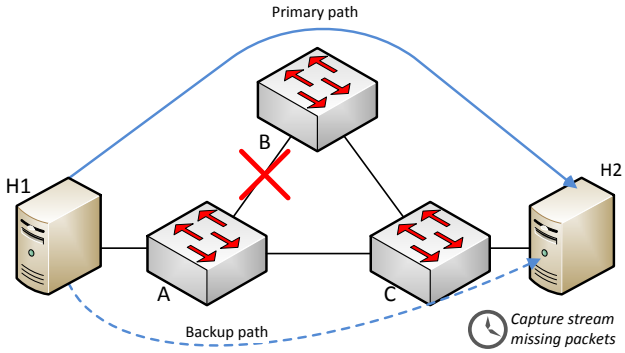
After confirming restoration functionality we need to confirm crankback functionality. To introduce crankback routing, we use a slightly more complicated ring topology in figure 4.3b, in which the primary path is set as $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E$. We break link $C \leftrightarrow D$ enforcing the largest crankback path to activate, resulting in the path $A \rightarrow B \rightarrow C \rightarrow B \rightarrow A \rightarrow E$.

EXTENDED EXPERIMENTS

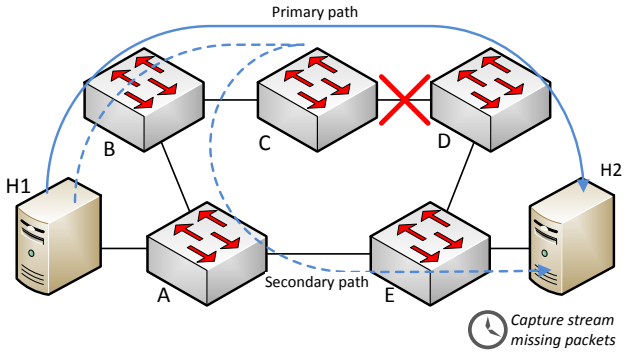
To test scalability, we perform additional experiments in which we simulate a real-world network scenario by configuring our software-switch testbed in the USNET topology shown in figure 4.3c. We set up connections from all East-coast routers to all West-coast routers, thereby exploiting 20 shortest paths. We configure each switch to initially forward packets along the shortest path to destination, as well as a backup path from each intermediate switch to destination omitting the protected link. At each iteration we uniformly select a link from the set of links participating in one or more shortest paths and break it.

4.4.4. RESULTS AND ANALYSIS

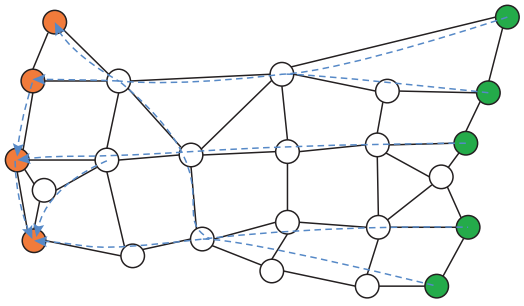
Our first set of results, displayed in figure 4.4, shows baseline measurements without BFD performed for the simple topology on both testbeds. This experiment shows link-failure detection based on Loss-of-Signal implies an infeasibly long recovery time in both our software and hardware switch testbeds. The Open vSwitch testbed shows an average recovery time of 4759 ms. Although the hardware switch testbed performs better



(a) Simple Network Topology.



(b) Ring Network Topology.



(c) USNET Topology.

Figure 4.3: Topologies used in our experiments, including functional and failover scenarios in case of specific link-failure.

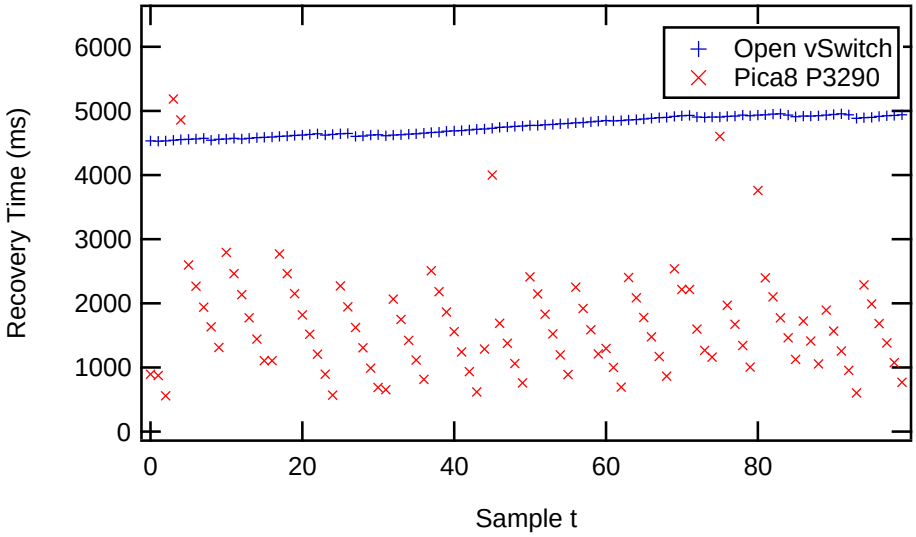


Figure 4.4: Baseline measurements using Loss-of-Signal failure detection.

with an average recovery time of 1697 ms, the duration of both recoveries are unacceptable in carrier-grade networks. The samples of both the software and hardware switches show periodic increases followed by a larger decrease in recovery time, we conjecture that these are system anomalies introduced by status timers internal to the switches running at a different interval than our sample repetition. Ultimately, all samples including both the low and high deviations introduced by this system behavior support our finding of unacceptably long recovery times.

In order to determine the time consumed by administrative processing additional to actual failure detection, we repeated previous experiments with the exception that we administratively brought down the interface at the switch performing the recovery. By doing this, we only trigger and measure the time consumed by the administrative processes managing link status. We found that Open vSwitch on average needs 50.9 ms to restore connectivity after detection occurs. Due to this high value, our patch omits the administrative link status update and directly checks BFD status instead.

Currently, the firmware of the hardware switches does not support BFD, therefore we only verified recovery using link-failure detection by BFD on the software switch testbed. Figure 4.5 shows 100 samples of measured recovery delay for the simple and ring topologies using three different BFD transmit intervals, namely 15 ms, 5 ms and 1 ms. For the simple topology, we measure a recovery time that was a factor between 2.5 and 4.1 ms larger than the configured BFD transmit interval due to the detection multiplier of $M = 3$. As shown in figure 4.7, for each BFD transmit interval we measure average and 95% confidence interval recovery times of 3.4 ± 0.7 ms, 15.0 ± 2.2 ms and 42.1 ± 5.0 ms. The results show that, especially with links having low RTTs, a great decrease in recovery time can be reached. At a BFD transmit interval of 1 ms, we reach a maximal recovery time of 6.7 ms.

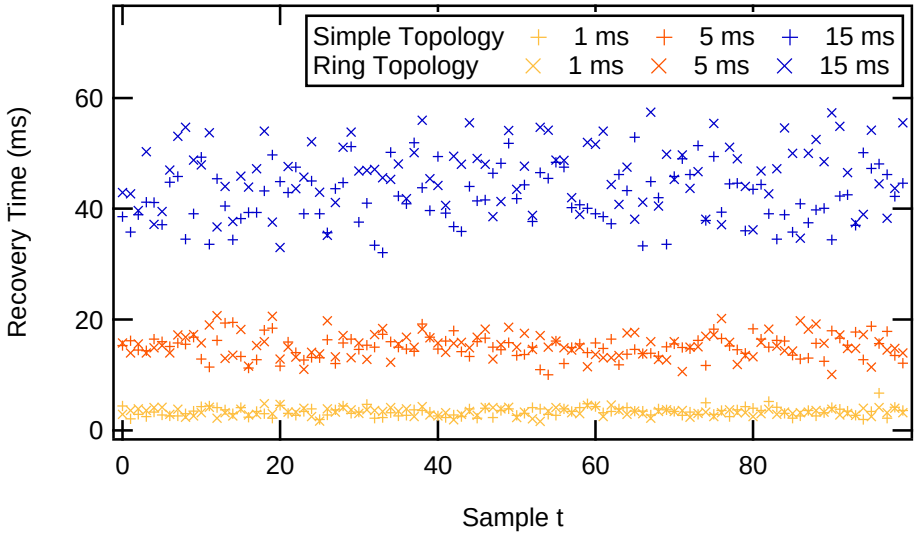


Figure 4.5: Recovery times for selected BFD transmit intervals and topologies.

For the ring topology, after 100 samples of measuring the recovery time for three different BFD transmit intervals, figure 4.7 shows averages and 95% confidence intervals of respectively 3.3 ± 0.8 ms, 15.4 ± 2.7 ms and 46.2 ± 5.1 ms for the different selected transmit intervals. At a BFD transmit interval of 1 ms, we reach a maximal recovery time of 4.8 ms. Even though the ring topology is almost *twice as large* as the simple topology, recovery times remain constant due to the per-link discovery of failures and the failover crankback route.

In order to verify the scalability of our implementation, we repeated the experiments by configuring our software-switch testbed with the USNET topology, hence simulating a real-world network. For each iteration, we average the recovery times experienced by the affected destinations. Figure 4.6 shows over 3400 samples taken at a BFD transmit interval of 5 ms, resulting in an average and 95% confidence interval of 13.6 ± 2.6 also shown in figure 4.7. The high frequency of arriving BFD and probe packets congested the software implementation of Open vSwitch, showing the necessity for hardware line card support of BFD sessions. As a consequence, we were unable to further decrease the polling frequency and had to decrease the frequency of probe packets to 1 per ms, slightly reducing the accuracy of this set of measurements. Although we experience software system integration issues showing the need to optimize switches for degree and traffic throughput, *recovery times remain constant independent of path length and network size*, showing our implementation also scales to larger real-world networks.

Prior to implementing BFD in the Group Tables, the Fast Failover Group Tables showed a 2 second packet loss when reverting to the primary path after repair of a failure. With BFD, switch-over is delayed until link status is confirmed to be restored and no packet loss occurs, resulting in a higher stability of the network.

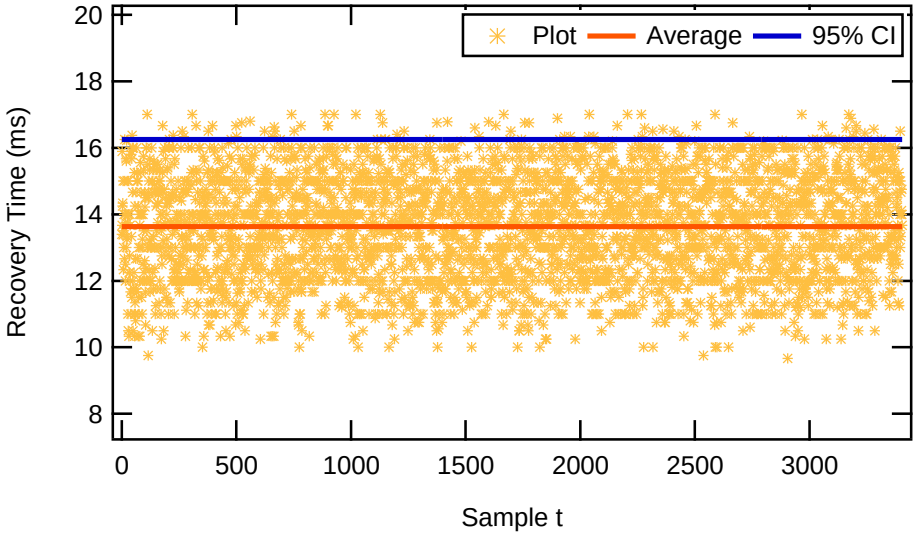


Figure 4.6: Recovery times for 5 ms transmit interval in the extended topology.

4.5. RELATED WORK

Sharma et al. [109] show that a controller-initiated recovery needs approximately 100 ms to regain network connectivity. They propose to replace controller-initiated recovery with a path-failure detection using BFD and preconfigured backup paths. The switches executing BFD detect path failure and revert to previously programmed backup paths without controller intervention resulting in a reaction time between 42 and 48 ms. However, the correctness and speed of detecting path failure depends highly on the configuration and switch implementation of BFD. According to [13], a detection time of 50 ms can be achieved by using an "aggressive session", using a transmit interval of 16.7 ms and window multiplier of 3. However, the authors of [109] do not provide details on their configuration of BFD. The BFD transmit interval is lower-bounded by the propagation delay between the end-points of the BFD sessions, therefore we claim a path-failure detection under-performs compared to a per-link failure detection and protection scheme as presented in this chapter.

Kempf et al. [110][111] propose an alternative OpenFlow switch design that allows integrated operations, administration and management (OAM) execution, foremost connectivity monitoring, in MPLS networks by introducing logical group ports. Introducing ring topologies in Ethernet-enabled networks and applying link-failure detection results to trigger failover forwarding rules, results in an average failover time of 101.5 ms [112] with few peaks between 140 ms and 200 ms. However, the introduced logical group ports remain unstandardized and are hence not part of shipped OpenFlow switches. Finally, Sgambelluri et al. [113] perform segment protection in Ethernet networks depending on OpenFlow's auto-reject function to remove flows of failed interfaces. The largest part of their experimental work involves Mininet emulations, which is considered inaccurate in

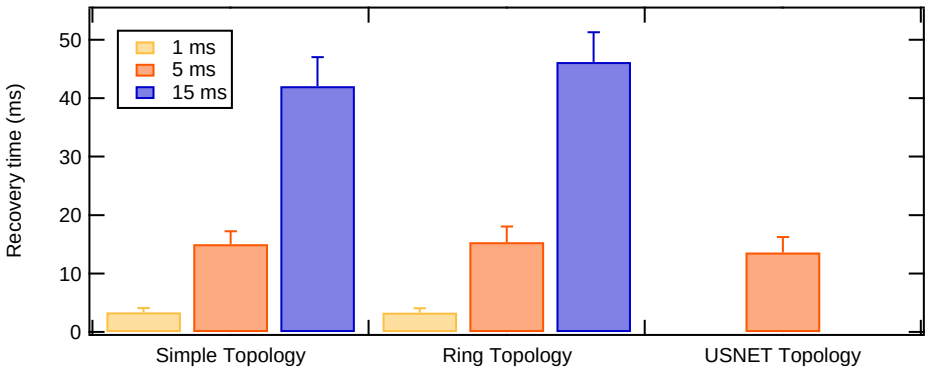


Figure 4.7: Bar diagram summarizing average recovery times and 95% confidence interval deviation on our Open vSwitch testbed using BFD at selected transmit intervals.

terms of timing due to its high level of virtualization [114]. The experiment performed on a physical testbed shows a switch-over time of at most 64 ms, however, the authors do not describe the method of link-failure detection, which make the results difficult to reproduce.

Table 4.1 shows the main differences between the aforementioned proposals and our work. Where the previous best recovery time was close to 30 ms, section 4.4 showed that our configuration regains connectivity within a mere 3.3 ms.

4.6. CONCLUSION

Key to supporting high availability services in a network is the network's ability to quickly recover from failures. In this chapter we have proposed a combination of protection in OpenFlow Software-Defined Networks through preconfigured backup paths and fast link failure detection. Primary and secondary path pairs are configured via OpenFlow's Fast Failover Group Tables enabling path protection in case of link failure, deploying crankback routing when necessary. We configure per-link, in contrast to the regular per-path, Bidirectional Forwarding Detection (BFD) sessions to quickly detect link failures. This limits the number of traversing BFD sessions to be linear to the network size and minimizes session RTT, enabling us to further decrease the BFD sessions' window intervals to detect link failures faster. By integrating each interface's link status into Open vSwitch' Fast Failover Group Table implementation, we further optimize the recovery time.

After performing baseline measurements of link failure detection by Loss-of-Signal on both a software and hardware switch testbed, we have performed experiments using BFD on our adapted software switch testbed. In our experiments we have evaluated recovery times by counting lost segments in a data stream. Our measurements show we already reach a recovery time considered necessary for carrier-grade performance at a 15 ms BFD transmit interval, being sub 50 ms. By further decreasing the BFD interval

Table 4.1: Summary of results and most important differences compared to related work.

Reference	*Avg recovery time* ±95% *CI*	*Network size* ($\|N\|, \|L\|$)	*Detection Mechanism*	*Recovery Mechanism*
[109]	42 – 48ms	(28, 40)	Per-path BFD	OpenFlow Fast Failover Group Tables using virtual ports
[110][111]	28.2 ± 0.5 ms	N.A.	Per LSP OAM + BFD	Custom extension of OpenFlow
[112][113]	32.74 ± 4.17 ms	(7, 7)	Undocumented	Custom auto-reject mechanism
This work	3.3 ± 0.8 ms	(24, 43)	Per-link BFD	Commodity OpenFlow Fast Failover Group Tables

to 1 ms we reach an even faster recovery time of 3.3 ms, which we consider essential as network demands proceed to increase. Compared to average recovery times varying from 28.2 to 48 ms in previous work, we show an enormous improvement.

Since we perform per-link failure detection and preconfigure each node on a path with a backup path, the achieved recovery time is independent of path length and network size, which is confirmed by experimental evaluation.

5

All-to-all Network Topology Failure Protection in SDNs

In the previous chapter, we designed a system that provides fast detection of link failures and reverts to backup paths to circumvent those failures. Although we proved through experiments that such a system is feasible for production networks, the system assumed that backup paths were already previously computed and configured. However, computation of these paths prior to the actual link or node failure is far from trivial, especially when those types of paths need to be precomputed from and to each and every node in the network. In this chapter, we propose algorithms for computing an all-to-all primary and backup network forwarding configuration that is capable of circumventing link and node failures. After initial recovery, we recompute the network configuration to guarantee protection from future failures. Our algorithms use packet-labelling to guarantee correct and shortest detour forwarding and are able to discriminate between link and node failures. The computational complexity of our solution is comparable to that of all-to-all shortest paths computations. Our experimental evaluation on both real and generated networks shows that network configuration complexity decreases significantly compared to classic disjoint paths computations. Finally, we proved a proof-of-concept OpenFlow controller in which our proposed configuration is implemented, demonstrating that it readily can be applied in production networks.

This chapter is based on a published paper [115] and report [116].

5.1. INTRODUCTION

Modern telecommunication networks deliver a multitude of high-speed communication services through large-scale connection-oriented and packet-switched networks running on top of optical networks, Digital Subscriber Lines (DSLs), cable connections or even wireless terrestrial and satellite links. As society heavily depends on modern telecommunication networks, much has been done to prevent network failure, e.g., by improving the equipment environment and physical aspects of the material. However, the past century of telecommunications shows that network components still fail regularly [82]. Regardless of the preventive protection measures taken, network nodes and links will eventually malfunction and cease to function.

In connection-oriented networks, e.g. wavelength-routed networks, network service interruptions due to the failure of network nodes or links can often be prevented by assigning at least two disjoint paths from the source node to the destination node of each network connection [117]. Connection status is then monitored from the source node to the destination node. When the primary path of a network connection fails, the connection can be reconfigured to use its backup path instead. Traffic can also be sent on the primary and backup paths of a connection concurrently, such that reconfiguration upon the failure of the primary path is not needed. Although finding a pair of (min-sum) disjoint paths from a source node to a destination node is polynomially solvable [118, 119], the returned paths may each be substantially longer than the shortest possible path between the nodes due to the existence of trap topologies [120]. An alternative would be to find a pair of min-min disjoint paths, where the weight of the primary path is to be minimized, instead of the sum of the weights of both paths (min-sum). However, the problem will then be NP-hard [121].

Packet-switched networks, e.g., Ethernet or IP networks, have no connection status since packets are forwarded in a hop-by-hop manner through local inspection of headers at each router it traverses. Though using disjoint paths is possible in packet-switched networks through end-to-end liveliness detection monitoring schemes (such as Bidirectional Forwarding Detection (BFD) [13], Ethernet OAM/CFM [14] or IP Fast Reroute [122]), the approach is more complex than in connection-oriented networks. Any network node may send messages to any other network node without prior reservation, leading to $O(|N|^2)$ (where $|N|$ is the number of network nodes) possible monitoring sessions for each source-destination pair: a great increase compared to the connection-oriented network, where the number of monitoring sessions are bounded by $O(|C|)$ (where $|C|$ is the number of factual connections). Hence, computing and maintaining disjoint paths for all possible node pairs may be suboptimal in packet-switched networks.

However, in packet-switched networks, traffic can be rerouted along a part of the primary path, which is not possible in connection-oriented networks. Each intermediate node along the primary path has the capability of forwarding packets through another link interface when necessary. Furthermore, after packets have been rerouted past the failure, packets are directed to the shortest remaining path towards the destination, possibly by following the remainder of the (initial) primary path that is unaffected by the failure. However, configuring an all-to-all configuration requires complex forwarding rule constructions, which is difficult to realize with traditional distributed routing proto-

cols that operate on embedded systems with lower computational and memory capacities. A Software-Defined Networking (SDN) approach may facilitate implementing such network functionality.

SDN enables the use of a controller for recomputing the network state reactively upon a failure, but incurs high processing delays [109]. In chapter 4, we provided an overview of SDN-specific related work on topology recovery mechanisms and have shown that failure recovery in OpenFlow-based SDN networks is best handled in three steps, being 1) fast failure detection through liveliness monitoring protocols, 2) failure protection through computation and configuration of backup rules prior to failure, which is the fastest recovery approach possible but may not deliver optimal network configuration, and 3) recomputation of optimal network state and new backup paths as soon as the failure detection has propagated to the network controller.

In chapter 4, we showed very fast results providing fast failure detection and reovery, but assumed the configuration of backup rules to be present. In this chapter, we explore existing algorithmic solutions and propose new ones to compute a network configuration that guarantees all-to-all network connectivity against any single node or link failure. Our aim is to be able to automatically configure and reconfigure any SDN networks with failure protection schemes without human intervention.

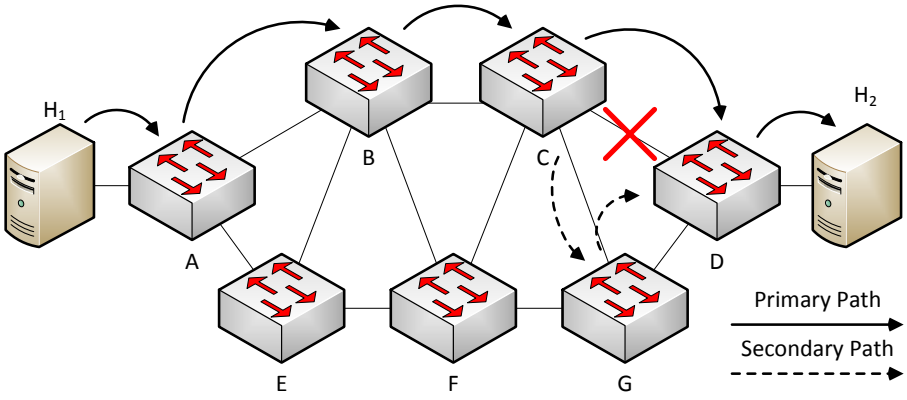
Our contributions in this chapter are three-fold:

1. We derive the hard and soft constraints that should be incorporated by a resilient routing configuration.
2. We present and evaluate algorithms for computing paths that meet those constraints in circumventing failures.
3. We implement and experiment with the presented algorithms in an SDN controller.

The remainder of the chapter is organized as follows. In section 5.2, we define the problem and give examples of what we need to compute and how traditional disjoint paths algorithms fail in doing so. Section 5.3 presents our algorithms finding failure-disjoint paths, which we evaluate and analyze in section 5.4. Our prototype SDN controller implementation is presented in section 5.5. Section 5.6 presents related work on finding disjoint paths and computes their overall complexity when applied to our problem. Finally, section 5.7 concludes this chapter.

5.2. PROBLEM STATEMENT

Figure 5.1a shows an example of a shortest path through a sample network, and a link failure between nodes C and D. Although we are looking for an all-to-all solution, for illustration purposes we will use the example of traffic flowing from node H_1 to node H_2 in the network. The primary path of the traffic, which is the shortest path, breaks by the failure of link l_{CD}, an event only noticeable by node C, which is an intermediate node along the primary path. In order for the traffic to arrive at node H_2, there must be an alternative rule to revert to at node C that will ultimately route the traffic to node H_2. In essence, we are looking for an all-to-all solution in which all nodes are preconfigured with backup forwarding rules to overcome any such single link or node failure



(a) Failure-disjoint path.

Figure 5.1: Failure-disjoint paths and labels used in forwarding from H_1 to H_2.

in the network. Moreover, since those rules will be computed for each possible specific single link/node failure, both the primary and backup paths will be as short as possible in length, which is a big gain over traditional path-disjoint protection schemes. The problem can be formally defined as follows.

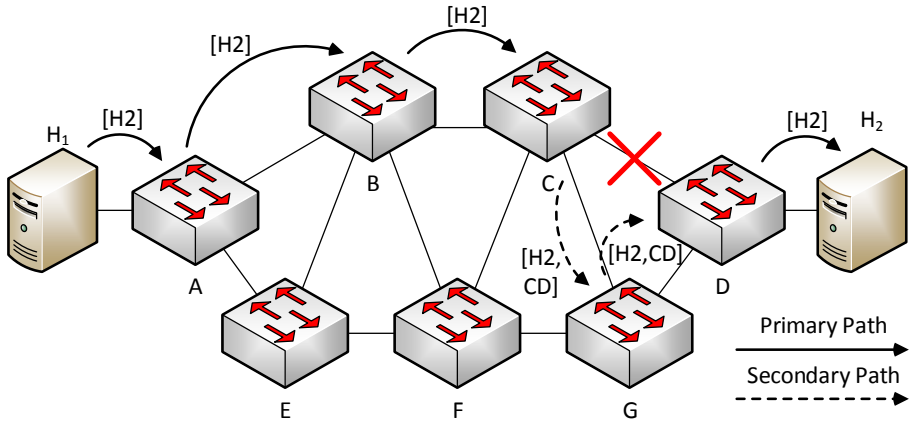
Single Failure Avoidance Rule Assignment (SFARA) problem: Given a directed network G of a set N of $|N|$ nodes and a set L of $|L|$ directed links. Each link $l_{uv} \in L$ connects nodes u and v, and is characterized by a link weight ℓ_{uv} and a boolean link status s_{uv} indicating link functionality. $s_{uv} = up$ implies that link l_{uv} is functioning normally, while $s_{uv} \neq up$ implies that link l_{uv} is not functioning. Find an overall set of primary and backup forwarding rules such that any possible source node $x \in N$ can send packets to any possible destination node $y \in N$ when all links are operational ($\forall l_{uv} \in L : s_{uv} = up$), or under a single link (or node) failure ($\exists! l_{uv} \in L : s_{uv} \neq up$).

The following constraints exist for the SFARA problem:

1. The status s_{uv} of each link $l_{uv} \in L$ is only available from its adjacent nodes u and v, and may be used in the forwarding logic of nodes u and v. For example, $(s_{uv} = up)?(output(l_{uv})) : (output(l_{uw}))$ describes the forwarding logic where node u forwards packets to node v when link l_{uv} is operational, or to node w over link l_{uw} otherwise. Node u thereby relies on node w to have a suitable backup path towards the destination.

2. A set of forwarding actions can be performed on a packet at each node, including (a) dropping it, (b) rewriting, adding or removing any of its labels - such as source and destination addresses, VLAN tags and MPLS labels, and (c) forwarding it to the next node by outputting it to a specific output port or link.

3. The appropriate forwarding actions for each packet are selected from a forwarding



(b) Labels used in forwarding.

Figure 5.1: Failure disjoint paths and labels used in forwarding from H_1 to H_2.

table based on properties such as: (a) the packet's incoming port, (b) (wildcard) matching on packet labels, such as its Ethernet addresses, IP addresses, TCP or UDP source and destination address, VLAN tags, MPLS labels, etc., and (c) status of the outgoing links of the router or switch.

5.3. PER-FAILURE PRECOMPUTATION

As shown earlier in figure 5.1a, disjoint-path based forwarding rules cannot instruct node C on how to circumvent the failed link. Node C can only send the packet back to the source node through crankback routing, which is an expensive process since it uses twice the network resources from the source node to the failed link plus the network resources on the disjoint-path. Instead, we propose to use a detour around the failure as shown in figure 5.1b, optimizing the primary path to the shortest path when possible.

We explain our algorithm for finding and configuring link-failure disjoint paths using labeling techniques in subsection 5.3.1, and later modify it to node-failure disjoint paths in subsection 5.3.2. Knowing whether a link or node failure has manifested can be difficult since each node can only determine that an adjacent link is broken, while the failure may only be limited to the reported link or may include the adjacent node. A conservative approach is to assume that all detections of link-failures imply node failures, but this leads to higher detours and possible false-negatives in determining whether a detour path to the destination node exists. Subsection 5.3.3 thus presents our hybrid adaptation from the link- and node-failure disjoint paths where we use a labelling technique to "upgrade" a link-failure to a node-failure only when necessary, and adapt the forwarding strategy accordingly. Finally, subsection 5.3.4 discusses how we optimize routing table complexity by removing redundant rules.

Algorithm 5.1 Per-link approach

Input: Adjacency matrix $adj = G(N, L)$
Output: Forwarding matrix fw containing primary and backup rules

```
1: set fw to all-to-all shortest paths matrix
2: for each node n ∈ N
3:     for each outgoing link l of n
4:         set tAdj to shadow copy of adj
5:         remove link l from tAdj
6:         set {n'} from N where nextLink = l
7:         compute 1-to-{n'} shortest paths from tAdj:
8:             store all nextLink as fw[(curNode, l)][n']
9: return fw
```

5.3.1. LINK-FAILURE DISJOINT PATHS

Algorithm 5.1 presents our algorithm for computing primary and backup forwarding rules for all possible source-destination pairs given that at most one link is broken at any time. The algorithm computes primary and backup forwarding rules for the whole network, such that it is resilient to any single link failure. The algorithm first optimizes the length of the primary path, and then optimizes the length of the detour towards the destination node for all possible link failures.

Line 1 computes a regular all-to-all shortest paths matrix, using algorithms such as $|N|$ iterations of Dijkstra's algorithm [56] or the Bellman-Ford algorithm [47][123], as long as it supports the link weights in consideration[1]. Lines 2 and 3 iterate through all nodes' outgoing links. Since any link connects exactly two nodes this results in a combined complexity of $2|L|$, leading to an intermediate complexity determined by $|N|$ times the one-to-all shortest paths computations and $O(|L|)$ for the following procedure. Line 4 creates a shadow copy of the adjacency matrix, which stores only the changes from the original. Calls to the shadow copy check for changes first and if absent return the original result from the original table. Since we remove at most one link, our shadow copy suffices to be a function call to the original table that filters out the one link before looking up the value in the matrix. Hence, creation and lookup both have a constant complexity. Line 5 removes the link under evaluation from the shadow copy, which has a complexity of $O(1)$. Line 6 selects all destinations whose shortest paths go through the removed link. Sets containing the shortest path destinations denoted per link can be created within a time complexity contained by any of the suggested shortest path algorithms and hence does not add to the overall complexity of the algorithm. Selection of the sets is done in constant time. Finally, lines 7 and 8 compute and store the backup paths using a regular one-to-all shortest paths computation (such as the Dijkstra's or the Bellman-Ford algorithms) with a slight change to the stop-criterion. First, line 7 indicates that the algorithms may stop when all currently unreachable nodes $\{n'\}$ have been found again, there is no need to find the shortest paths to all nodes. Line 8 adds the found forwarding

[1] In dense graphs one may consider to use Floyd-Warshall's algorithm [54, 55]. Although it is computationally more complex, $O(|N|^3)$, its memory complexity during computation is limited to the size of the input adjacency matrix and output forwarding matrix ($O(|N|^2)$).

Algorithm 5.2 Per-node approach, changes compared to algorithm 5.1 are underlined

Input: Adjacency matrix $adj = G(N, L)$
Output: Forwarding matrix fw containing primary and backup rules

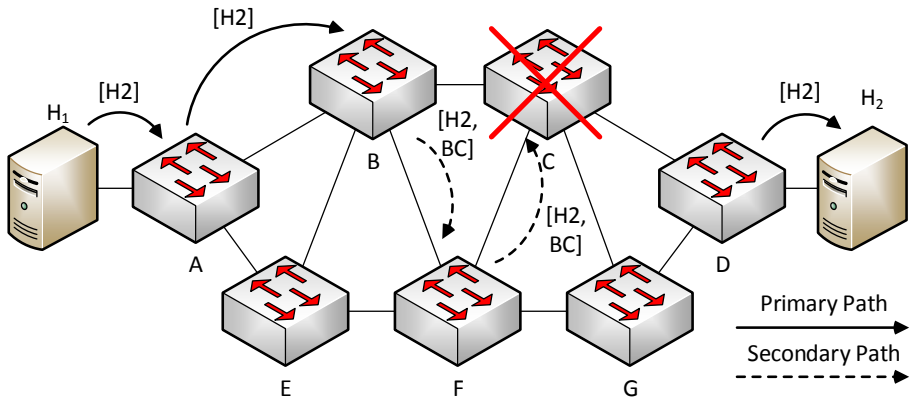
```
1:  set fw to all-to-all shortest paths matrix
2:  for each node n ∈ N
3:      for each outgoing link l of n
4:          set tAdj to shadow copy of adj
5:          set n^R to node opposite of link l
6:          remove node n^R and adjacent links from tAdj
7:          set {n'} from N where nextLink = l
8:          compute 1-to-{n'} shortest paths from tAdj:
9:              store all nextLink as fw[(curNode,n^R)][n']
10: return fw
```

rules to the original forwarding matrix. A distinction between the original and backup shortest path forwarding rules from a node n to its destination forwarding rules is made by saving it under a label identifying the specific failure, in this case link l. As presented in figure 5.1b, the node that initiates sending packets through backup paths should add a label identifying the failure it is detouring from. From this label nodes along the backup path derive that these packets need special treatment until they reach their destination or a shortest path that is not affected by the failure anymore. In the latter case, the label may be removed.

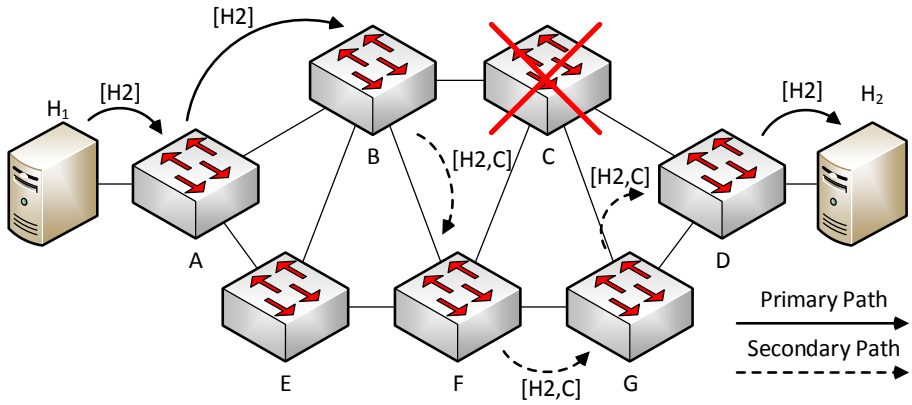
The overall complexity of the algorithm is mostly defined by the chosen shortest path algorithm. In general, our algorithm has a worst-case complexity of $O(|N|+|L|)$ times the complexity of the implemented shortest path algorithm, since we need $|N|$ iterations to derive the all-to-all forwarding table and need to recompute broken shortest paths twice for all $|L|$ links. Our solution optimizes shortest and backup path length in sequential order. Hence, it does not include Quality-of-Service constraints. Such functionality can be implemented by computing primary paths using a multi-constrained path algorithm (e.g. [95]), and subsequently computing the backup paths compared based on the remaining set of resources. The implementation and evaluation of this solution, however, is beyond the scope of this dissertation.

5.3.2. NODE-FAILURE DISJOINT PATHS

In the case of a node failure, algorithm 5.1 may not work as the node opposite to the detected broken link is not excluded from the backup path. Figure 5.2a shows how the selection of a link-disjoint path may send packets right back towards the broken node. Even if node F would select its link-disjoint backup path towards node H_2, this path is not guaranteed to be loop-free from a previous backup path. As suggested in figure 5.2b, in the case of a node failure we need a node-disjoint backup path that eliminates the failed node instead of individual links from the backup paths. Algorithm 5.2 presents our solution that computes primary and backup forwarding rules for all-to-all paths given that at most one node is broken. The algorithm computes primary and backup



(a) Node failure at node *C* incorrectly handled by link-disjoint backup path.



(b) Proper detour around failure of node *C*

Figure 5.2: Node failure disjointness and labels used in forwarding.

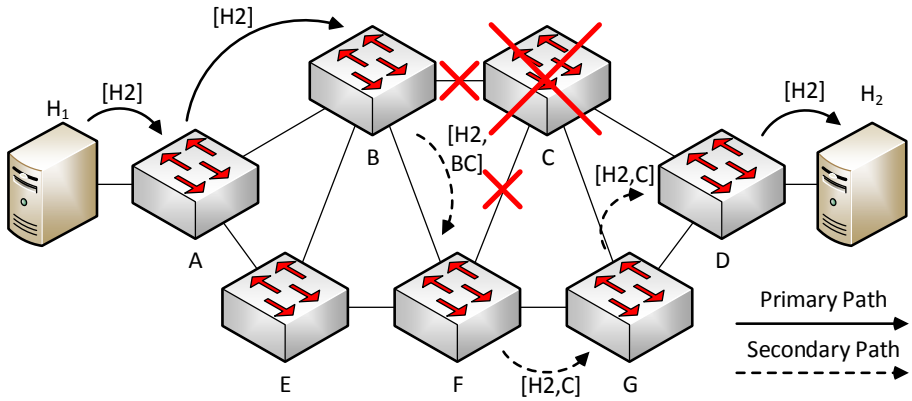


Figure 5.3: Link-, then node-failure disjoint approach and forwarding labels.

forwarding rules resilient to any single node failure. The algorithm is almost equal to algorithm 5.1, except for minor changes. The biggest change is found in lines 5 and 6, where instead of the removal of link l, its opposite node n^R is removed from the shadow copy. The stored label n^R is used in forwarding. The computational complexity remains unchanged.

5.3.3. HYBRID APPROACH

The biggest problem with link-failure disjoint paths is that they may show problems when the node opposite of the detected failed link is broken. The node that detects link failure cannot determine whether the link failure is a result of a single link failure or node failure that affects all the failed node's links. The trivial solution to use node-failure disjoint paths whenever possible may work, but implies longer backup paths as one cannot return to the opposite node when it is still functional and may break connectivity when there is no node-disjoint path available. In practice, link failures occur more than node-failures. Although the node asserting the backup path cannot know whether a link or node failure is present, we prefer a link-failure disjoint path whenever possible, and a node-disjoint path otherwise.

In order to accomplish such routing, as depicted in figure 5.3, we let the asserting node assume a link-failure and act accordingly to it by adding a label denoting link-failure and forwarding through the link-failure disjoint path. If any node along this backup path has a primary forwarding rule to the failed node through another of its links, it assumes node failure based on the local link-failure detection combined with the label on the incoming packets indicating it is not the first broken link of that node. Furthermore, this knowledge is added to the attached label. When every attached label of a failed link is a concatenation of its interconnecting nodes ($\{u, v\}$), a forwarding rule wildcard match such as $\{*, v\}$ can detect previous link failures to node v.

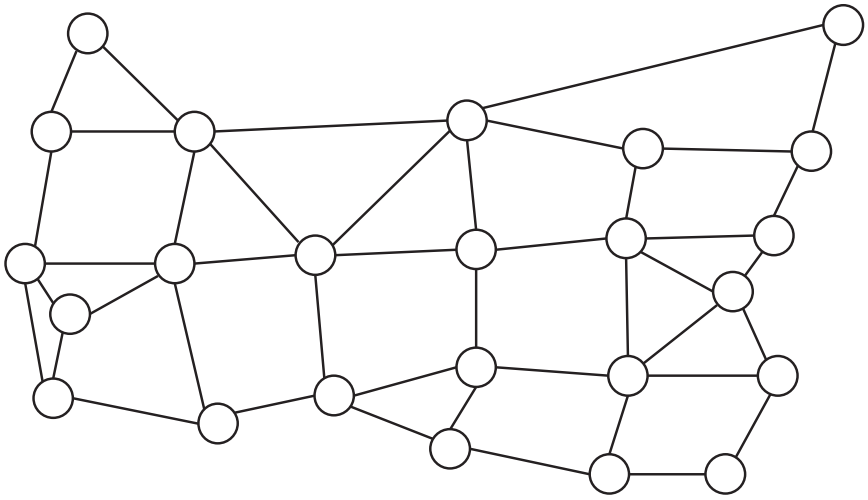


Figure 5.4: USnet topology with 24 nodes and 43 links.

To compute these rules, we compute both node- and link-failure disjoint paths and place these using their unique labels in the shared forwarding matrix. Note that the initial forwarding matrix only needs to be computed once, and removal of links and nodes and their respective recomputations may occur sequentially. This procedure runs in the same worst-case time complexity as the previous two algorithms.

5.3.4. ROUTING TABLE OPTIMIZATION

The procedures described in the previous subsections looks for min-min link-, node- and hybrid-failure disjoint paths. However, without optimizing forwarding rule complexity, this results in a state explosion of forwarding rules. While the realistic *USnet* topology (shown in figure 5.4) initially has a total of 552 forwarding rules (23 per switch, one for each destination), our link-, node- and hybrid-failure disjoint approaches change most of these from regular output actions to group tables and respectively leads to 1606, 2078 and 3684 (sum of previous two) additional entries in the forwarding matrix *fw*. Switches and routers employ Ternary Content-Addressable Memory (TCAM) to store and quickly lookup forwarding configuration. Although switches often have very large forwarding tables for Layer 2 matching, the number of TCAM entries in a switch for multiple field matches as performed in OpenFlow lies in the order of 1000 to 10000 rules [124]. Hence, it is necessary to minimize the number of forwarding rules to allow applicability in larger networks.

Considering that detoured packets at a certain point follow the default shortest paths from intermediate nodes on the backup path to destination, unaffected by the found failure, a first optimization is found by removing the failure-identifying label once a suitable default shortest path is found, leading to an addition of only 487 and 576 node- or link-failure disjoint entries, which is a big improvement. Since the hybrid-failure disjoint

path may not revert to a shortest path before a potential node-failure is omitted, we find an additional 1293 hybrid-failure disjoint entries, which, although larger than the sum of the previous two, is still a factor three lower than before.

Moreover, if we consider the USnet topology to be unweighted, hence introducing multiple shortest paths, we find an additional 621 and 741 node- and link-failure disjoint entries, which is larger than its weighted counter result, indicating that it is important for resilience in a network to have unique shortest paths.

We further optimize rulespace utilization by removing link-failure disjoint forwarding rules in the hybrid computation when they are equal to their respective node-failure disjoint rule, leading to a decreased number of 847 additional entries. A more extensive evaluation of our proposal compared to fully disjoint paths is presented in section 5.4.

Although counting entries in the forwarding matrix is still rather loose, no distinction between regular forwarding actions and group table entries are made although often two entries merge into one group table entry, it gives a good insight in the taken optimization steps.

5.4. EVALUATION

In this section, we study the performance of our algorithms through simulation in three network topologies, Erdös-Rényi random networks [125], lattice networks, and Waxman networks [126]. For our generated Erdös-Rényi random networks, we choose $\frac{2log|N|}{|N|}$ as the probability for link existence, since the network will almost surely be connected when the probability for link existence exceeds $\frac{(1+\varepsilon)log|N|}{|N|}$, where ε>0. In the lattice network, all interior nodes have a degree of four and the exterior nodes are connected to their closest exterior neighboring nodes. The lattice network is useful in representing grid-based networks, which may resemble the inner core of an ultra-long-reach optical data plane system [127]. We choose a square lattice network of $i \times i$ dimension, where $i = \sqrt{|N|}$, for our generated lattice networks. The Waxman network is frequently used to model communication networks and the Internet topology [128], due to its unique property of decaying link existence over distance. In the Waxman network, nodes are uniformly positioned in the plane, and link existence is reflected by $ie^{\frac{\ell_{uv}}{ja}}$, where ℓ_{uv} is the Euclidean distance between nodes u and v, a is the maximum distance between any two nodes in the plane, and i and j can vary between 0 to 1. We set $i = 0.5$ and $j = 0.5$ since higher i leads to higher link densities, and lower j leads to shorter links. We consider only two-connected generated graphs, such that the network can never be disconnected by a single node or link failure. In the Erdös-Rényi and lattice networks, each link has a random link weight between 0 and 1. No self-loops or parallel links are allowed. Simulations were conducted on an Intel(R) Core i7-3770K 3.50 GHz machine with 16GB RAM memory, and all results are averaged over a 1000 runs and grouped by the network sizes 9, 16, 25, 36, 49, 64, 81 and 100 due to the dimension of the lattice network.

We compute and compare the results of different disjoint algorithms, being our link-, node- and hybrid-failure disjoint approaches and min-sum pairs of fully link- and node-disjoint paths. More specifically, we measure:

1. the total number of flow entries
2. the number of flow entries that forward to a Group table entry
3. the number of distinct Group table entries
4. the average primary path length
5. the averages of
   (a) the average, minimal and maximal backup path length for each node pair and
   (b) the average and maximal crankback length for experienced backup paths.

We compute link-, node- and hybrid-failure disjoint paths according to our approach and link- and node-disjoint paths according to Bhandari's algorithm[2] for the generated networks, and calculate the enumerated values for these paths.

Figure 5.5 presents the average number of Flow table entries for each generated network. A regular shortest paths computation always generates exactly $|N|(|N|-1)$ Flow table entries (from each node to each other node). This number increases when more complex path computations are used. Specifically, we see a strikingly high increase in Flow table entries when fully-disjoint paths are used, which is caused by the fact that each forwarding rule has to take both source and destination into account for primary path forwarding, as well as the incoming port for crankback routing. As also shown in table 5.1, our failure-disjoint proposal shows an increase in Flow table entries varying from 15.7% to 38% for a network size of $|N| = 100$ nodes, whereas for the fully-disjoint computations this is limited from no increase to 7.7%. Given that fully-disjoint paths lead to an increased table usage by a factor of 21, our method appears to be much more conservative in Flow table usage. Whereas we found that our proposal uses significantly fewer flow table entries, figure 5.6 shows up to 94% of these are forwarded to Group table entries compared to a worst case of 44% for a fully disjoint path. Although this looks like a significant increase, the absolute number of Flow entries forwarding to Group table entries remains much lower in all cases. Moreover, table 5.1 and figure 5.7 show that our proposal contains a significantly lower usage of distinct Group table entries in each network, which are considered scarce resources.

Besides a smaller configuration complexity, also the primary paths taken are better. While the primary path in our proposal always defaults to the shortest path, figure 5.8 shows that using fully-disjoint paths leads to an increase of primary path lengths of up to 5.0%, and thus incurs higher network operation costs. Although the increase of primary path length of disjoint paths in most cases grows and at a certain point stabilizes, with

[2] Note that any other min-sum disjoint paths algorithm renders the same results, given that solutions are unique or an equal tossing method is used.

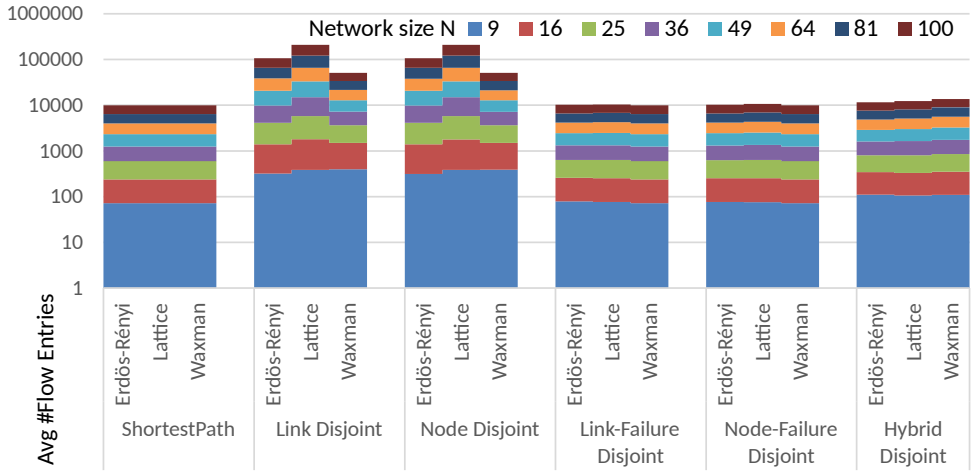


Figure 5.5: Average number of Flow entries in each network, categorized per network type and disjoint computation and incrementally stacked per network size.

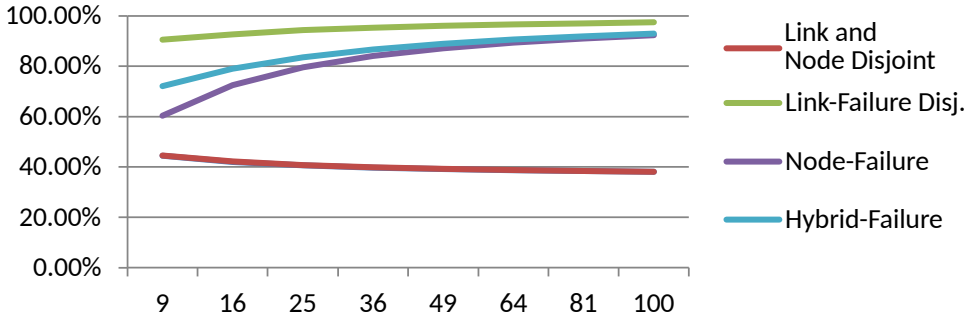


Figure 5.6: Ratio of forwards to Group table entries for Erdös-Rényi generated random networks of increasing size $|N|$.

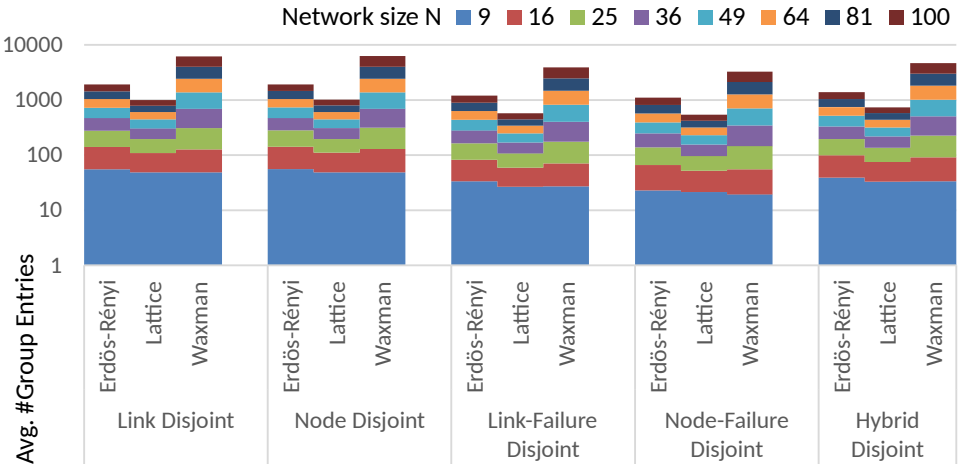


Figure 5.7: Average number of distinct Group Entries in each network, categorized per network type and of disjoint computation and incrementally stacked per network size.

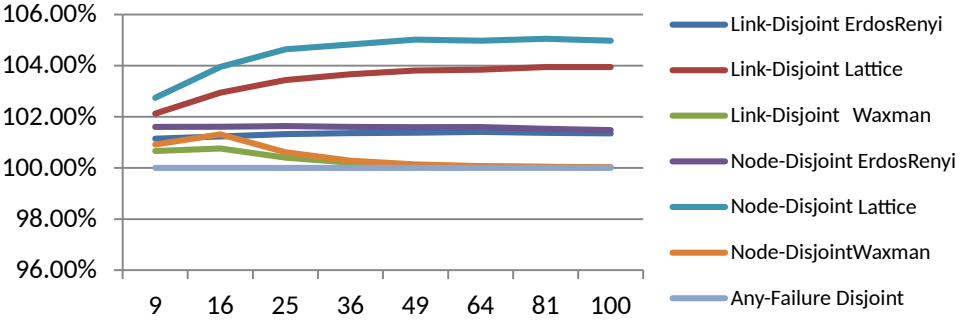


Figure 5.8: Increase in primary path per network size, categorized by algorithm and network type.

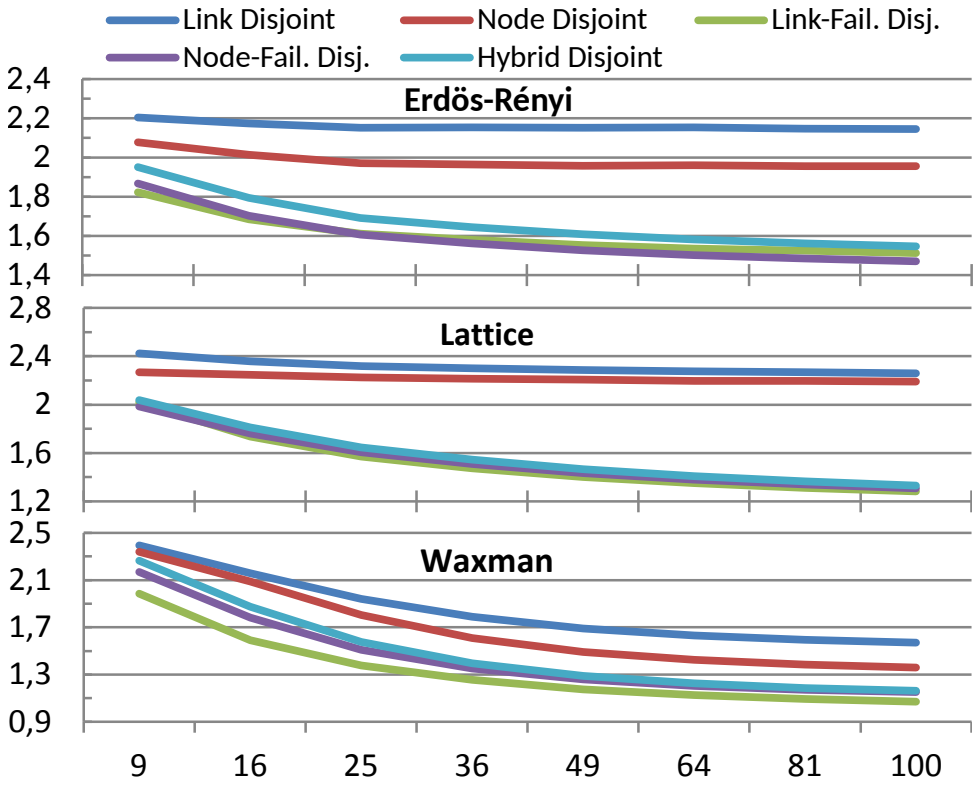


Figure 5.9: Average length of backup paths relative to length of their respective shortest path.

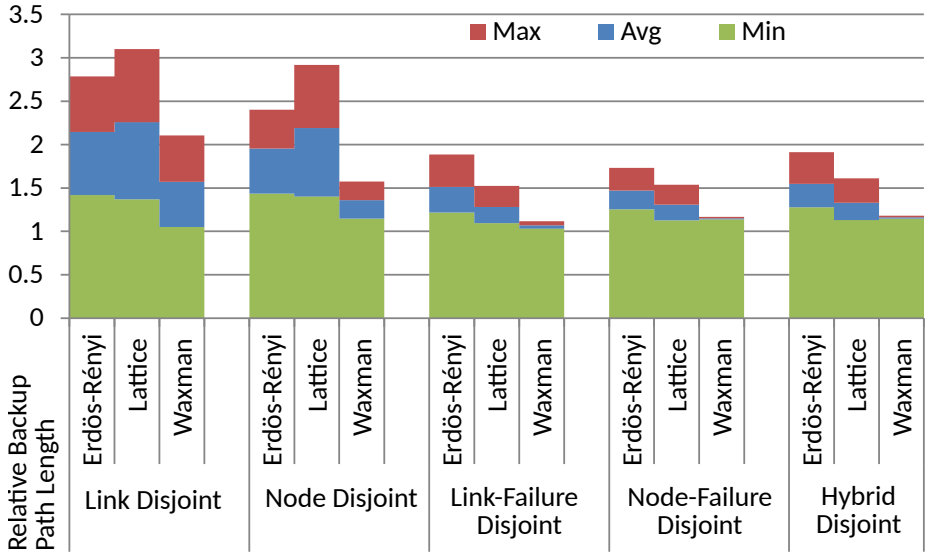


Figure 5.10: Comparison between minimal, average and maximal relative backup path lengths for networks of size $|N| = 100$.

Waxman-generated networks the path increase decreases over time implying that the design of the network has implications for the relative cost of robustness.

Figure 5.9 shows that besides a shorter primary path, our proposal on average also has significantly shorter average backup paths. In order to determine the average backup path for a node pair, we took its primary path and for each link or node on the path computed the length of the path if that specific link or node would fail and averaged accordingly. Hence, as figure 5.10 shows, the average backup path deviates significantly based on the link that fails. Especially the fully-disjoint paths suffer from a high deviation due to the high order of crankback routing that is involved when a link further down the primary path breaks. Figures 5.11 and 5.12 additionally show that the ratio and deviation of crankback paths is much larger for fully-disjoint paths than for our approach. Furthermore, crankback paths only exist temporarily in our proposal, since the controller reconfigures the network by applying the protection scheme to its newly established topology once it is notified of the failure, thereby removing existing crankback subpaths from the shortest paths.

The hybrid-failure disjoint path lengths are only shown for a node failure, since the path lengths for a respective link failure are equal to the results in the link-disjoint approach by design. Although the number of Flow and Group table entries, as well as the secondary path and crankback length for node failures slightly increases in the hybrid-failure approach, we claim this number is justified by the merits of shorter paths for the more often occurring link failures.

Although no exact measurements were made, we found that our proposal had a much faster computation time than its fully-disjoint counterpart. Our hybrid approach in general took 4 seconds to finish, compared to 20 seconds in Lattice networks and even up to a minute in Erdös-Rényi- and Waxman-generated networks for the fully-disjoint approach. Hence, our implementation is much faster in computing a new network configuration that offers protection from a possible next failure.

5.5. SOFTWARE IMPLEMENTATION

In order to evaluate failure circumventing methods as described in the previous two sections, we have implemented an open-source prototype OpenFlow controller module that configures Software-Defined Networks with such backup rules.

We have used the Ryu controller framework [73] as basis for our implementation, which loads and executes our network application. The network topology is discovered by Ryu's built-in *switches* component, while host detection occurs by a simple MAC learning procedure in our application.

Our application is OpenFlow 1.1+ [104] compatible, since it depends on the Fast-Failover Group Tables to perform the switchover to backup paths. Tests have been performed using OpenFlow 1.3 [129] which is considered the current stable version of OpenFlow.

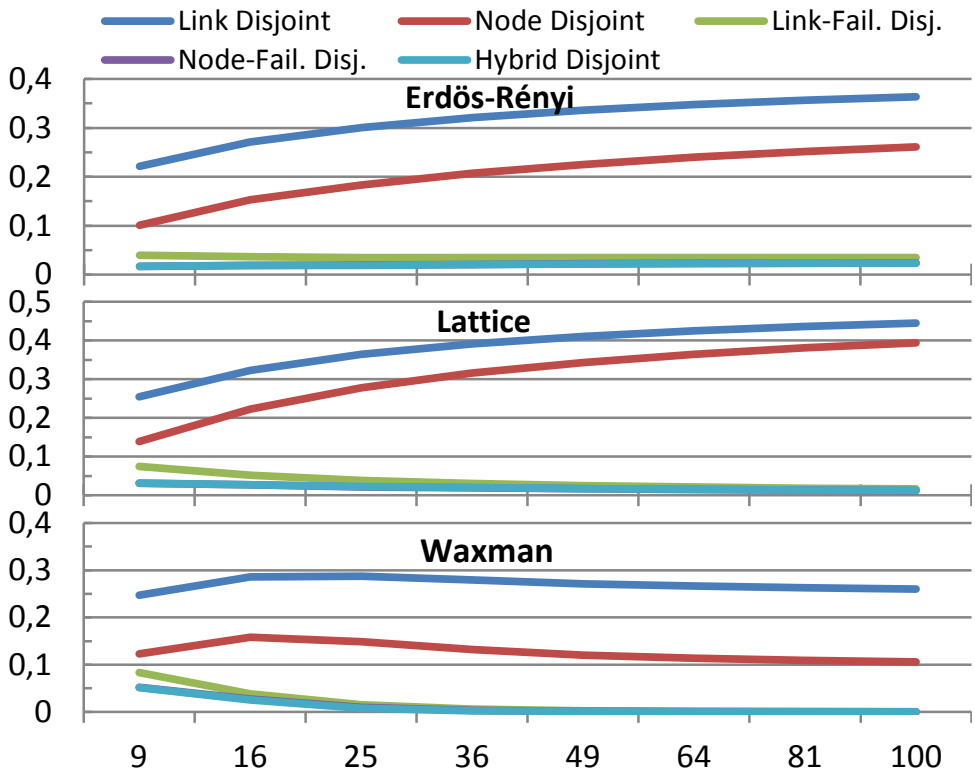


Figure 5.11: Average length of crankback paths relative to length of their respective shortest path.

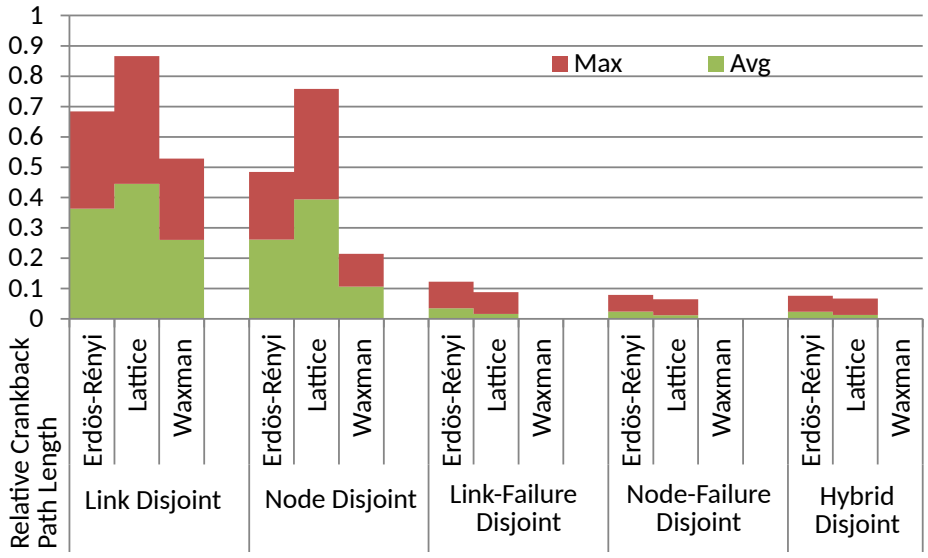
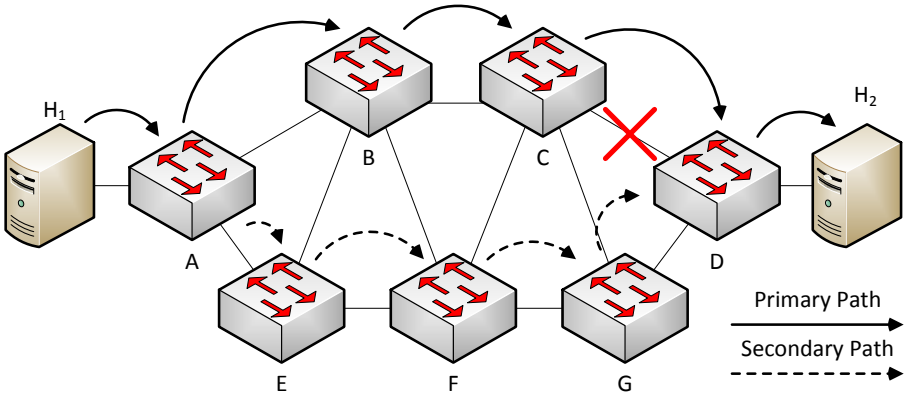


Figure 5.12: Comparison between average and maximal relative crankback lengths for networks of size $|N| = 100$.



(a) The disjoint paths from H_1 to H_2 do not instruct node C how to handle the link failure.

Figure 5.13: Disjoint paths and labels used in forwarding.

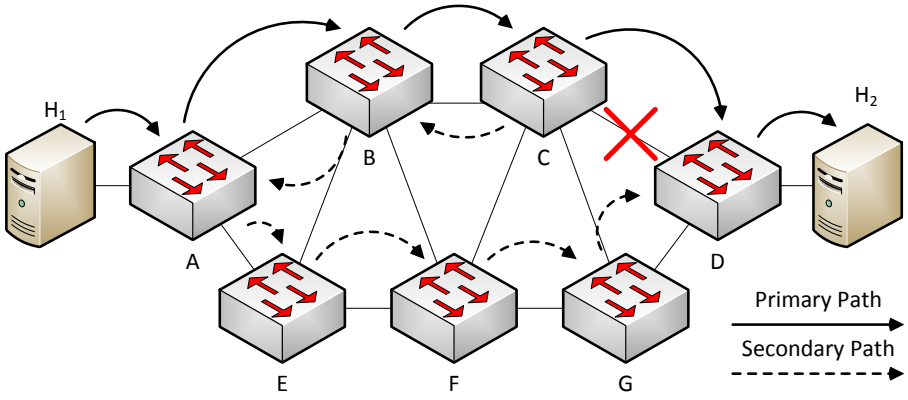
We have used the NetworkX package [130] to perform graph creation from the learned network topology and also used it to perform further graph manipulations and computations. We have extended the NetworkX package in the following ways:

- Cleaned up the shortest path algorithms
- Extended and standardized the (Queued) implementation of the Bellman-Ford shortest-paths algorithm
- Implemented Bhandari's disjoint-paths algorithm
- Implemented our failure-disjoint approach

The application configures the network according to our protection scheme, enabling it to circumvent link or node failures independent of (slow) controller intervention. After the controller is notified of an occurred failure, it reapplies the protection scheme to the new network topology, reestablishing protection from future topology failure where possible. Reconfiguration occurs without traffic interruption using a Flow entry update strategy as explained in [105]. Our additions to NetworkX are contributed to its source code repository. Our open-source OpenFlow controller is published on our GitHub webpage [131].

5.6. RELATED WORK

Disjoint paths, as depicted in figure 5.13a, are often used to preprogram alternative paths for when the primary path of a network connections breaks. A simple and intuitive approach for finding such disjoint paths is by using Dijkstra's algorithm [56] iteratively [120]. At each iteration, all of the links constituting the earlier $\{x\}_{1 \leq x < k}$ disjoint paths



(b) Using disjoint paths, the link failure of l_{cd} can only be omitted through crank back routing.

Figure 5.13: Disjoint paths and labels used in forwarding.

are removed from the network (temporarily) before Dijkstra's algorithm [56] is used for finding the k-th disjoint path. However, this iterative approach is but a heuristic and thus cannot always return the optimal solution even when it exists (e.g., in the presence of trap topologies [120]).

Suurballe [118] proposes an iterative scheme for finding k one-to-one disjoint paths. At each iteration, the network is (temporarily) transformed into an equivalent network such that the network has non-negative link weights and zero-weight links on the links of the shortest paths tree rooted at the source node. Dijkstra's algorithm can then be applied for finding the k-th disjoint path from the knowledge of the earlier $\{x\}_{1\leq x<k}$ disjoint paths. Bhandari [132] later proposed a simplification of Suurballe's algorithm by an iterative scheme for finding the k-th one-to-one disjoint path from the optimal solution of the $\{x\}_{1\leq x<k}$ disjoint paths. At each iteration, the direction and algebraic sign of the link weight is reversed for each link of the $\{x\}_{1\leq x<k}$ disjoint paths. The network can thus contain negative link weights. A modified Dijkstra's algorithm [132] or the Bellman-Ford algorithm [47][123], both usable in networks with negative link weights, can then be applied for finding the k-th disjoint path.

Both Suurballe's algorithm and Bhandari's algorithm need to be repeated $|N|(|N|-1)$ times for finding k disjoint paths between each possible node pair, since both algorithms return only the one-to-one directed min-sum disjoint paths between two given nodes. The Suurballe-Tarjan algorithm [119] has reduced worst-case time complexity for finding $k=2$ disjoint paths from one source to all possible node pairs. The Suurballe-Tarjan algorithm also uses the equivalent network transformation of the Suurballe algorithm to ensure that the network contains no negative link weights in each run of the Dijkstra's algorithm.

One of the disadvantages of using disjoint-paths based protection is that the traffic needs to be transmitted again from the source node using the backup path whenever the primary path fails. For example, figure 5.13a shows that even when node C detects the failure of link (C,D), node C has no means of rerouting the packets intended for node H_2 as it is not aware of the backup path. The only way to resolve this matter is to rely on crankback routing as depicted in figure 5.13b. Crankback routing, as may be evident from the picture, implies a high network overhead. On the other hand, our proposed algorithms enable traffic to be rerouted directly at the current node whenever its adjacent link or node fails, thus saving time and network resources.

There are also algorithms that propose protection schemes based on (un)directed disjoint trees, e.g., the Roskind-Tarjan algorithm [133] or the Medard-Finn-BarryGallager algorithm [134]. The Roskind-Tarjan algorithm finds k all-to-all undirected min-sum disjoint trees, while the Medard-Finn-Barry-Gallager algorithm finds a pair of one-to-all directed min-sum disjoint trees that can share links in the reverse direction. Contrary to our work, their resulting end-to-end paths can often be unnecessarily long, which may lead to higher failure probabilities and higher network operation costs.

A more extensive overview of disjoint paths algorithms is presented in [117], an overview of SDN-specific related work in topology recovery mechanisms is presented in chapter 4, a survey on disaster-resiliency in SDNs is given in[135].

In terms of work related to configuration computation Software-Defined Networks, Capone et al. [136] derive and compute an MILP formulation for preplanning recovery paths including QoS metrics. Their approach relies heavily on crankback routing, which results in long backup paths and redundant usage of links compared to our approach. Their follow-up work SPIDER [137] implements the respective failure rerouting mechanism using MPLS tags. The system relies heavily on OpenState [138] to perform customized failure detection and data plane switching, making it incompatible with existing networks and available hardware switches. Furthermore, the system does not distinguish between link and node failures as our approach does.

IBSDN [139] achieves robustness through running a centralized controller in parallel with a distributed routing protocol. Initially, all traffic is forwarded according to the controller's configuration. Switches revert to the path determined by the traditional routing protocol once a link is detected to be down. The authors omit crankback paths through crankback detection using a custom local monitoring agent. The proposed system is both elegant and simple, though does require customized hardware, since switches need to connect to a central controller, run a routing protocol, and implement a local agent to perform crankback detection. Moreover, the time it takes the routing protocol to converge to the post-failure situation may be long and cannot outpace a preconfigured backup plan.

Braun et al. [140] apply the concept of Loop-Free Alternates (LFA) from IP networks to SDNs, where nodes are preprogrammed with single-link backup rules when not creating loops. Through applying an alternative loop-detection method more backup paths are found than using traditional LFA, although full protection requires topological adaptations.

5.7. CONCLUSION

In this chapter we have derived, implemented and evaluated algorithms for computing an all-to-all network forwarding configuration capable of circumventing link and node failures. Our algorithms compute forwarding rules that include failure-disjoint backup paths offering preprogrammed protection from future topology failures. Through packet labelling we guarantee correct and loop-free detour forwarding. The labeling technique allows packets to return on primary paths unaffected by the failure and carries information used to upgrade link-failures to node-failures when applicable. Furthermore, we have implemented a proof-of-concept network controller that configures OpenFlow-based SDN switches according to this approach, showing that these types of failover techniques can be applied to production networks. Although implemented in OpenFlow, our method is applicable to all networks in which central controllers have an equally granular topology overview and the ability to match and add or rewrite protocol labels.

Compared to traditional link- or node-disjoint paths, our method shows to have significantly shorter primary and backup paths. Furthermore, we observe significantly less crankback routing when backup paths are activated in our approach. Besides shorter paths, our approach outperforms traditional disjoint path computations in terms of respectively the needed Flow and Group table configuration entries by factors up to 20 and 1.9. Our approach allows packets that encounter a broken link or node along their path, to travel the second-to-shortest path to their destination taken from the node where the link or node failure is detected. We apply Software-Defined Networking, specifically the OpenFlow protocol, to configure computer networks according to the derived protection scheme, allowing them to continue functioning without (slow) controller intervention. After the network controller is notified of the link or node failure it reconfigures the network by applying the protection scheme to its newly established topology, therewith reassuring protection from future topology failure within reasonable time. For future work, we suggest researching the protection of multi-link and -node failures as well as strictly guaranteeing QoS constraints under failure.

Table 5.1: Results for the evaluated algorithms and networks of network size $|N| = 100$.

Network	Disjoint	Flow Entries	Distinct Groups	Primary Path Ratio	Backup Path Ratio	- Min	- Max	Crankback Ratio	- Max
Erdös-Rényi	Link	106852.298	1908.580	1.014	2.146	1.419	2.787	0.363	0.684
Erdös-Rényi	Node	106632.844	1913.405	1.015	1.956	1.434	2.402	0.261	0.484
Erdös-Rényi	Link-Failure	10160.248	1187.016	1.000	1.512	1.218	1.887	0.035	0.123
Erdös-Rényi	Node-Failure	10149.929	1096.475	1.000	1.471	1.254	1.733	0.024	0.079
Erdös-Rényi	Hybrid-Failure	11451.396	1388.225	1.000	1.547	1.277	1.914	0.023	0.076
Lattice	Link	207585.132	1002.772	1.039	2.259	1.369	3.101	0.445	0.866
Lattice	Node	207642.592	1006.752	1.050	2.190	1.403	2.919	0.394	0.758
Lattice	Link-Failure	10381.567	571.113	1.000	1.283	1.095	1.525	0.016	0.088
Lattice	Node-Failure	10663.192	542.702	1.000	1.308	1.127	1.538	0.012	0.065
Lattice	Hybrid-Failure	12320.495	735.551	1.000	1.331	1.131	1.612	0.013	0.067
Waxman	Link	50885.088	6208.709	1.000	1.570	1.049	2.105	0.260	0.528
Waxman	Node	50572.606	6247.912	1.000	1.359	1.147	1.576	0.106	0.214
Waxman	Link-Failure	9900	3874.098	1.000	1.070	1.032	1.117	0.000	0.001
Waxman	Node-Failure	9900	3268.110	1.000	1.152	1.139	1.166	0.000	0.000
Waxman	Hybrid-Failure	13671.039	4660.458	1.000	1.163	1.147	1.180	0.000	0.000

6

GLOBALLY-ACCESIBLE NAMES IN NAMED DATA NETWORKING

The previous three chapters presented improvements to the field of Software-Defined Networking (SDN); in this chapter we take a step aside from SDN and propose improvements to the field of Information-Centric Networking (ICN). In chapter 7, we will couple the two fields of ICN and SDN by showing one can implement the first through the latter. Information-Centric Networking (ICN) paradigms aim at optimizing computer networks for information distribution. Named Data Networking (NDN) and its implementation CCNx propose a promising globally implementable ICN. Routing on names, however, may result in extremely large global routing tables. In this chapter, we propose to confine the global routing table size by decoupling context-related names, such as domain names, from names routable within the network. By aggregating routable names to their topological location, the size of global routing tables decreases to the number of Autonomous Systems. Furthermore, mapping context-related names back to location-aggregated names using a directory service eases the process of sharing information on the ICN. The robustness of the network is further increased by employing dynamic multihoming without changing application names. Additionally, we include a prefix-delegation method that allows Local Area Networks to be automatically configured with such location-aggregated globally routable names.

This chapter is based on a published paper [141].

6.1. INTRODUCTION

Host-to-host networks – such as IP networks – are designed to efficiently forward packets from one host to another over large networks, such as the Internet. However, practice shows that host-to-host networks are inefficient at distributing content on a large scale, since data will often duplicately travel along the same links and nodes. Optimizing for information distribution in IP is difficult due to applications explicitly requesting the network to deliver packets, therefore only enabling optimizations for host-to-host packet delivery. Lately, much work [142][143][12][11] has been done in creating Information-Centric Networks (ICNs) that optimize for information distribution instead, by making the networks more aware of their function as information distributor.

Named Data Networking (NDN) [9] implements an Information-Centric Network by forwarding packets based on the names of the content they request or carry, instead of the addresses of their requester and generator. By basing the packet forwarding across the network on the content name or description, we allow the network to be optimized for distribution of content. The most straightforward optimization is obtained by delivering duplicate requests from caches placed within the shortest paths from client to server. Duplicate requests can be detected by comparing the names of requested content to earlier delivered data and providing the results from memory accordingly.

NDN proposes a promising globally implementable Information-Centric Networking technique:

- It is independent of underlying – if any – transport protocols.
- It deploys a clear three-table prefix match and forwarding mechanism, as discussed in section 6.2.
- It offers new opportunities to Quality of Service and exploitation of multipath routes.

Unfortunately, NDN also knows problems that need to be solved before a global implementation becomes feasible:

(1) As described in [143], we need to contain the global routing table size resulting from name-based forwarding. Currently, over 240 million domain names[1] [144], versus more than 42.000 ASes and 438.000 IP subnets propagated across BGP [145], exist. Given that the size of the BGP routing tables is already problematic [146], an increase of a factor 550 or more introduced by name-based routing cannot be solved without further aggregation of names.

(2) Another problem is the difficulty for end-users to share information on the network. Currently, sharing information requires configuring the NDN-enabled topology-discovery mechanism OSPFN [147]. We need a mechanism enabling end-users to dynamically access and share information on the ICN without the need to configure topology-discovery mechanisms.

Where previous research mainly focused on speeding up lookups from the routing tables using hash tables [148][149], we aim to reduce the size of the problem. We will first

[1] Excluding the existence of NDN-routable subdomains and subnames.

describe the basic techniques of NDN in section 6.2, followed by our proposal of decoupling context-related names from globally routable names in section 6.3. In section 6.4, we discuss the implications on naming and topology discovery in the different routing domains[2]. Section 6.5 describes our proposed mapping technique and shows that our implementation realizes such a mapping mechanism for NDN, while maintaining the NDN features. In section 6.6 we present our implementation, which dynamically generates aggregated names within LANs. Finally, section 6.7 discusses previous research in this area.

6.2. NAMED DATA NETWORKING

In this section, we will briefly discuss how Named Data Networking (NDN) [9] and its implementation CCNx form an Information-Centric Network (ICN) by using a route-by-name principle. Opposed to source and destination addresses in IP packets, NDN knows Interest and ContentObject packets identified by names composed of one or more name components (e.g., Alice could share her photos using the name `/alice.eu/photo`). Whenever a client requests information, it sends out an Interest containing a name describing the desired information. Subsequent nodes forward that Interest hop-by-hop along the shortest path to the generator responsible for that name until it either reaches a node that has a cached copy satisfying the description of the Interest, or it reaches the content generator. The resulting data is encapsulated in a ContentObject and is forwarded along the exact reverse path back to the requester.

Functionally, each subsequent node maintains three tables:

1. The ContentStore (CS), containing cached copies of previously delivered data.
2. The Pending Interest Table (PIT), in which forwarded Interests and their originating faces[3] are stored.
3. The Forwarding Information Base (FIB), containing forwarding rules based on names instead of addresses.

Each incoming Interest is first compared to the content of the CS to determine if it can be fulfilled from cache. If not, the packet is compared to the PIT to prevent duplicate requests traveling down the network. Finally, the FIB is consulted to determine to which faces the Interest needs to be forwarded. The Interest and its originating face will be added to the PIT accordingly. Once an Interest hits a valid copy or a new ContentObject is generated, the PIT entries are used to forward the resulting ContentObject back to its original requester.

While IP prefix matches on subsequent bits, NDN prefix matches on subsequent name components. Each subsequent component adds a more restrictive requirement to the content (e.g., the name `/alice.eu/photo/holidayPhoto` prefix matches an Interest for `/alice.eu/photo`, as the requester did not specify the exact photo). Additional filters regarding the queried content can be added through so-called selectors, including minimal and maximal number of name components, exclude filters, publisher

[2] Inter-domain, intra-domain and local area networks.

[3] The NDN equivalent of an interface. Besides Ethernet links a face can also include connections with applications and several other types of connections.

public keys and child selectors (which are especially useful to select for example the first sequence of frames of a movie or the newest or oldest element from the tree of content).

Ultimately, the queried content is not produced and sent by the publisher at each and every Interest packet, but delivered from caches and peers scattered through the network. To verify authenticity of the received information, each ContentObject is cryptographically signed by its publisher who can be identified through a DNS-like first name component.

6.3. RECURSIVE NAME AGGREGATION

In order to keep the complexity of global routing tables within limits, we will decouple routable names from user-registered context-related names as suggested in [143]. Context-related names are mapped to one or more routable names describing the locations of the hosts responsible for generating the corresponding content. In this section, we will propose a structure for the routable names in different routing domains.

(1) Inter-domain routing should be based on single NameComponents describing the AS to which the generating host belongs. This bounds the size of global routing tables to the number of ASes in the network[4]. For example, global routing may occur using names such as `/isp-name.net` for providers and `/company-name` for enterprise companies maintaining their own AS.

(2) By using subnames of the AS-specific name for nodes within an AS network, the AS only needs to set up node-specific routes internally without affecting global routing tables. For example, a server within an AS network may be named `/isp.net/server1`, which becomes globally accessible since the prefix `/isp.net` points towards the AS and the AS will route incoming Interests to the specific node internally. In-network routing tables will be as complex as they would be in IP networks, as the process of subnaming conforms to the process of IP subnetting.

(3) Local network nodes may use names aggregating to the name of their modem or access router as provided by their ISP, which in turn can be used for further subnaming without affecting the AS its routing tables.

The advantage of a topologically layered naming scheme is the fact that subnets can be dynamically generated and globally used without affecting the routing tables of the higher-level networks. NDN advantages – such as caching, multipath routing and the Strategy layer – are preserved since forwarding is still done based on globally unique hierarchical names. In section 6.4, we will discuss the implications in terms of naming and routing discovery for the identified routing domains.

Although the global name complexity can be contained, users prefer to work with context-related names. In IP, applications use the Domain Name System (DNS) [150] to translate the authoritative part of a (generally context-related) URL to a location in the form of an IP address. We, however, propose to use a mapping system similar to the Location Identifier Separation Protocol (LISP) [151] to have the network map context-related names to location-based names. Applications base communication and content retrieval on the context-related names, while networking nodes (such as the host CCNx daemon or the default gateway) rename Interests and ContentObjects from the context-

[4] Which, according to [145], is a magnitude of 10 smaller than the current number of propagated IP prefixes.

related names to the location-based names and vice versa.

By mapping context-related names to routable names in the network, application communication takes place independent of the location, the number and the availability of end-points; in NDN one does not care who delivers the information nor the path it has taken as long as the delivered information is guaranteed to be authentic.

6.4. NAMING AND DISCOVERY

In this section, we will discuss the naming of entities and how to find them by means of topology discovery.

6.4.1. INTER-DOMAIN ROUTING

We propose to use AS-bound context-unrelated first name components for inter-domain routing. The first NameComponent can be either descriptive for the AS (e.g., `/isp.as`), contain the Autonomous System Number (e.g., `/AS1103`) or a variable-length byte-encoded integer containing the AS Number (as prefix-matching on integers is a less expensive operation than on strings). If all inter-domain routing is based on these first name components, the size of global routing tables is upper bounded by the number of ASes present on the Internet.

In order for ASes to forward and serve NDN traffic, we need to adapt the inter-domain topology-discovery mechanism BGP [152] to propagate NDN compatibility and name information. Since the standardization of Multiprotocol Extensions for BGP-4 [153][154], it is possible to propagate the availability of address families other than IPv4 and IPv6 in BGP. Narayanan et al. [155] are working on realizing such an implementation by introducing a new address family supporting NDN names, although they presently do not consider crossing NDN-incompatible ASes.

De Clercq et al., in RfC 4798 [156], discuss a standard in which IPv6 is tunneled dynamically across (IPv6 incapable) IPv4/MPLS networks. NDN needs a mechanism with properties from both Narayanan et al. and RfC 4798 to realize connectivity across NDN-incompatible ASes by using dynamically set-up tunnels. Where NDN however does not know IP-mapped names, we need to find other ways to insert tunnel end-points into the propagated names[5].

6.4.2. INTRA-DOMAIN ROUTING

Nodes within an Autonomous System can, in the location-aggregated naming scheme, be named at will as long as their name prefix matches the previously discussed AS-bound name that is globally propagated. Names consisting of concatenating the AS-bound name and an internally unique router identification, such as `/isp.net/routerX`, qualify. If an AS knows different layers in which subnetworks are connected it may represent those in the names of the nodes.

Internal routing discovery needs to occur in order to propagate the availability of names internally. Popular intra-domain routing discovery mechanisms, such as OSPF

[5] For example by using a directory service or by assigning AS-specific IP addresses that each AS should run tunnel end-points on.

and IS-IS, need to be adapted to support the new address family introduced by the hierarchical naming structure of NDN.

OSPFN [147] is an implementation of such an adaptation using OSPF to propagate available NDN names in a network. By using OSPF's Opaque LSA options, the developers of OSPFN have designed a conveniently accessible mechanism taking advantage of an already existing infrastructure. However, the OSPFN daemon reuses the shortest paths computed by OSPF on the local Routing Information Base without considering whether each node within the network is NDN compatible.

We propose a mechanism similar to OSPFN that uses Opaque LSAs to spread NDN compatibility and names. When considering shortest paths between NDN-compatible nodes, parts of the path that are NDN-incompatible should either be omitted or crossed using IP-encapsulated tunnels.

6.4.3. LOCAL AREA NETWORK

We propose to base naming and routing of Local Area Networks (LANs) on the subnames of the access router (generally a cable- or DSL-modem) by which they are connected to their ISP. The ISP generates the name of the entrypoint and solves routing in its network, while users can use subnames aggregating to that name to share information from within the LAN.

Similar to IP's Dynamic Host Configuration Protocol, we need mechanisms to dynamically configure end-user devices and enable them to dynamically retrieve and share information on the NDN. CCNx-DHCP [157] already fulfils part of that duty by introducing a server-client model that preconfigures many clients on a network. Unfortunately, CCNx-DHCP does not enable end-users to also share information on the network. Currently, a user must propagate his name using OSPFN [147] into the network before it properly forwards Interests meant for him, which is undesirable. Furthermore, the local area network topology discovery usually lies out of the propagation scope of the AS, which prevents global propagation of the name.

In order to have users fully participate in the NDN[6], we propose to deploy a configuration-free routing discovery mechanism to find the border routers within the LAN connecting it to the ISP network. The shortest paths to those entrypoints are used as gateway forwarding rules, while the reverse path is used to form names aggregating back to the names of the entrypoints. This enables (both consumer and enterprise) Local Area Networks to form dynamically without the effort to administrate address ranges and routing tables. Section 6.6 discusses our proposal and its implementation for recursive name aggregation in LANs.

6.5. MAPPING

An important aspect of our proposal is how mappings between context-related and location-aggregated names are stored and retrieved. Given the fact that the mapping or translation of the names needs to occur on-the-fly, a very low time complexity is necessary for a successful implementation.

We propose to use the Domain Name System (DNS) to store mappings between

[6] Meaning that they can both *access* and *share* content dynamically.

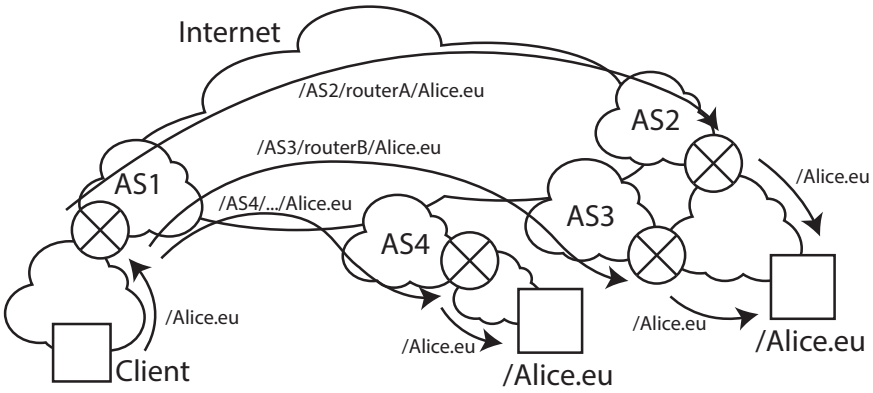


Figure 6.1: Topology containing context-related routable networks connected over a globally routable network using mapping. Inter-domain forwarding occurs based on the first NameComponent, intra-domain on the second, while the LAN networks forward based on the context-related names.

context-related and location-aggregated names. Although DNS's worst-case time complexity is not guaranteed, [158] shows an average response time of 43 ms for conventional DNS compared to 350 ms response time for a peer-to-peer distributed DNS. Throughout the years, DNS has been used to map domain names to – among others – IP addresses. Its low response time is supported by the design that each queried DNS server is authoritative for at least one extra level of the requested name. In general, 3 recursive requests are sufficient to obtain an authoritative answer. We propose to either extend DNS by adding a new type of record to store NDN mappings, or reuse one of the generalized service (SRV) or text (TXT) descriptions.

Where IP clients use the address retrieved by DNS to base their networking connections on, we propose a slightly different solution based on LISP-TREE [159] and its equivalent LISP-DDT [160], both implementations of the Locator/ID Separation Protocol (LISP) [161]. In LISP, a subset of the IP address space is used to identify hosts disregarding their physical location or local network. Subsequently, Ingress Tunnel Routers (ITRs) are placed within the physical local networks and receive, encapsulate and forward packets intended for hosts outside of the immediate broadcast range, looking up their physical location in a (distributed) mapping database. Egress Tunnel Routers (ETRs) receive those packets, decapsulate them and forward them to the local destination. Effectively, LISP offers a system where host-related address are decoupled from the traditional location-based addresses.

Similarly, in our proposal applications base their communication on the context-related names, while networking instances on either the hosts or gateways near them translate the Interests to their location-aggregated names[7]. The renamed Interests are forwarded accordingly to a host in the network of the content generator who can translate the name back to its original value. Figure 6.1 shows a topology in which a local network maps the desired context-related name to three globally reachable end-points that forward the Interests to the responsible nodes.

[7] Whose values are fetched using DNS.

6.5.1. LOCATION INDEPENDENCE AND DETECTING BROKEN PATHS

By basing the communication protocol between applications on context-related names, one allows the network to find another path to the content serving the application[8] independent of its location and without interrupting the application data flow.

While IP routing discovery selects the best of all paths to each destination and IP kernels merely forward packets along these paths, NDN multipath routes may coexist and Interests can be duplicated and sent along multiple paths. NDN introduces a strategy layer in which the performance of these different routes are evaluated. Based on this evaluation the forwarding engine can decide to send a larger portion of the Interests necessary to retrieve the complete file to the route behaving best (in terms of latency, throughput and packet loss), while maintaining the occasional Interest to monitor the state of the other routes.

Using our renaming mechanism implementation discussed in subsection 6.5.2, we enable the strategy layer to balance between the available location-aggregated names – and thus paths – for any context-related name. Using the strategy layer, broken paths are detected and omitted without changing the mapping[9] or recalculating the network topology.

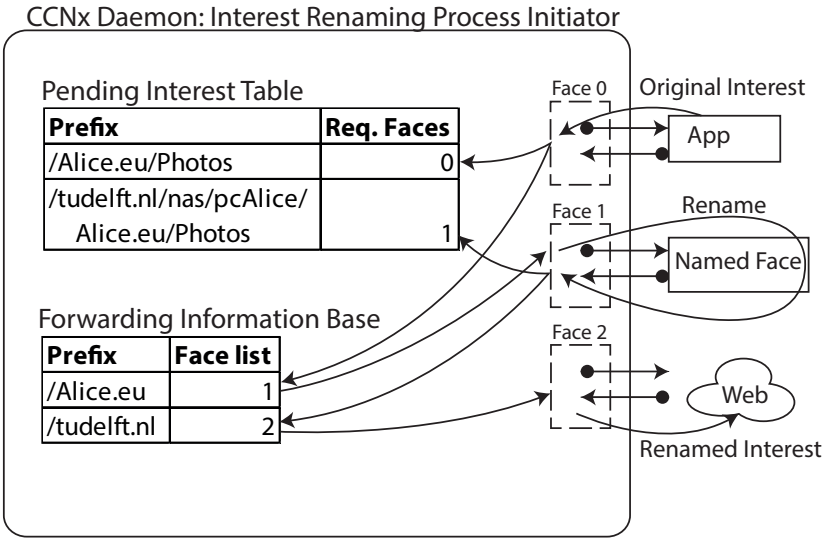
6.5.2. CCNX RENAMING IMPLEMENTATION

In this section, we will discuss the details of implementing the described renaming from context-related names to location-aggregated names in CCNx. As shown in figure 6.2a, we add a new type of face – the named face – which renames Interests containing context-related names by appending the location-aggregated name of one of the receiving end-points to the front of the requested Name. At the receiving node depicted in figure 6.2b, a similar face removes the location-specific part of the name and forwards the Interest to the appropriate application or next-hop. As all intermediate names are stored in the Pending Interest Table with their respective sources, we assure that the returned ContentObject travels the renaming steps in reverse order.

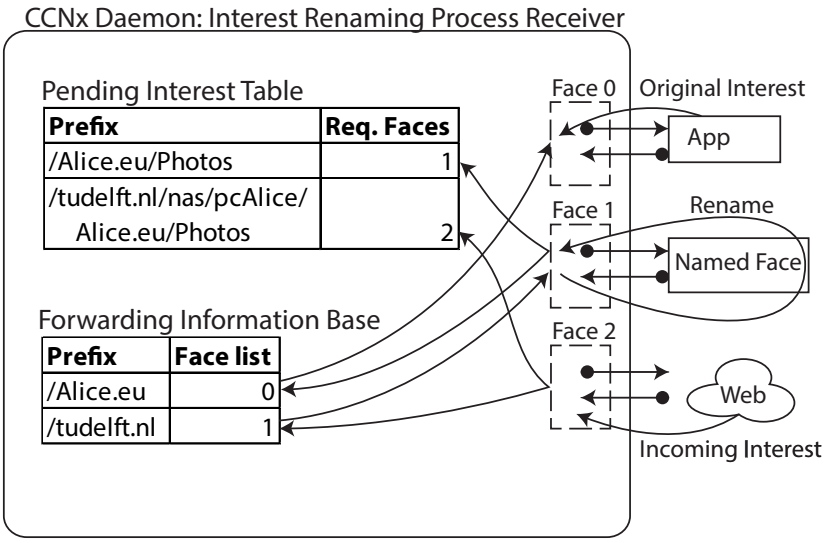
Since each mapping from a context-related name to a location-aggregated name is expressed in the form of a distinct unique face and a route, multiple faces exist when multiple mappings are available resulting in a forwarding rule for each mapping. The CCNx strategy layer chooses the best of these routes – and thus mappings – over time and reverts to the other route when the path of the primary route breaks. This failover procedure is shown by experiment in [9]. Our freely available implementation [162] offers a clean and simple mechanism to insert mappings from context-free to location-aggregated names and back into an NDN. Once an Interest arrives for an unknown context-related name, the implementation queries the DNS system, which stores the mappings using TXT-records and creates the renaming faces accordingly. The newly created named faces are used to rename Interests until they time out due to inactivity or outlive the TTL stated in DNS.

[8] By selecting another location-aggregated name by which the serving host can be connected.

[9] If one or more location-aggregated names for a context-related name do not respond, the strategy layer will simply disregard those.



(a) Interest Renaming Initiator.



(b) Interest Renaming Receiver.

Figure 6.2: The Interest renaming mechanism. After a ContentStore mismatch, context-related Interests are forwarded to the appropriate Named Face which re-expresses the Interests with the corresponding location-aggregated names.

6.5.3. CONTENTOBJECT SIGNATURE VALIDATION

By renaming the ContentObject (CO) resulting from a successfully delivered Interest, one invalidates its signature as the signature is partially based on the name. Although the CO regains its validity when it is renamed to the original name at the initiating node, any node along the path may check its validity and drop a CO if it is invalid [163].

In order to prevent unjustified dropped COs two possible solutions exist. Currently, we encapsulate the original (signed) CO in a new CO with the renamed name and sign accordingly. Using this method the traveling ContentObject lives through signature checks at any point. The encapsulation of a CO in another CO however renders overhead. A solution containing less data overhead is to remove signature validation across the path and make the named faces and connected applications responsible for validation of the received ContentObjects. To support this feature, we propose to add a new type of flag to the Interest messages indicating whether the resulting ContentObject's signature may or may not be validated during transport.

6.5.4. CCNX DNS IMPLEMENTATION

Since a mapping stored in DNS is a form of content itself, we can use the NDN to efficiently distribute the mappings among interested parties. In order to make DNS suitable for NDN, we have defined an application specific naming-scheme which describes the DNS request. The name contains

1. the authority hosting the DNS server: in order to forward the packets to the right party,
2. the name one wishes to query: the first NameComponent of the context-related name (e.g. Alice.eu), and
3. (optionally) the type of record one needs (e.g. A-, AAAA-, MX- or NS-records).

The NDN-record for the domain Alice.eu can be obtained by querying a local (recursive) DNS-server using the name `/server.local/dns/Alice.eu/NDN` and receiving a result from that server over the NDN. Using our CCNx DNS client and server extensions [164] one can extend existing DNS servers and clients with NDN capabilities. Once DNS root and Top Level Domain servers are also configured to support NDN, the NDN can be used to query all recursion steps.

6.5.5. EXPERIMENTAL MEASUREMENTS

In order to verify the functionality of our renaming scheme, we have set up a small network containing 4 nodes connected in a straight line giving a diameter of 3 edges. In this network, we have set up experiments measuring the round-trip-time (RTT) between the end-hosts using 3600 consequent echo requests, submitting 1 echo request each second, and their subsequent replies. We have used *ccnping* [165] to confirm RTTs over the plainly routable and renaming NDN network. Although the network is rather small, the largest additional delay is generated by the two renaming nodes independent of the size of the network.

Figure 6.3 shows the RTTs for forwarding the packets using (1) a plain CCNx network, (2) a renaming CCNx network not considering signature validity and (3) a renam-

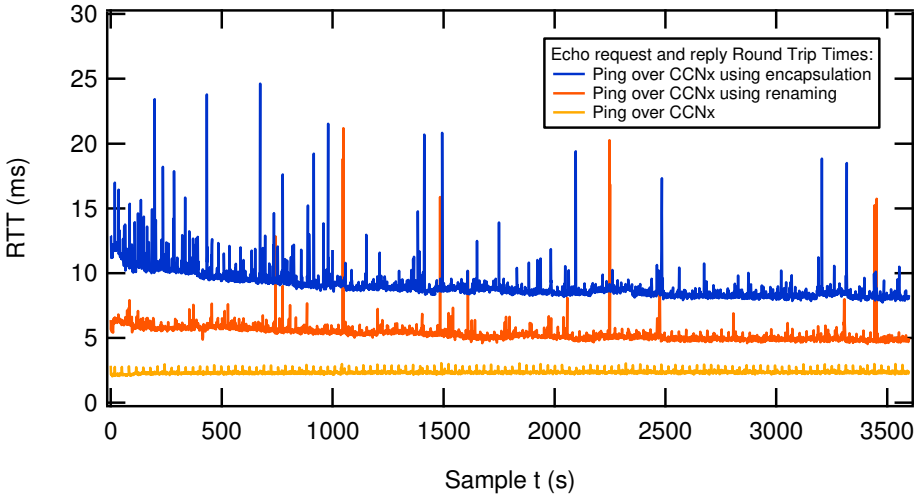


Figure 6.3: Measured round-trip-times using Ping echo requests and replies over CCNx and renaming CCNx networks.

ing CCNx network that guarantees valid signatures by encapsulating signed ContentObjects into new. The RTT averages of respectively 2.338, 5.327 and 9.031 ms show that the delay overhead by applying renaming in a network equals 2.989 ms while encapsulating and signing packets adds another 3.704 ms.

We consider the average delay of 2.989 ms to be a feasible penalty compared to the recurring profits of the decreased routing table sizes. Furthermore, we expect future optimizations [148][149] on comparing content names and FIB lookups to further decrease this delay. Finally, the additional transmission delay introduced by encapsulating packets supports our proposal from subsection 6.5.3 to disable intermediate signature validation for renamed packets.

6.6. LOCAL AREA NETWORK IMPLEMENTATION

In this section, we will discuss the details of our LAN prototype implementation [166] using recursive name aggregation. As discussed in section 6.4, we propose to use the names of the nodes connecting a LAN to its ISP and use subnames of these names to base naming on within the network. This enables LANs and their names to form dynamically without the administration of reserving ranges and setting up forwarding manually. The shortest paths to all edge routers can be used as default forwarding rules between which the NDN strategy layer employs heuristics to benefit from the multipath routing rules. The reverse path vector of hostnames, added to the entrypoint's base name, dynamically generates a globally unique routable name for each node in the LAN as shown in figure 6.4. Where intra- and inter-AS routing delivers Interests meant for the LAN to the LAN edge routers, the routers within the LAN set up correct rules to forward Interests internally.

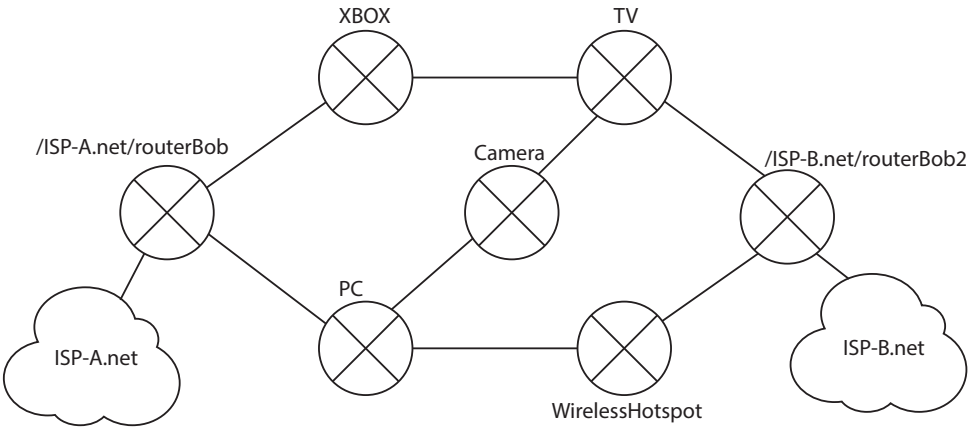


Figure 6.4: An example LAN, in which the photos on the camera can be shared by both the names `/ISP-A.net/routerBob/PC/Camera` and `/ISP-B.net/routerBob2/TV/Camera`. By following the shortest paths from the camera to the entrypoints `/ISP-A.net/routerBob` and `/ISP-B.net/routerBob2` and appending the reverse path vector of hostnames to the entrypoint names, the globally unique location-aggregated names are generated.

6.6.1. PROPAGATION

In order to fulfill the propagation of available routes and names, our prototype routing discovery application [166] employs a path vector routing mechanism exchanging messages containing the following properties for each possible route:

- The originating LAN edge router, called entrypoint, to identify multiple entrypoints and multipath routes.
- The route one can offer: either a gateway route, a specific route or the aggregation base name (to ensure one can reach all nodes within its own LAN).
- The cumulative cost of the offered route.
- The path vector to the entrypoint, which can be used for both naming and loop detection.
- A Boolean indicating whether aggregation upon the offered route is permitted.

When a node enters the network, it broadcasts a discovery message on all its interfaces indicating that it is new and needs to receive all possible routes neighboring nodes are willing to forward. It receives a set of routes containing the previously discussed properties from each neighboring node. From the received sets, the new node selects the optimal subset of routes it will use based on cost and path vector, including multipath routes that are identified based on the originating entrypoint. The node requests the usage of the selected routes from the neighbors in question, which will set up the necessary forwarding and acknowledge thereafter.

At this point the new peer has established its state within the network and may offer forwarding and aggregated names to even newer peers recursively. The end-user can start sharing content globally using the received location-aggregated names and may register the names at a mapping service as discussed in section 6.5 to publish information using context-related names.

6.7. RELATED WORK

The idea of name aggregation was first proposed in a technical report [143], where it is referred to as ISP-based aggregation. In this chapter we have further enhanced this aggregation by applying a recursive extension of names when one descends into the network, as well as proposing dynamic generation of those names in Local Area Networks. Where the report proposes to have the PC's themselves map "pretty" names into ISP-based names in an early phase, we propose to use a late binding of identifiers as implemented in the Locator/Identifier Separation Protocol [151].

The NDN name conventions [167] only state that names have to be hierarchical in nature. Stricter conventions are solely enforced by the applications or institutions using NDN. Only in the case of globally reachable names, a name should be globally unique and its first name component should be a "DNS name". Further name conventions regard markers such as version timestamps and segment numbers which do not influence routing as they are appended to the end of a name.

Since our proposal contains first name components that can be registered for both context-related as location-aggregated names[10], both can be inserted into the DNS system and used accordingly. Furthermore, [9] and [167] state that the DNS may be used for looking up IP tunnel endpoints, which can be used to bridge NDN-incompatible nodes and ASes.

Finally, recent research has been done in the area of speeding up NDN table lookups [148][149].

6.8. CONCLUSION

In this chapter, we have proposed to decouple context-related NDN names from routable names. By using routable names aggregating to Autonomous Systems and their internal structure, the size of global routing tables becomes upper bounded by the number of ASes in a system. By using a LISP-style late binding of context-related names to location-aggregated names using the DNS directory service, we enable applications to communicate based on context-related names. Our implementation of mapping and renaming context describing names to routable names, enables the NDN strategy layer to facilitate multipath routes and multihomed applications.

[10] Both domain names and ASNs already need to be registered at the appropriate administration organizations.

7

NDNFlow: Software-Defined Information-Centric Networking

The previous chapter presented the field of Information-Centric Networking (ICN) and focused on the Named Data Networking (NDN) paradigm. In this chapter, we couple ICN to the field of Software-Defined Networking (SDN). The biggest problem with new network-layer forwarding schemes, such as CCNx, is that the use of IP is so prevalent that it is difficult to actually implement new technologies in existing networks. The biggest problem in upgrading a network to support new forwarding mechanisms is that all networking devices must be upgraded or replaced before full connectivity can take place. However, upgrading all network devices at once is expensive and unlikely to occur at any ISP. Evidence of this can be found in the long transition period from IPv4 to IPv6 and its many, many transition mechanisms delivering intermediate connectivity in not-fully upgraded networks, but also adding even more configuration complexity.

In this chapter we show that ICNs, and other non-IP forwarding schemes for that matter, can be deployed using SDN as a lever to ease the installation, configuration and transition process in a fundamentally sound way. As a proof of concept, we introduce NDNFlow, an open-source software implementation of a Named Data Networking based forwarding scheme in OpenFlow-controlled SDNs. By setting up an application-specific communication channel and controller layer parallel to the application-agnostic OpenFlow protocol, we obtain a mechanism to deploy specific optimizations into a network without requiring a full network upgrade or OpenFlow protocol change. Our open-source software implementation consists of both an NDN-specific controller module and an NDN client plug-in. NDNFlow allows OpenFlow networks with NDN capabilities to exploit the benefits of NDN, enabling the use of intermediate caches, identifying flows of content and eventually performing traffic engineering based on these principles.

This chapter is based on a published paper [168].

7.1. INTRODUCTION

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

OpenFlow [68] is often considered to be the de facto standard to implement SDN. Other emerging future internet architectures, such as Information-Centric Networking (ICN), introduce application-specific forwarding schemes. In particular, we believe that SDN and ICN can benefit from each other. SDNs can benefit from the power of caching from ICNs and in general need to be able to quickly adapt to new application-specific forwarding schemes, such as ICNs. ICNs, on the other hand, greatly benefit if they can be adopted with little effort by already existing SDNs. Furthermore, the benefits of SDN to IP, being greater management control and monitoring over the network, also apply to ICNs. Finally, ICNs benefit from SDNs as they can efficiently distribute content in partially upgraded networks, removing the necessity to upgrade the full network and thus easing the deployment and transition phase.

In this chapter, we discuss our experiences in setting up an SDN for the application-specific forwarding mechanism of Named Data Networking (NDN) [9], a popular ICN implementation. Although this chapter is dedicated to setting up an SDN-supported ICN, our experiences and decisions also apply to other forwarding mechanisms that may emerge.

In section 6.2 of the previous chapter, we have already explained the functionality of the ICN implementation NDN. In section 7.2, we present related work and our two initial proposals towards application-specific SDNs and reasons why we think these approaches are infeasible for standardization. Section 7.3 proposes our mechanism in which we have divided the SDN in two layers: the regular OpenFlow layer based on traditional forwarding mechanisms, supplemented with an application-specific layer. Section 7.4 presents the exact details of our implementation. Section 7.5 presents experimental measurements performed on NDNFlow. Finally, section 7.6 concludes this chapter.

7.2. RELATED WORK

In order to use SDN and OpenFlow to set up ICN networks, we have evaluated multiple techniques before coming to our final proposal in section 7.3. In this section, we discuss previous initiatives and parts of our early work and argue why we think these are not feasible for standardization.

A straightforward way to implement ICN using SDNs is to implement ICN functionality into the Open vSwitch specification and enhance the OpenFlow protocol to support ICN names, as suggested in [169]. We, however, think the joint maturing of both ICN protocols and the OpenFlow protocol will increase the complexity of realizing a stable standardization of OpenFlow that supports both regular IP/Ethernet forwarding and ICN. As [141] shows, the concept of naming in NDN is, among others, still subject to further optimization to decrease routing table size and thereby increase the forwarding efficiency.

Given that the standardization of OpenFlow for regular packet forwarding is already a complex task, chances that standardization will include application-specific forwarding schemes are small.

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

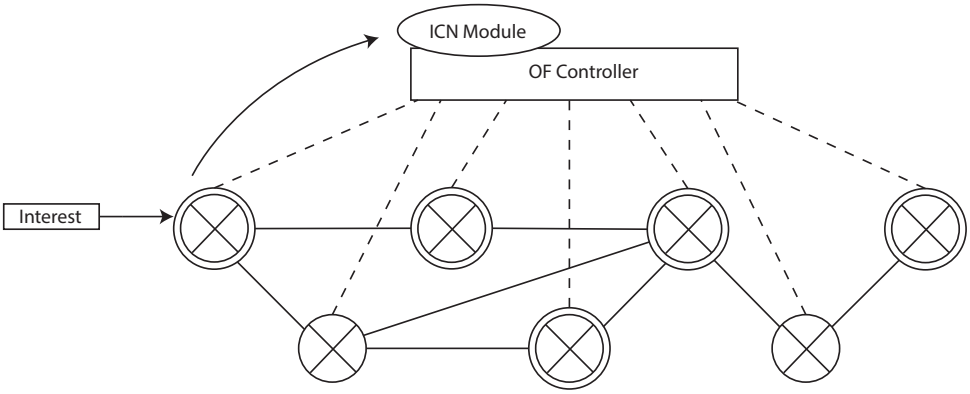
Both [170] and [171] propose to reuse IP's address and port fields to contain hashes of content names in order to allow OpenFlow switches to forward Interests to an OpenFlow controller that performs path calculation. Where [170] uses additional IP-options to indicate ICN packets, [171] remains agnostic to how ICN packets are distinguished from regular IP packets. We argue that this approach leads to an excessive increase in IP routing table complexity.

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

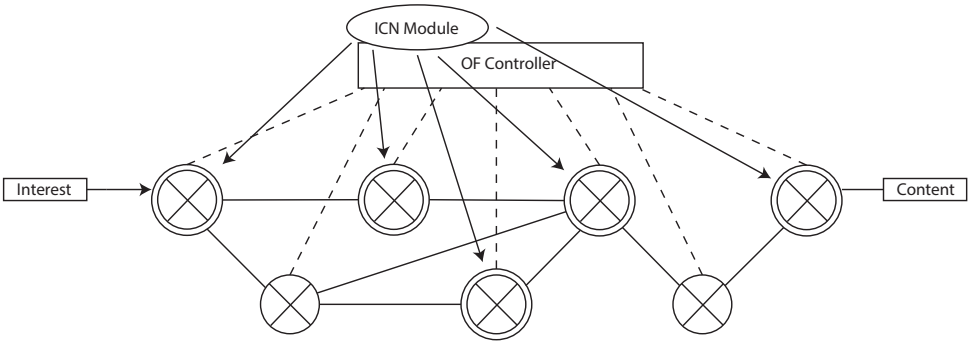
Finally, P4 [172] and POF [173] respectively implement a packet processor description language and forwarding architecture design to allow protocol-oblivious forwarding, work resulting in the ONF OF-PI proposal [174]. However, P4 implements static field sizes, rendering it unsuitable for use with the CCNx implementation, which carries a variable number of variable-sized name components. Furthermore, while using $\{offset, length\}$ search keys as proposed in POF may work, we consider the complexity of rewriting all abstract comparisons and functions to bit-wise operators too tedious.

7.3. NDNFLOW

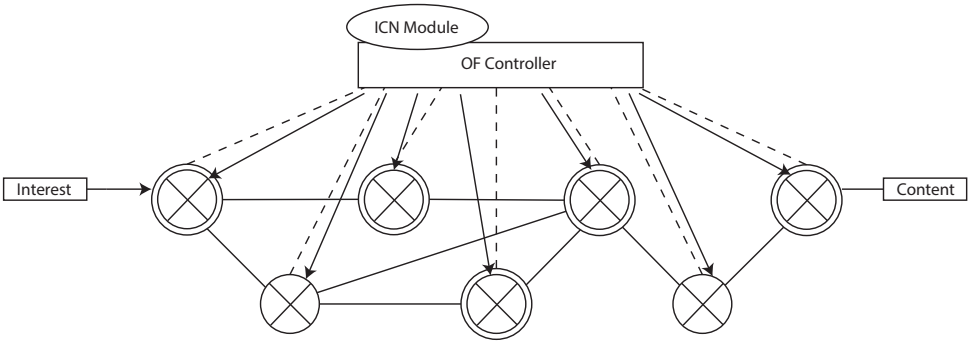
Given that neither extending the OpenFlow protocol for application-specific forwarding schemes nor using application-specific IP broadcast addresses to distinguish ICN traffic from regular IP traffic is feasible, NDNFlow introduces a second application-specific layer to OpenFlow. NDNFlow implements a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



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



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



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

Figure 7.1: An overview of the steps necessary to configure an ICN flow over an OpenFlow-enabled network. All switches are OpenFlow capable, doubly-circled nodes additionally have ICN capabilities.

In this application-specific layer, all communication and path computation regarding the ICN are handled separately from the regular IP and Ethernet plane. By separating the ICN layer from the regular OpenFlow layer, we introduce SDN functionality independent of changes and restrictions in the standardization of the OpenFlow protocol. This prevents inter-dependencies on versions of protocols, easing the deployment and future maintenance. As shown in figure 7.1, switches that are ICN enabled set up a communication channel, parallel to their regular OpenFlow communication channel, to the ICN module of the OpenFlow controller. This ICN channel is then used to announce ICN capabilities, information availability and requests for unreserved flows. The controller's ICN module computes paths for ICN flows, and configures both the ICN capable and legacy IP and Ethernet switching fabric to allow ICN flows to pass through the network. Hence, we introduce a separate SDN control mechanism for the application-specific OSI Layers 5 to 7 (ICN), independent from OSI Layers 2 to 4, reusing the separation of layer responsibilities to maintain overall network manageability.

Where ICN-enabled switches receive ICN-specific flows directly, flows between ICN-enabled switches that are initially unreachable due to intermediate legacy IP and Ethernet switches are realized by setting up IP-encapsulated tunnels. The legacy switches are configured by the legacy OpenFlow controller to forward those tunnels accordingly. Hence, the configuration procedure consists of 3 steps shown in figure 7.1, where a doubly-circled node represents a switch capable of both ICN and OpenFlow.

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

7.4. IMPLEMENTATION

Our software currently runs on general-purpose x64-architecture servers running Ubuntu Server. On these servers, we have installed stock Open vSwitch 2.0.2 [77] to enable configuration of operation by the OpenFlow protocol. In addition, we enable switches with ICN capabilities by installing the CCNx daemon [175], the open-source implementation of NDN.

7.4.1. OPENFLOW CONTROLLER IMPLEMENTATION

In order to implement our proposal, we have extended the POX (branch betta) controller [70] by designing an additional custom ICN module. We use the native POX Discovery module to perform topology discovery and learn a switch adjacency matrix. We reuse the OpenNetMon forwarding module designed in chapter 3 to perform legacy path computation and enable end-to-end IP forwarding. Additionally OpenNetMon may be used to perform fine-grained monitoring of flows. Finally, we implement a CCNx-specific plug-in that is added to communicate with ICN-enabled switches and perform ICN-specific path computations. The implementation of NDNFlow is published open source and can be found at our GitHub web page [176].

7.4.2. CCNX DAEMON IMPLEMENTATION

The CCNx daemon is extended by implementing an additional SDN plug-in, which sets up a connection to the POX ICN module, parallel to Open vSwitch's regular OpenFlow connection, and announces its ICN capabilities, capacity and information availability. The extension is realized similarly to our plug-in solving global NDN routing table complexity [141]. Whenever a CCNx daemon receives an Interest for which no flow or previously defined forwarding rule exists, it forwards this Interest to the POX ICN module. In turn, the POX ICN module looks up the appropriate location or exit-point of that Interest, calculates the appropriate path based on the topology information learned from the discovery module and announcements from CCNx-enabled switches and configures the intermediate NDN nodes accordingly. Finally, the Open vSwitch is configured by the controller as shown in figure 7.1.

7.4.3. PROTOCOL IMPLEMENTATION

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

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

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

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

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

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

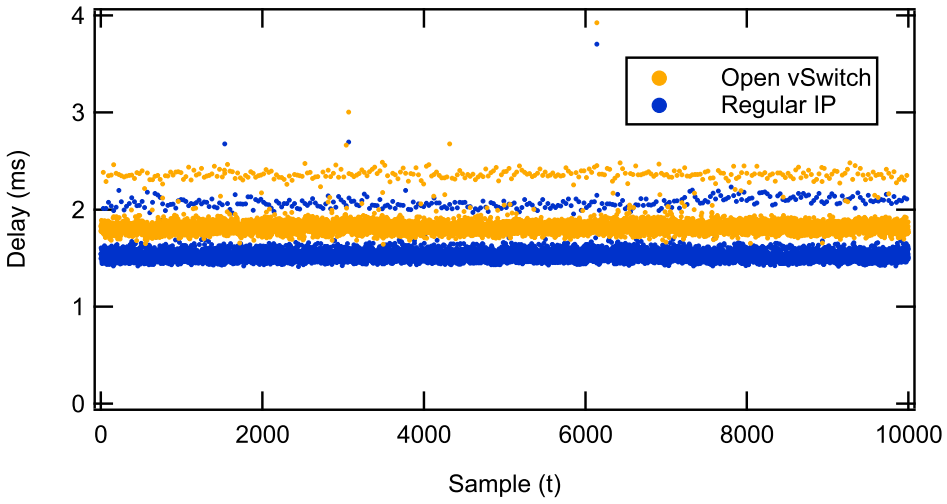


Figure 7.2: Baseline measurements using CCNx over regular IP and OVS.

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

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

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

7.5. EXPERIMENTAL EVALUATION

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

7.5.1. TESTBED ENVIRONMENT

We have conducted our experiments on a testbed of physical, general-purpose, servers all having a 64-bit Quad-Core Intel Xeon CPU running at 3.00 GHz with 4.00 GB of main memory and 1 Gbps networking interfaces. OpenFlow switch functionality is realized using the Open vSwitch 2.0.2 software switch implementation, Named Data Networking functionality by installing CCNx 0.8.2, both running in parallel on Ubuntu Server 14.04.1

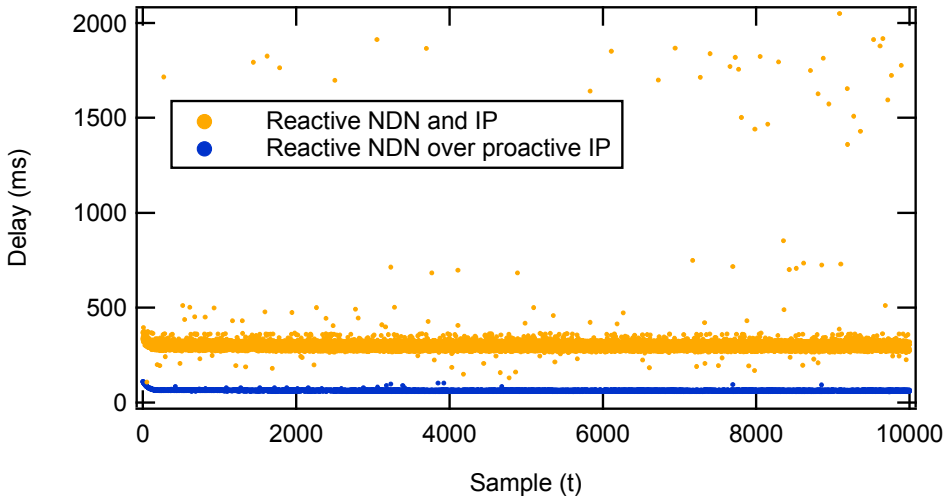


Figure 7.3: Measured delay of CCNx path setup in pro- and reactive configuration.

LTS with GNU/Linux kernel version 3.13.0-29-generic. The CCNx is connected to Open vSwitch via a socket to the internal bridge interface to realize connectivity, hence data is forwarded to and from CCNx through the OpenFlow LOCAL port.

Throughout, we use a 2-switch topology on which the discussed switching fabric and additional plug-ins from section 7.4 are configured. A third server is configured as controller using the POX controller and modules discussed in section 7.4.

In order to measure the delay time between requesting and receiving content we use *ccnping* [165], an NDN alternative to the popular application *ping* that can be used to confirm connectivity and measure round-trip times in classical IP networks. Similar to *ping*, *ccnping* sends an Interest per interval to a given destination prefix concatenated with a random value. When sending, *ccnping* stores the timestamp of creation and computes the round-trip time (RTT) upon arrival of the appropriate ContentObject. The *ccnping* server and client are installed on the 2 switches and connect to the CCNx switch fabric using the application interface.

7.5.2. EXPERIMENTS AND RESULTS

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

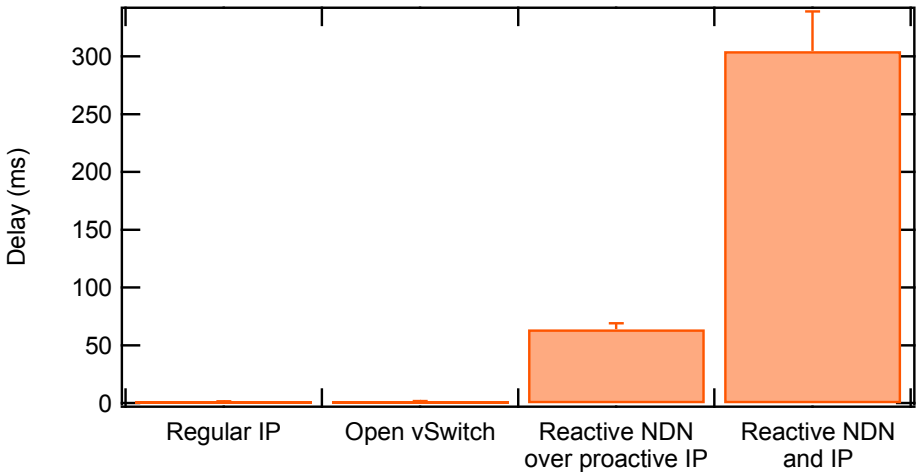


Figure 7.4: Delay averages and 95% confidence interval.

of the first packet), we continue measuring the delay of the first packet of every new flow with a reactive configuration of CCNx in both a (3) proactive and (4) reactive configured OVS network.

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

Figure 7.2 shows the results for (1) and (2), while figure 7.3 shows the results for (3) and (4). Figure 7.4 shows the relative averages and 95% confidence interval, giving: (1) a baseline of 1.534 ± 0.101 ms for CCNx over IP networks, (2) 1.834 ± 0.115 ms for a fully proactive configuration of CCNx and OVS, and (3, 4) 64.006 ± 5.028 and 304.630 ± 34.191 ms for a reactive NDNFlow configuration in a proactive and reactive OVS configuration, respectively, to determine the additional costs of configuring the content flows in CCNx and OVS.

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

7.6. CONCLUSION

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

8

DNSSEC MISCONFIGURATIONS

The former chapters mainly focused on improving on-going work. In this chapter, we evaluate an Internet Architecture improvement that has already been implemented, the Domain Name System Security Extensions (DNSSEC). The Domain Name System (DNS) is a crucial technology for the functioning of the Internet as it enables communication using domain names that are easier to remember than numerical IP addresses. The popularity of this mapping system is explained by the use of Fully Qualified Domain Names (FQDNs) as the primary component of URLs with which all Internet users identify websites. DNSSEC offers protection against spoofing of DNS data by providing origin authentication, ensuring data integrity and authentication of non-existence by using public-key cryptography. Although the relevance of securing a technology as crucial to the Internet as DNS is obvious, the DNSSEC implementation increases the complexity of the deployed DNS infrastructure, which frequently results in misconfiguration. Furthermore, similar transition problems apply to DNSSEC as to the network layer discussed in previous chapter, where one party relies on other parties to also configure DNSSEC correctly for correct functioning of the system.

In this chapter, we provide insight into the level of implementation and correctness of DNSSEC implementation. To do this, we measure and analyze the misconfigurations for domains in six zones (.bg, .br, .co, .com, .nl and .se). Furthermore, we categorize these misconfigurations and provide an explanation for their possible causes. We evaluate the effects of misconfigurations on the reachability of a zone's network. Finally, we research common denominators responsible for misconfiguration.

This chapter is based on a published conference paper [177] and extended journal article [178].

8.1. INTRODUCTION

The Domain Name System (DNS) [179] is a crucial technology for the functioning of the Internet as it enables communication using domain names that are easier to remember than numerical IP addresses. Among others, DNS maps human-readable hostnames to IP addresses and provides a distributed database from which users can request these mappings. The popularity of this mapping system is explained by the use of Fully Qualified Domain Names (FQDNs) as the primary component of URLs with which all Internet users identify websites.

The importance of DNS lies in the fact that it is not only useful to end users, but that it is also essential to several other core network technologies [180], such as telephone number mapping (ENUM), SIP, email, spam filtering and Microsoft's Active Directory for Windows. However, even though DNS is one of the fundamental building blocks of the Internet, its original design from 1983 focused on scalability and did not include security considerations. Even as early as 1990, the first flaws in the DNS were detected and the need for protection was discussed [181]. The Domain Name System Security Extensions (DNSSEC) were published in 1997 and their refinement in 2005 [182].

DNSSEC, in a broad sense, offers *protection against illegitimately falsifying data stored in the DNS* by providing origin authentication, ensuring integrity and authentication of non-existence. To make sure that the user receives authentic replies, DNSSEC deploys cryptographic keys. With private keys, digital signatures are generated for resources which can be verified by their public counterparts.

8.1.1. MOTIVATION AND PROBLEM DEFINITION

At the time of writing, there are 115,807,705 .com domains that are configured within name servers, of which 422,037 are signed [183]. For DNSSEC to work as intended, deployment has to span all levels of the DNS architecture. Adoption by all involved actors in the DNS resolution process is therefore essential for success. One big step was taken in July 2010 when the DNS root zone was signed [184]. Since then, resolvers are enabled to configure the root zone as a trusted anchor which allows the validation of the complete chain of trust for the first time. Nevertheless, even though 84% of existing domains could already be using DNSSEC, as more and more Top Level Domains (TLD) are being signed, less than 1% of authoritative domain name servers have implemented it [185]. Most commonly mentioned causes for this small percentage are:

- the implementation of DNSSEC increases the complexity for the management of the deployed DNS infrastructure,
- a misconfiguration might result in Internet users being unable to reach the protected network [186].

The effects of misconfiguration were very recently made apparent, when a new service from a major television network was unreachable by a significant number of end users due to DNSSEC misconfiguration. Furthermore, the television network resolved the problem by completely disabling DNSSEC authentication for their zone [187]. These types of failures and resulting unreachability of services are not rare incidents, but appear quite often [188]. In fact, there is an IETF Internet-Draft working on clarifying the authoritative domain's responsibility towards a correct DNSSEC configuration [189].

Besides authoritative domains, DNSSEC validation failures have also been reported for complete TLDs [190]. Such problems may occur more frequently when a new group of TLDs (up to 1400) [191] start rolling out within the next few years, since all of them must implement DNSSEC and misconfigurations of the zone files of domains could potentially hide them from the Internet.

Despite the importance of the stated problem, there does not exist sufficient information on the current status of DNSSEC deployed zones. Therefore, in this chapter, we measure the status of several DNSSEC-enabled production zones measuring both the level of DNSSEC implementation and the correctness of DNSSEC configuration. We believe that conducting experimental research on DNSSEC is of great value but, in order to get deeper insights, it should also be complemented with an analysis of the most common problems DNSSEC is experiencing in day-to-day production environments. Performing an analysis on real production data from operational zones brings a better understanding on the current status of DNSSEC deployment. Moreover, it also helps to define the biggest challenges that need to be overcome for this technology to succeed.

8.1.2. OUTLINE

The chapter is organized as follows. Section 8.2 summarizes the internal workings of DNS. Section 8.3 presents related work to misconfigurations, measurements and highlights proposed methods and shortcomings in DNSSEC. In section 8.4 the used measurement tools are presented, while section 8.5 discusses the performed "top-down" and "bottom-up" measurement scenarios.

First, section 8.7 presents and analyzes the results of the top-down measurement scenario containing .bg, .br, .co and .se domains. The domains are, among others, chosen as they are browsable through zone walking[1]. Furthermore, the .se zone draws importance for analysis as it was the first DNSSEC signed zone [192] in 2005, 5 years prior to the root domain, hence a high implementation is expected. Similarly, .co is also popular for commercial host names, the zone .br has one of the largest DNS registries in South America, while the .bg domain has signed its zone more recently.

Afterwards, section 8.6 presents and analyzes the results from the bottom-up verification approach on a larger dataset containing .bg, .br, .co, .com, .nl and .se domains. Where the former 4 domains are included for means of comparison, .nl and .com are added for respectively having the highest implementation of DNSSEC-enabled domains [193] and being the largest TLD available. A conclusion is drawn in section 8.8.

8.2. DNS AND DNSSEC

In this section, we summarize the functions and internal organization of the Domain Name System (DNS) and its Security Extensions (DNSSEC). In subsection 8.2.1 we discuss how DNS implements a distributed lookup directory. Subsection 8.2.2 discusses the most relevant extensions and possible errors in configuring DNSSEC.

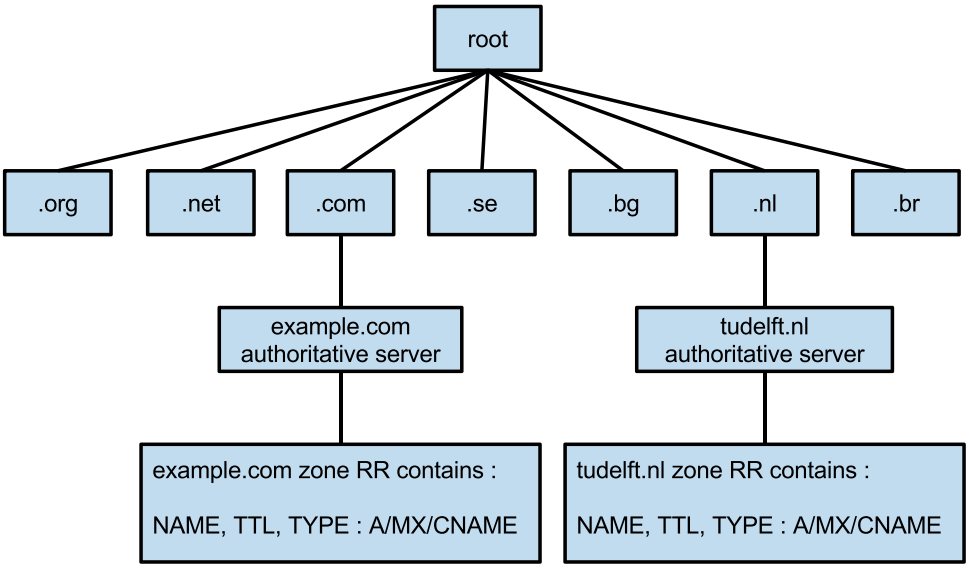


Figure 8.1: A tree depicting the hierarchical distributed nature of DNS names and their location in the distributed database.

8.2.1. DNS

The Domain Name System (DNS), first standardized in 1983 [179], primarily offers a distributed database storing typed *values* by *name*. Each record is called a resource record (RR), often shortly referred to as "record", containing the following ordered properties:

- The name of the record.
- The type of the record. Popular records include, the A-, AAAA-, MX-, CNAME- and SPF-records, respectively storing an IPv4 address, IPv6 address, the responsible mail exchange hostname for that domain, an alias referring to another hostname and domain-specific rules regarding spam policies according to the Sender Policy Framework protocol.
- The class code, specifying a name-space scope. Although multiple codes exist, usage of values different from IN for Internet are uncommon.
- The time-to-live, specifying in seconds how long intermediate or recursive DNS servers may cache that specific record, often defaulting to a zone's TTL.
- The length of the RDATA field, used for communication protocol implementation.
- The RDATA field, containing the record's actual stored data.

Most often, DNS is used to request mappings from computer hostnames to IP addresses. In order to support such a high frequency of requests, DNS employs a tree-wise hierarchy in both names and database structure. A so-called Fully Qualified Domain Name

[1] The process of zone walking is explained in subsection 8.2.3.

(FQDN) consists of multiple name components specifying the location of its records in a tree of databases. Clients, such as home and business PCs, connect to a local recursive (often caching) DNS server that traverses the tree to receive the requested information. In general, the traversed tree-wise structure of DNS consists of 3 layers: (1) The root-layer, a set of name servers named [a-m].root-servers.net. (2) The Top-Level-Domain (TLD) layer. (3) The authoritative layer.

Every recursive DNS server has a "root hints file", containing a list of all root-server hostnames and IP addresses. For means of load balancing and geographic distribution of requests, anycast addresses are used to deploy multiple servers per hostname. The root-servers themselves do not contain any mappings for FQDNs, but instead refer to Top-Level-Domain (TLD) servers responsible for the requested TLD by replying with an NS-record containing the name server of the next layer and its IP-address in an A-record. The top-most used types of TLDs are generic TLDs such as the domains .com, .net, .edu and .org, as well as country-code TLDs whose last name suffix refers to country-specific sites such as .nl for the Netherlands, .uk for Great-Britain, etc. These TLD-servers once again refer to the next layer, which is generally authoritative for that domain name and returns the requested mapping. Where further recursion is possible, commonly 3 steps are sufficient. The relation between the name and the place of its records in the distributed tree is summarized in figure 8.1.

8.2.2. DNSSEC

Although DNS has proven to be very scalable, the architecture shows many possibilities for both un- and intended malicious behavior and attacks. It is fairly easy to tune-in and mangle with DNS requests and replies by executing a so-called man-in-the-middle attack, hence secretly redirecting the client to obscure locations, for means of hijacking personal authentication details, or falsely denying existence of resources. This, for example, could occur at open WiFi hotspots, where providers often offer their own, potentially malicious, DNS service. Hence, DNSSEC has been introduced to authenticate the validity of both returned RRs and non-existent records through cryptographic signing of resources.

In order to support the cryptographic signing process, each domain has multiple associated keys containing at least 1 public-private key pair. For each record set (RRset) of distinct name and type, a signature is generated using the private key and stored in an RRSIG-record, which can be verified using the domain's public key stored in a DNSKEY-record, both placed in the zone of the domain.

To confirm the authenticity of the DNSKEY, which is essential to check the authenticity of any record in a zone, its digest, called the Delegation Signer, is stored in a DS-record in its parent zone. Recursively, the DS-record is signed in its local zone, and the process repeats until the root-zone is consulted. Since the root-zone has no ancestors, its DNSKEY-record is confirmed by globally publishing its digest, which is referred to as the Trust Anchor [194]. As summarized in figure 8.2, a DNS client or resolver recursively requests these records to determine authenticity of a record.

The recursive chain from Trust Anchor, to intermediate DNSKEY-, DS- and RRSIG-records authenticates each RRset of a properly configured DNSSEC domain. Part of managing public-private key pairs is refreshing them regularly to prevent malicious par-

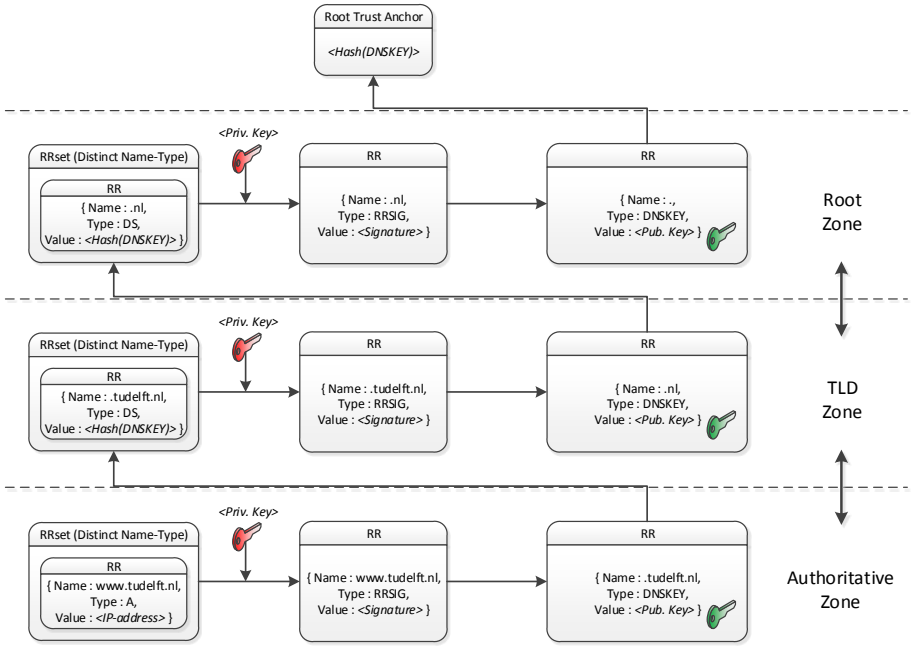


Figure 8.2: Relationship between resource record sets, keys and signatures for the chain of trust of the domain name www.tudelft.nl.

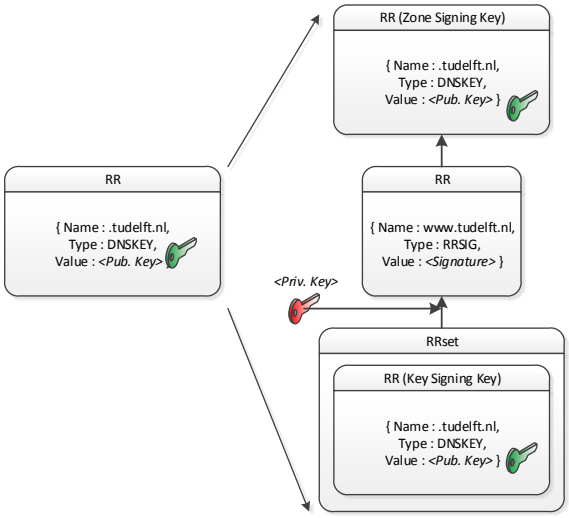


Figure 8.3: Relationship between Zone Signing Keys and Key Signing Keys.

ties from deriving the private key. Hence public-private key pairs are equipped with an expiration date and therefore need to be renewed at regular intervals. Replacing a DNS record, however, is non-trivial due to possibly long periods of caching in clients and intermediate resolvers. Furthermore, changing one's public-key does not only involve updating one's DNSKEY-record, but also implies updating the parent DS-record whose replacement needs to be performed by the TLD, a third party, again keeping cache synchronization in mind. Therefore, key rollover can be a tedious process [195].

In order to ease the process, a zone is equipped with 2 types of public-private key pairs, Key Signing Keys (KSKs) and Zone Signing Keys (ZSKs), both stored in DNSKEY-records and distinguished by a flag. KSKs concern the keys whose DNSKEY-record is confirmed by the parent's DS-record and are exclusively used to sign other DNSKEY-records in the zone, the ZSKs. ZSKs are hence used to sign all other types of resources in the zone. As ZSKs are signed by a local KSK, those public-private key pairs can rollover independent of the parent-zone. The KSK often is a longer, cryptographically more complex, key pair that changes less often. Besides a decreased need of replacing the key due to its greater complexity, KSKs only sign a limited number of resources (ZSKs) making them less prone to attacks as less in- and output of the key pair is available. The ZSK changes more often and may be cryptographically less complex if sufficiently often replaced with new keys. The relationship between KSK and ZSK is summarized in figure 8.3.

8.2.3. AUTHENTICATION OF NON-EXISTING RESOURCES

When a DNS server is queried for a non-existent record, i.e. there is no record for that requested name, it will respond with a NXDOMAIN message indicating its absence. Where DNSSEC so far authenticates existing resources, it is difficult to authenticate non-existing resources as non-existent records (1) have no corresponding signature, and (2) if for every NXDOMAIN response a signature would be computed online, key pairs would be more vulnerable to attacks. Since an NXDOMAIN message may be hijacked and replaced with a false response to silently mislead a user, it is important to authenticate non-existing resources.

In order to authenticate non-existent resources, DNSSEC introduces NSEC-records [196] containing a linked-list of existing records ordered by name, hence actively denoting non-existing namespaces. For an example domain named example.com with just 2 subnames mail. and www., the NSEC-records would look as follows:

- The first NSEC-record named example.com refers to mail.example.com indicating that mail.example.com is alphabetically the first subname of example.com, hence actively denying the existence of any subnames that would be ordered prior to it, such as ftp.example.com.
- The second NSEC-record named mail.example.com refers to www.example.com indicating there are alphabetically no subdomain names between them, hence denying the existence of intermediate subnames, such as news.example.com.
- The final NSEC-record named www.example.com refers back to the zone name example.com, indicating it is alphabetically the last record.

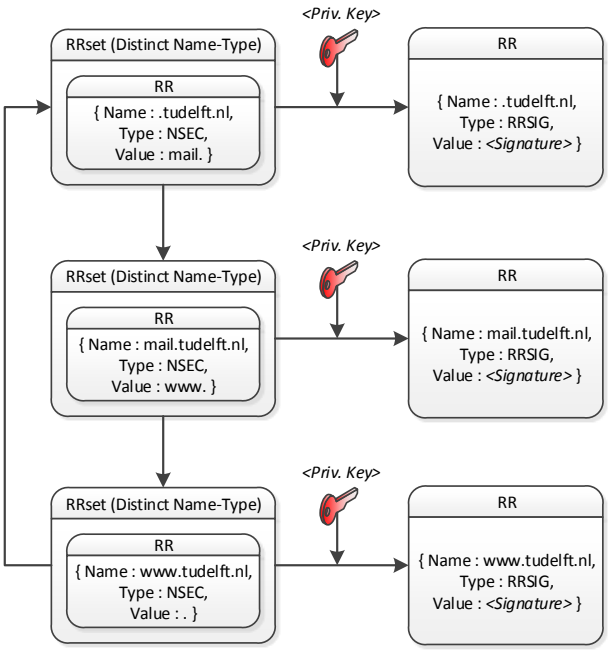


Figure 8.4: Relationship between resource records and respective NSEC-records.

8

The relationship between resource records and NSEC-records is summarized in figure 8.4. An NSEC-enabled server will reply with the appropriate NSEC-record to requests for non-existing resources. Each existing NSEC-record, of whom as many exist as names exist for a domain, is ultimately signed by an RRSIG-record to confirm authenticity of the claimed non-existence. A property of the NSEC-records is that they can be iterated to gather a *complete list of valid subnames* for a domain or TLD, a process called zone-walking. We have extensively used this method to extract the lists of domain names from our selected TLDs.

However useful to our research, publishing the complete list of available resources does not relate to the occasional request of non-existent resources and may even raise concerns on privacy. Therefore, the NSEC3 additions hash the zone-specific name-component (i.e., mail. and www.) prior to ordering, and generate a recursive linked-list of NSEC3-records containing hashed values [197]. In order to authenticate non-existence of a resource, the DNS-server will hash the requested zone-specific name component and return the NSEC3-record indicating the non-existent range to which it belongs. Hence, proving non-existence without giving away existing names.

8.3. RELATED WORK

In this section, we discuss previous work on DNSSEC. We found that most studies focus on the performance of DNSSEC, such as latency, delay, or resource load on the server, rather than on misconfigurations [198]. In [180], the authors analyze the availability of DNSSEC resolution and service, but omit configuration correctness. Lian et al. [199] present a quantitative measurment study towards the capability of resolvers and end-users to perform DNSSEC authentication, which represent the client side of DNSSEC. The authors of [200] present additional security vulnerabilities to DNSSEC itself, while in [201] the security benefits of the NSEC3 hashed authenticated denial of existence[2] are evaluated.

One related study focuses on quantifying and improving DNSSEC availability [186]. The authors first identify what kind of misconfigurations in DNSSEC can affect a DNS query request. They list the potential failures due to DNSSEC misconfiguration, and then they create a metric to quantify those DNSSEC misconfigurations. They classify DNSSEC misconfigurations into three categories:

1. Zone (missing/expired/invalid RRSIGs covering zone data, or missing DNSKEY RRs required to verify RRSIGs).

2. Delegation (bogus delegations because of lack of appropriate DNSKEYs in the child zone corresponding to DS RRs in the parent zone, or insufficient NSEC RRs to prove an insecure delegation to a resolver).

3. Anchor (stale trust anchors in a resolver, which no longer match appropriate DNSKEYs in the corresponding zone).

They analyze 1,456 signed zones out of which 194 show to be misconfigured [202]. Out of these, most of the misconfigurations are related to zone data that correspond to the first class of misconfigurations. However, the authors do not explain why this was the case and what were the main technical causes for such misconfigurations. The class 1 misconfigurations arise due to missing or outdated RRSIGs or DNSKEYs and, as explained in section 8.1, the deployment of those records in the zone file is the responsibility of the zone administrator. Technically, the administrator should always ensure the correctness and validity of RRSIGs and DNSKEY deployment. Hence, the previous work does not give insight in the causes of misconfiguration, nor its effects in authenticity confirmation and reachability by a DNSSEC-aware resolver. Furthermore, the analysis was done in 2010, 5 years ago when DNSSEC was in an earlier stage of deployment compared to now in 2015 when DNSSEC is more widely implemented. In this chapter, we analyze over 122,779 signed domains from .bg, .br, .co, .com, .nl and .se domains.

8.4. MEASUREMENT TOOLS

There exist several different tools to work with DNSSEC. However, most of them are intended to be used by zone administrators in order to verify their own zone file before publishing it and require the user to have the complete zone file in order to perform

[2] Summarised in subsection 8.2.3.

their tests. Examples of such tools include Verisign's jDNSSEC [203] or NLnetLabs' LDNS [204]. We selected a set of tools that are able to perform tests over a list of several domains without possession of their zone files, that is to say, from the point of view of an external user. Furthermore, we selected these tools based on execution time and ease of automation to ease the process of checking a large number of domains. We have used the following measurement tools to perform our measurements:

1) Dnsrecon [205]

A DNS enumeration program, written in Python, that allows to discover relevant information from the content of a zone. It performs several types of enumerations including zone transfer, reverse lookup, a Google lookup and, most importantly, zone walking using NSEC-records as discussed in section 8.2.3. After testing the tool, we found its usage straightforward and effective since it provided us with the necessary means to easily and effectively retrieve all the authoritative name servers from a zone.

In the process of retrieving the domain lists for the TLDs, we enhanced the program with the following 3 functions: (1) To decrease the chance and impact of getting blocked by a DNS server due to excessive usage, we added support for multiple DNS servers. (2) We added the possibility to save the current state of iteration, so we could continue iteration from a different source IP-address when blocked. (3) Initially, Dnsrecon verified a domain's intent to deploy DNSSEC capability by checking whether the authoritative name server returned a DNSKEY-record for its hostname. Instead, we verified the intent through the DS-record at its parent.

2) DNSSEC-Debugger [206]

A web-based tool developed by Verisign Labs that inspects the chain of trust for a particular DNSSEC-enabled domain. It shows the step-by-step validation of a given domain and indicates any error or warning found in its DNSSEC configuration. We found DNSSEC-Debugger to be fast (analysis consumes approximately 3 seconds per domain) and ideal for automation as any domain can be inspected by executing a HTTP-request to `http://dnssec-debugger.verisignlabs.com/<DOMAIN>`. We use DNSSEC-Debugger to perform the top-down validation approach used in our first measurement scenario in subsection 8.5.1 and section 8.6.

3) Google's Public DNS Service [207]

Found at IP-addresses 8.8.8.8 and 8.8.4.4, Google offers a free and globally accessible DNSSEC-enabled DNS resolution service, which can be used as an alternative to one's in-house or ISP-provided DNS resolution server. In order to evaluate the effects of DNSSEC misconfiguration on the reachability of a domain, we assume a misconfigured DNSSEC domain to be unavailable when it does not pass Google's Public DNS Service.

4) Dig [208]

Dig (domain information groper), part of the popular DNS server BIND, is a command-line tool that can be used to query DNS servers. It is DNSSEC capable and can be used to verify the DNSSEC chain of trust from a top-down and a bottom-up perspective. However, we found that the current version queries all possible name servers for a TLD or authoritative zone for their A-record, even when glue records are known, when using the top-down approach, resulting in an infeasible number of lookups. Hence, we only used Dig in a bottom-up approach using a DNSSEC-capable resolver as performed in our second measurement scenario in subsection 8.5.2.

8.5. MEASUREMENT SCENARIOS

The DNSSEC chain of trust, consisting of a per-domain zone overlapping chain of public-private key-pairs and signatures as explained in section 8.2, can be verified in two distinct approaches, being:

1. A top-down approach, where first the root zone is verified against the previously known root trust anchor, followed by the TLD zone against its corresponding DS in the root zone, finished by the authoritative zone against its corresponding DS in the TLD zone.

2. A bottom-up approach, where - in reverse other - first the authoritative zone is verified against the TLD zone, which is verified against the root zone, which in turn is verified against the root trust anchor.

In terms of authenticity verification denoting either authentic or not, both methods are correct. However, in terms of finding the exact misconfiguration both are incomplete. For example, a top-down approach will assume an authoritative zone to be DNSSEC incapable when it finds no corresponding DS in the TLD zone, while the authoritative zone may serve private-public key pairs and signatures. In such a case, one could argue the domain intends to employ DNSSEC but fails to do so as it did not properly communicate the DS record to be published by the TLD. This specific misconfiguration cannot be found by a top-down approach. The bottom-up approach will find the previously described misconfiguration.

The bottom-up approach, however, assumes an authoritative zone to be DNSSEC incapable when it finds no RRSIG for the record it tries to authenticate. This implies that misconfigurations where the intention to apply DNSSEC by the existence of possible authoritative DNSKEYs or corresponding DSes in the TLD are omitted. To partially solve this problem and get insight in both misconfiguration errors, we have performed two different measurement scenarios on distinct datasets described in subsections 8.5.1 and 8.5.2.

8.5.1. TOP-DOWN MEASUREMENT SCENARIO

The measurement of the different domains in our first measurement scenario consists of 4 different phases, followed by an additional 5th phase in which we evaluate the effects of misconfiguration in everyday use. The first phase consists of gathering a comprehensive list of domain names. To do this, we use Dnsrecon to perform zone-walking of the 4 NSEC-enabled TLDs .bg, .br, .co and .se, hence retrieving extensive lists of domain names from these domains.

The second phase consists of filtering the list of domain names by the intent of them being DNSSEC enabled. We assume a domain name to intend to be DNSSEC enabled when a DS-record for that domain is registered at its TLD-zone. Filtering is performed by iterating the list of domain names and performing DS-record lookups using the internal functions of Dnsrecon.

Having retrieved a list with a sufficient number of DNSSEC-enabled domains, we verify their configuration using the DNSSEC-Debugger online tool from Verisign Labs.

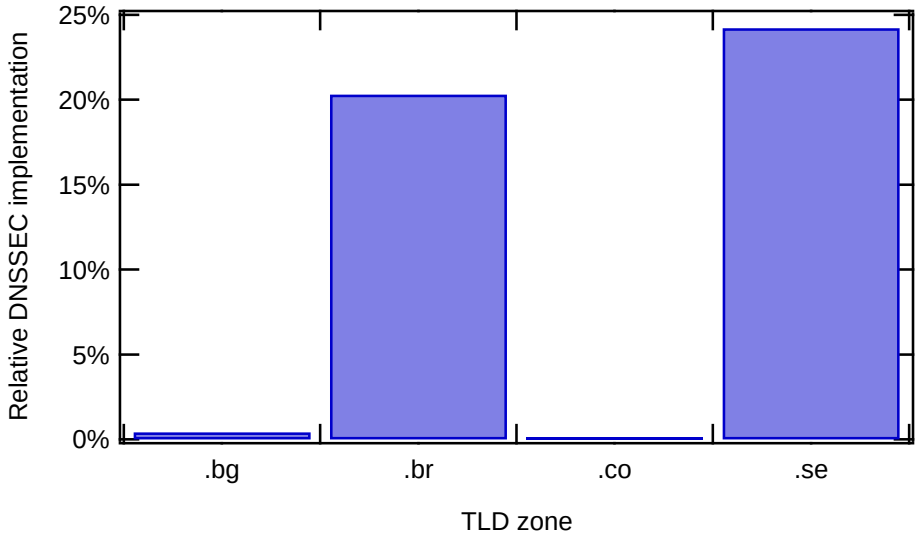


Figure 8.5: Relative implementation of DNSSEC per country-code TLD.

We programmatically iterate through the list, performing a HTTP-request to the appropriate URL and parse the response for further analysis. To verify the correctness of the DNSSEC-Debugger, we took a sample from the results and compared them with results from Dig. Normally, it takes approximately 3 seconds to receive the verification result for one domain name. In order to overcome this time limitation and to speed up the process, we perform up to 10 lookups in parallel by employing multithreading.

In the 4th phase we categorize the misconfiguration in the categories and subcategories denoted in table 8.3 enabling analysis by the type of misconfiguration. Besides the expected DNS and DNSSEC related misconfigurations, we found 2 additional errors. We found that a zone's DS could be retracted in the time between retrieving the list of DNSSEC-enabled domains and performing the measurements, meaning the domain has withdrawn from implementing DNSSEC. Furthermore, we found an additional error where the server does not implement the resource record type DNSKEY and, therefore, is DNSSEC incapable.

After performing the initial measurements, we verified the effects of misconfiguration by requesting the A-records associated with misconfigured domains from Google's Public DNS Service which performs DNSSEC authentication verification.

8.5.2. BOTTOM-UP MEASUREMENT SCENARIO

In our second measurement scenario, we use a publicly available dataset of domain names known as the DNS Census 2013 [209]. The census is published anonymously and contains over 2.6 billion DNS records gathered from over 106 million domain names in 2013. Although the dataset is incomplete and it does not contain official sources, the census can be considered significantly large to be representative. From the list of all available domain names we have selected the .bg, .br, .co., .com, .nl and .se TLDs to per-

Table 8.1: Historical statistics on DNSSEC implementation per ccTLD.

ccTLD	Statistics from:	Total	DNSSEC	%
.bg	08/2008 [210]	N/A	80	N/A
.br	01/2014 [211]	3310972	487471	14.72%
.co	10/2013 [212]	1560000	196	0.01%
.com	2015 [183]	116259506	429047	0.37%
.nl	2013 [193]	5388364	1700000	31.55%
.se	09/2013 [213]	1292596	327684	25.35%
Total				

form our measurements on. We have chosen the .bg, .br, .co and .se TLDs as those have also been used in our initial measurement scenario, .nl since it has the highest number of DNSSEC implemented domains and .com since it has the largest number of domains in general. The high number of domains in the .com TLD, over 64 million, implies it is difficult to verify all chains of trust within a feasible timespan. To guarantee that the verified subset will be representable, we have randomized the order of the list up front.

Alike to the latter 4 steps of the previous scenario, we iterate the list of domains and perform DNSSEC validation using Dig and a DNSSEC-capable resolver. Due to the bottom-up approach, authoritative zones are considered DNSSEC capable if they publish RRSIGs for the A-records of their domain name. From the DNSSEC-capable domains, the chain of trust is analyzed and categorized. Finally, the reachability of misconfigured domains is also verified using Google's Public DNS Service.

8.6. RESULTS AND EVALUATION: TOP-DOWN APPROACH

In this section, we discuss and evaluate the results from the experimental measurements described in subsection 8.5.1. Subsection 8.6.1 shows the results from the first and second measurement phase, gathering domain names and measuring the integration of DNSSEC in the different zones. Subsection 8.6.2 categorizes the misconfigurations of the zone .se into the type of misconfiguration. Finally, subsection 8.6.3 analyzes the result of the misconfigurations on the availability of the domain.

8.6.1. DNSSEC IMPLEMENTATION

In this subsection, we present the results of the first two phases of our measurements, (1) gathering domain names and (2) measuring the integration of DNSSEC within the list of zones. While gathering the lists of domain names by walking the NSEC-records, we were often blacklisted by the TLD name servers as the excessive amount of performed DNS requests are classified as possible attacks on the service. As shown in table 8.2, for most zones we were able to gather and analyze a considerable number of domain names. The .br zone, however, appeared to have additional counter-measures against zonewalking. Regularly, the .br TLD name servers would reply with an NSEC-record indicating the requested domain was the last domain name of the zone, hence terminating the zone walking process as it appeared to be finished.

Table 8.1 shows historical statistics found on per-zone DNSSEC implementation,

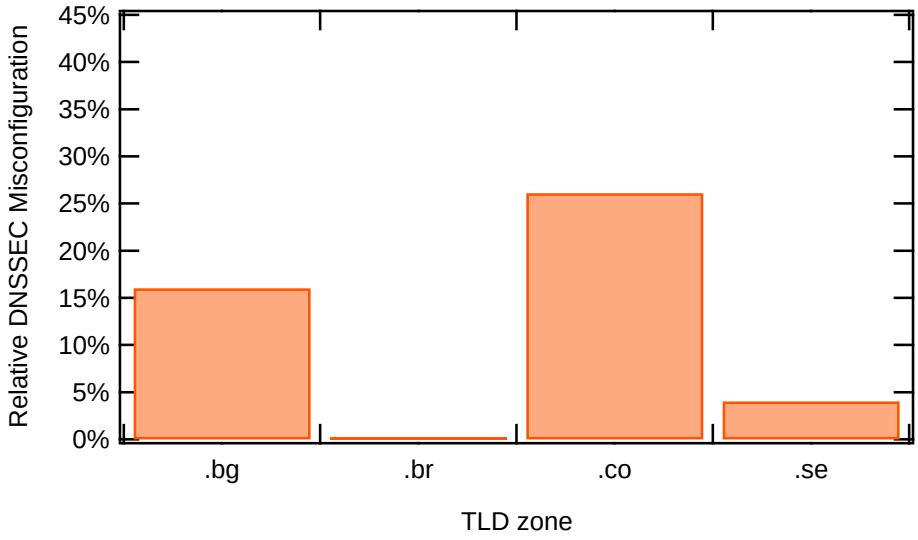


Figure 8.6: Relative misconfigurations found through top-down verification of DNSSEC domains.

Table 8.2: Top-down measurement on DNSSEC implementation per ccTLD.

ccTLD	Retrieved	DNSSEC	%	Misconfigurations	%
.bg	38806	162	0.42%	26	16.04%
.br	2481	504	20.31%	2	0.00%
.co	151707	23	0.02%	6	26.09%
.se	89772	21748	24.23%	876	4.03%
Total	282766	22437	7.93%	910	4.06%

while table 8.2 shows the number of domain names we were able to gather and check for DNSSEC implementation. Although our lists are incomplete, we were able to confirm the relative implementation of DNSSEC for the selected zones. We found that the zones .bg and .co both have a very low implementation of DNSSEC, resulting in a very small set of DNSSEC-enabled domains. For the zone .br, we found a significant number of DNSSEC-enabled domains. Due to the aforementioned zone-walking countermeasures, however, we were unable to gather a large set of DNSSEC-enabled domains for the .br domain. For the zone .se, we were able to gather an extensive number of domains and DNSSEC-enabled domains. Hence, we continue to further analyze the configuration mistakes found in the zone .se. Figure 8.5 shows the relative percentage of DNSSEC implementation per zone.

8.6.2. DNSSEC MISCONFIGURATIONS

As seen in table 8.2 and figure 8.6, the ccTLD .se has a significant number of misconfigurations. Table 8.3 shows the misconfigurations related to the categories and subcategories listed in section 8.5. As also prospected in figure 8.7, approximately two-thirds

Table 8.3: Top-down measured misconfiguration statistics for the ccTLD .se and respective reachability by Google's public, DNSSEC-aware, DNS resolution service.

Error Category	Subcategory	Misconfigurations	%	Unreachable	%
DNSKEY	**Total**	**564**	64.38%	**477**	84.40%
	Not found	261	29.79%	259	99.23%
	Invalidated by DS	88	10.05%	3	3.41%
	KSK invalidated by ZSK	215	24.54%	214	99.53%
	Valid but expired	0	0%	N.A.	
RRSIG	**Total**	**1**	0.11%	**1**	100.00%
	Not found	0	0%	N.A.	
	Invalidated by ZSK(s)	0	0%	N.A.	
	Valid but expired	1	0.11%	1	100.00%
	Invalidate RRset	0	0%	N.A.	
General DNS failure	**Total**	**295**	33.68%	**163**	55.25%
	REFUSED	12	1.37%	11	91.67%
	SERVFAIL	191	21.80%	142	74.35%
	Time-out	64	7.31%	9	14.06%
	No SOA	4	0.46%	0	0.00%
	SOA Serial differs	24	2.74%	1	4.17%
Miscellaneous	**Total**	**16**	1.83%	**7**	43.75%
	DS retracted	14	1.60%	5	35.71%
	DNSKEY RR Failure	2	0.23%	2	100.00%
All categories	**Total**	**876**	100%	**647**	73.86%

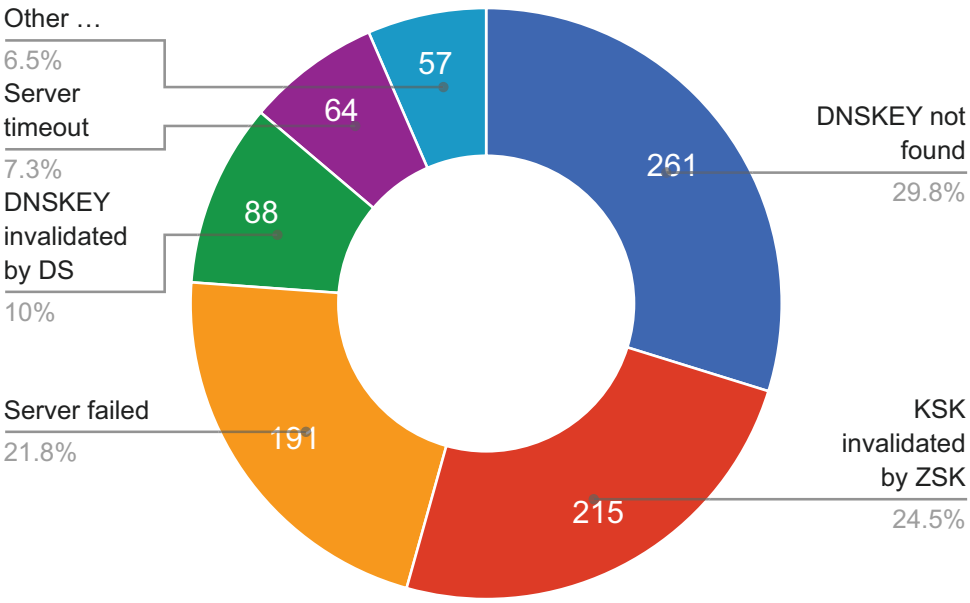


Figure 8.7: Distribution of most significant DNSSEC misconfigurations.

of the misconfigurations are related to configuration of the DNSKEY records. Slightly less than one-third of the misconfigurations are caused by missing DNSKEY records, indicating there once was an intention or maybe even a running configuration to deploy DNSSEC. However, the DNSSEC configuration has never been properly configured or was removed from the authoritative name server. Slightly less than a third of the misconfigurations indicate a Key Signing Key that is not properly signed by its Zone Signing Key as described in subsection 8.2.2, hence breaking the chain of trust. The situation where a ZSK invalidates the zone's KSK indicates a problem with the key-rollover of the KSK, most probably it has not been resigned after it was renewed. Once DNSKEY configuration is properly done, we find little evidence in the internal configuration of the authoritative name server, from the category of possible RRSIG misconfigurations we only found 1 occurrence of an expired signature.

Stunningly, a third of the misconfigurations seem to revolve around general DNS misconfigurations or errors that could also occur in non-DNSSEC environments. Especially the number of reported server failures and time-outs are surprisingly high. We were unable to confirm whether these errors are strictly related to non-DNSSEC configuration, and thus unrelated to DNSSEC, or are caused by a server malfunction due to incompatibility with the DNSSEC-extended query. We did however find two occurrences with a more specific error, where the server indicated incompatibility with the DNSKEY resource record type, showing that DNSSEC-incompatibility with DNS servers once intended to perform DNSSEC is a problem. Finally, we found 14 occurrences in which the authoritative administration retracted its intention to implement DNSSEC before we were able to scan its zone for misconfiguration. In figure 8.7, we show the distribution

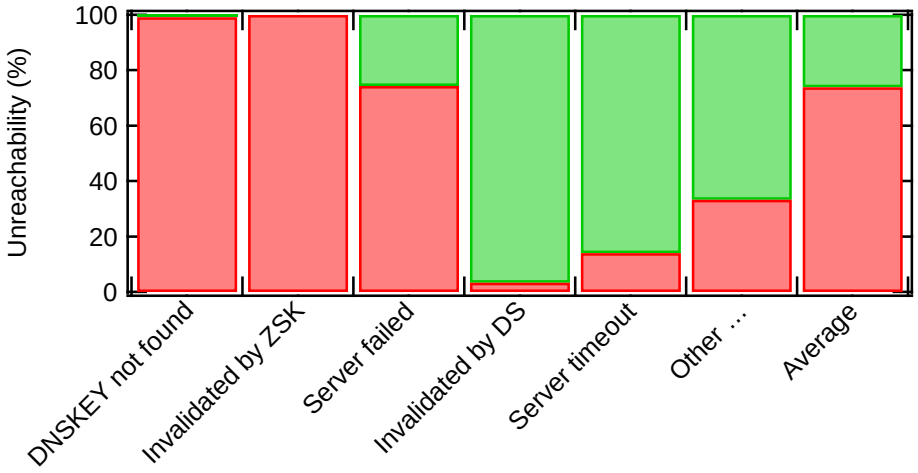


Figure 8.8: Relative misconfigurations per category in the ccTLD .se found through top-down evaluation.

among the most significant configuration errors.

8.6.3. EFFECTS ON AVAILABILITY

After performing the measurements, we proceeded to verify the reachability of the misconfigured domains using a DNSSEC validating resolver. For that purpose, we used Google's Public DNS Service, which has implemented DNSSEC validation by default since May 2013 [207].

The result of this experiment shows that 73.86% of the misconfigured domains in the ccTLD .se were completely unreachable from a DNSSEC-aware resolver. The remaining 26.14% of domains still had some misconfiguration, though not as severe to provoke the domain to become unavailable. To learn the impact of a misconfiguration, we correlated the (un-)reachability of each domain to its misconfiguration category in table 8.3.

Summarized in figure 8.8, after combining the categories with less than 50 misconfigurations, the impact on reachability of the most common misconfiguration types becomes clear. Concerning the DNSSEC-specific misconfigurations, the impact of a missing DNSKEY record or a ZSK being invalidated by its KSK is large, nearing a 100% of unreachability. A DNSKEY invalidated by the parent DS-record indicating a potential security breach of the complete domain, however, only fails integrity checks at 3.41% of the sampled domains, even though this error may be considered as serious as the previous DNSKEY-related errors. A general DNS server failure when the DNSKEY is requested leads to an unreachability level of 74.35%, similar to the overall average. Finally, we notice server timeouts are handled correctly in most cases by the caching functionality of the resolver.

Table 8.4: Bottom-up measurement on DNSSEC implementation per ccTLD.

ccTLD	In dataset	Retrieved	DNSSEC	%	Misconfigurations	%
.bg	13288	13288	549	4.13%	41	7.47%
.br	572506	572506	25037	4.37%	649	2.59%
.co	72305	72305	6871	9.50%	260	3.78%
.com	64337635	42239548	122779	0.29%	55297	45.04%
.nl	1062209	1062140	260752	24.55%	26278	10.08%
.se	352235	352235	128008	36.34%	53321	41.65%
Total	66410178	44312022	543996	1.23%	135846	24.97%

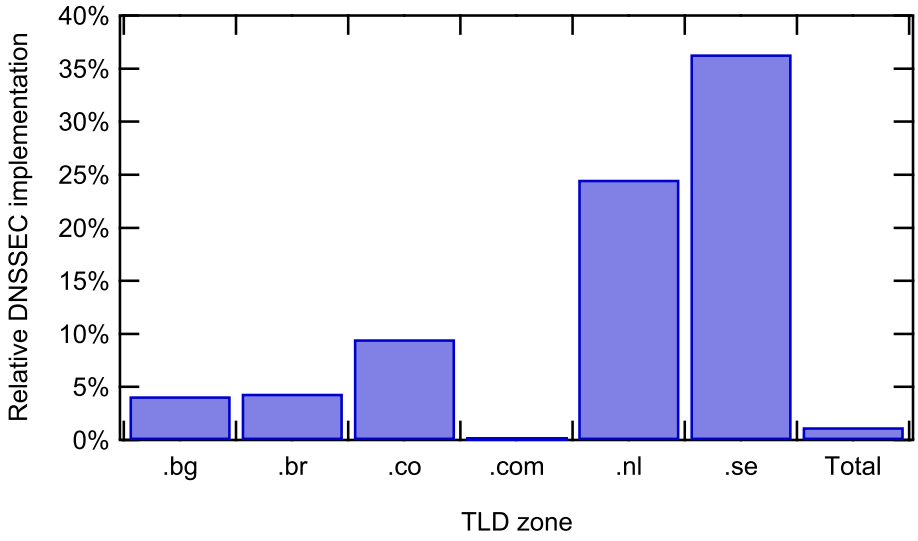


Figure 8.9: Relative implementation of DNSSEC found through bottom-up evaluation per country-code TLD.

8.7. RESULTS AND EVALUATION: BOTTOM-UP APPROACH

In this section, we discuss and evaluate the results from the experimental measurements described in subsection 8.5.2. Subsection 8.7.1 shows the results from the first measurement phase, measuring the integration and initial validation of DNSSEC in the different domains. Subsection 8.7.1 further discusses the different categories of found misconfigurations and their implications. In subsection 8.7.3, we perform further experiments on the root cause of the most common misconfiguration mistake. Finally, subsection 8.7.4 discusses additional results we collected on the validity of signatures.

8.7.1. DNSSEC IMPLEMENTATION

We were able to traverse all available domains from the .bg, .br, .co, .nl and .se TLDs. Due its large size, we were only able to retrieve 65.74% of available .com domains, resulting in over 42 million retrieved domains. Due to upfront randomization of the traversed list,

Table 8.5: Bottom-up measured misconfiguration statistics for per ccTLDs.

TLD	Validation successful	No DS Found	RRSIG Expired	Timeout	Measurement Error
.bg	481	28	1	12	27
.br	23385	403	61	185	1003
.co	6507	89	5	166	104
.com	66580	38213	560	16524	902
.nl	232334	4789	4124	17365	2140
.se	63987	49094	10	4217	10700
Total:	393274	92616	4761	38469	14876

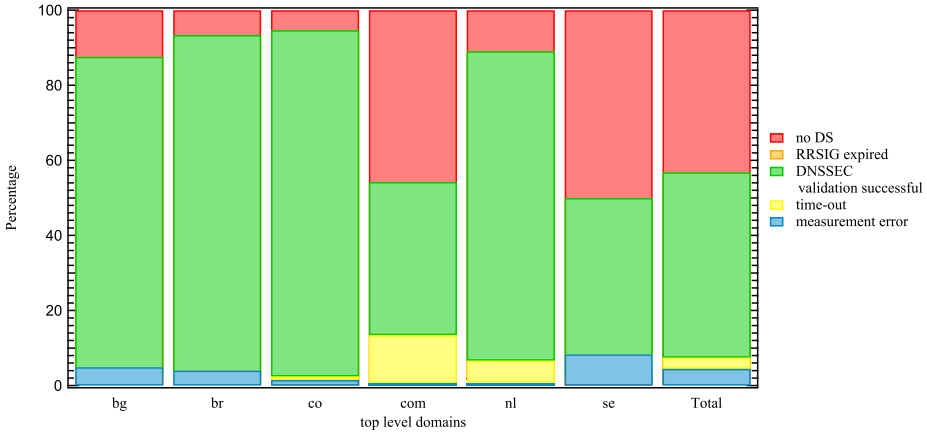


Figure 8.10: Distribution of most significant DNSSEC misconfigurations through bottom-up evaluation.

we claim to have retrieved a representable dataset. Table 8.4 and figure 8.9 show the level of DNSSEC implementation and ratio of found misconfigurations.

Due to the difference in measuring the intent to employ DNSSEC compared to the previous measurement strategy, we find different numbers for the implementation of DNSSEC. Particularly, .bg, .co and .se show much higher numbers of domains implementing DNSSEC, due to the inverse direction of detection. Where previously a DS record in the TLD denoted a zone DNSSEC capable, now an existing RRSIG for the domain does. The TLD .br, on the other hand, shows a lower number of implemented number of DNSSEC-capable domains, which may be explained by the small dataset of verified domains in the previous measurement scenario.

Finally, .nl and .com are added to the list of traversed domains. Where .nl shows a high implementation of DNSSEC, the implementation of DNSSEC in .com, the largest TLD, remains excruciatingly low. However, .com's level of misconfiguration is very high, due to a specific misconfiguration explained in the following two subsections.

8.7.2. DNSSEC MISCONFIGURATIONS

Where the bottom-up approach not only results in more domains to be considered DNSSEC capable, it may also change the detected misconfiguration as a domain can be misconfigured at multiple locations in the chain. From the domains that were verified to implement DNSSEC, table 8.5 shows the per-TLD validation results and the relative configuration distribution is shown in figure 8.10. Apart from the previously witnessed relative increase in DNSSEC implementations, we also witness an increase in misconfigurations within found DNSSEC domains.

The increase in misconfiguration is explained by the fact that where a DNSSEC-capable domain without a DS record would be considered DNSSEC incapable in the top-down scenario, in the bottom-up scenario it is considered a distinct misconfiguration categorized as "No DS Found". In fact, the results show that the main misconfiguration error concerns domains that have no valid DS published in their respective TLD. Since the bottom-up approach first verifies local RRSIG and DNSKEYs before requesting a DS, it implies that further configuration of the zone is correct and the chain of trust is merely broken by this one absent record. Hence, in total over 68% of the misconfigurations is due to this relatively small error of not communicating one's DS to the TLD.

Furthermore, we find that where the top-down approach showed many different categories of misconfigurations, in the bottom-up approach further errors are reduced to an expired RRSIG or time-out. The high occurrence of expired RRSIGs and absence of errors in DNSKEY validation (and exactly vice versa in the top-down approach) implies that domains that have DNSKEY validation problems (either due to DS- or KSK/ZSK- invalidation) also let their signatures expire. Hence, there was once a valid chain of trust which became outdated and invalid due to administrative neglection.

Finally, we had few occurrences in which Dig would crash with unclear errors. For means of completeness, we have added these occurrences as measurement error to our dataset.

8.7.3. ANALYSIS OF DOMAINS WITHOUT A DS

As explained in the previous two subsections, many more misconfigurations have been found due to the detection of DNSSEC-capable domains implementing signatures and private-public key pairs that omit communicating their delegate signer to their TLD. Whereas implementing DNSSEC locally may be a matter of configuration and key computation, uploading the DS is a process that is TLD specific and may require periodic renewal. Hence, we find many domains that show this specific error.

Most owners of a domain do not run their own DNS- and webservers, but instead use a webhoster to arrange those technicalities for them. Therefore, it is interesting to find whether the source of these many misconfigurations share a common cause. Hence, for each domain we have performed a *whois* lookup for the IP address stored in its A-record. A whois lookup is an information request about the owner of an IP address or domain name. For this, we have used the whois host server of Team Cymru [214], which contains mappings from IP address to Autonomous System (AS). We use the AS to identify the webhosting party that serves that particular domain.

Given that we know the technically responsible party for each domain, we have counted the number of DNSSEC-capable domains without DS in its TLD per party. Fig-

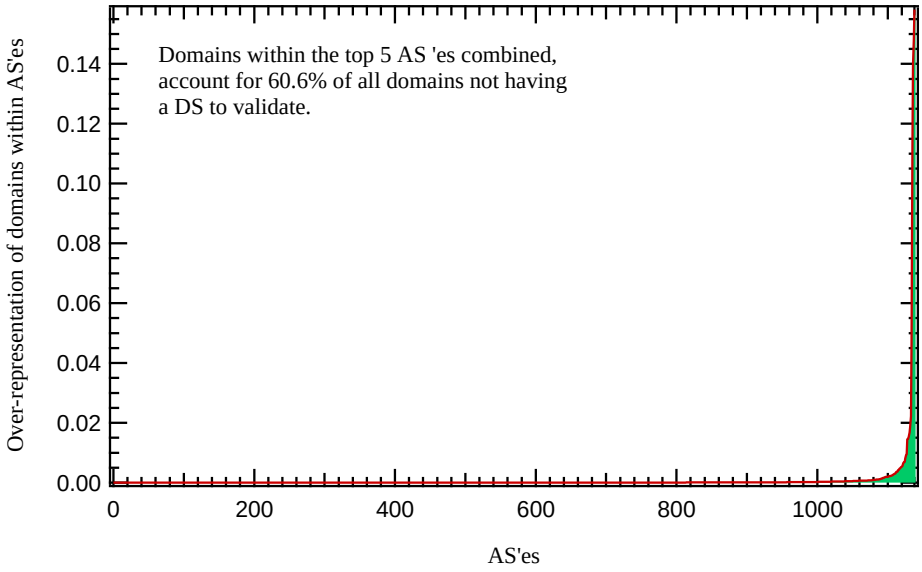


Figure 8.11: Ordered distribution of ASes overrepresented in the misconfiguration category of missing DSes.

ure 8.11 shows the over-representation in percent points from the average, ordered by the level of over-representation. From this figure we derive that the top 5 ASes account for 60.6% of all domains not having a DS in its TLD to verify it, hence breaking the chain of trust. Hence, *most misconfigurations are caused by a very limited set of webhosting parties.* This conclusion is further backed up by the fact that the 35,153,800 domains containing A-records refer to only 3,370,503 unique IP addresses, indicating many providers deploy virtual hosting by having a single webserver host many websites. Once such a service is affected by a configuration error, all of its domains are.

8.7.4. SIGNATURE VALIDITY

Additionally, we were able to measure the validity duration of the RRSIGs of the found traversed names by subtracting the expiration field by its inception field. Figure 8.12 shows the probability distribution function of the measured validity periods. A 3-week validity period is most common, followed by a (short) 2 week period. Although a few shorter intervals occur, round numbers such as half a year (183 days), 1 month (30 days) and 40 days are most popular. Interestingly, we found 1 signature that wouldn't expire until 69 years after its inception. This value conforms to the maximum duration implied by the compulsory use of serial number arithmetic [215], which is necessary to wrap-around the maximal available date and time of 2^{32} seconds (around 136 years) after 1 January 1970 00:00:00 UTC.

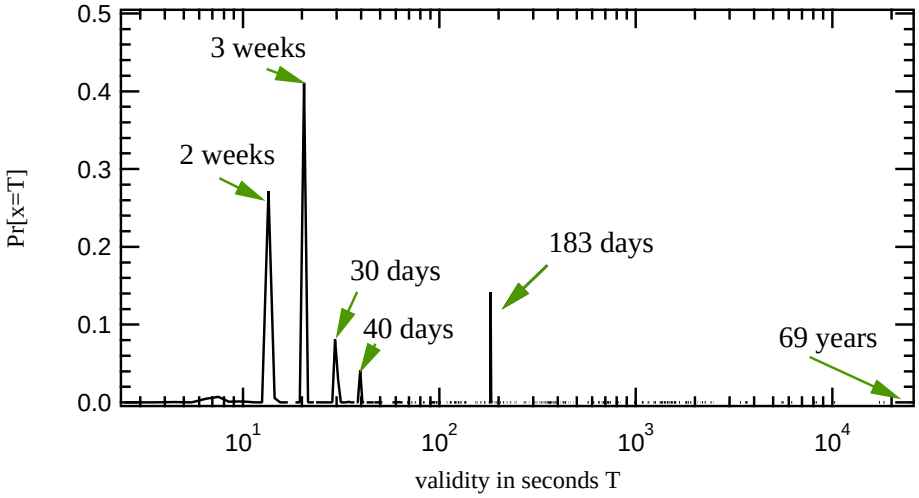


Figure 8.12: The probability distribution function of signature validity.

8.8. CONCLUSION

Having analyzed the DNSSEC misconfigurations of four zones (.bg, .br., co. and .se), our measurements show that implementing DNSSEC is not trivial and that misconfigurations exist in large numbers. From the 282,766 gathered domain names, only 7.93% show the intent to implement DNSSEC. Furthermore, over 4% of DNSSEC-enabled domains show a form of misconfiguration, emphasizing the configuration complexity. Where one might expect expiration of keys to be a significant means of misconfiguration, categorization of the errors found in the .se domains shows its impact to be neglectable. Instead, most DNSSEC-related misconfigurations are caused by an inconsistency concerning the DNSKEY, the main public key of a domain. In more than 99% of the cases of a missing DNSKEY or an error in the two-stage ZSK and KSK DNSKEY signing process, the error led to an unreachable domain and thus unreachable website or other network service. User availability shows to vary per type of misconfiguration. On average, 73.86% of the misconfigured domains appeared unreachable from a DNSSEC-aware resolver. Hence, organizations implementing DNSSEC need to frequently verify the correct configuration of DNSSEC parameters and perhaps implement mechanisms to guarantee continuous correctness of configuration and authentic availability of their resources.

Measurements on a second dataset, including domains from .bg, .br, .co, .com, .nl and .se, show that many more domains have the intent of implementing DNSSEC (over 20% excluding .com) but fail to communicate an essential part to their respective TLD. We show that a limited number of faulty configured webhosting parties cause most of the misconfigurations. Furthermore, we observe that where signature expirations were initially practically absent, now we exclusively find expired signatures. This implies that key invalidation almost always coincides with signature expiration, both results of a neglected zone. Finally, we show that a 3 and 2 week validity period between inception and expiration are most popular, followed by half-year and 1 month renewal periods.

9

CONCLUSION

In this dissertation, we have successfully proposed, implemented and evaluated improvements to the Internet architecture. Built upon the general consensus that we need improvements to the Internet architecture to contain its current and future functional complexity, we have made numerous improvements to the Future Internet Architecture areas of Software-Defined Networking and Information-Centric Networking and performed extensive measurements on the, more recent but related, Domain Name System's Security Extensions. In this final chapter, we present the main findings, recommendations and future work.

In chapter 3, *we have presented OpenNetMon, a POX OpenFlow controller module monitoring per-flow QoS metrics to enable fine-grained Traffic Engineering.* By polling flow source and destination switches at an adaptive rate, we obtain accurate results while minimizing the network and switch CPU overhead. The per-flow throughput and packet loss is derived from the queried flow counters. Delay, on the contrary, is measured by injecting probe packets directly into switch data planes, traveling the same paths, meaning nodes, links and buffers, and thus determining a realistic end-to-end delay for each flow. We have performed experiments on a hardware testbed simulating a small inter-office network, while loading it with traffic of highly bursty nature. The experimental measurements verify the accuracy of the measured throughput and delay for monitoring, while the packet loss gives a good estimate of possible service degradation. Thanks to the open source publication of its implementation, OpenNetMon has already been used and extended by others to support network failure support [97]. Where Software-Defined Networking (SDN), and in particular its popular implementation OpenFlow, are often presented as the holy grail in computer networking promising many improvements in the area of network management, OpenNetMon is the first study showing the feasibility and accuracy of network monitoring in SDNs.

Key to supporting high availability services in a network is the network's ability to quickly recover from failures. In chapter 4 *we have proposed a combination of protection in OpenFlow Software-Defined Networks through preconfigured backup paths and fast link failure detection.* Primary and secondary path pairs are configured via OpenFlow's Fast Failover Group Tables enabling path protection in case of link failure, deploying crankback routing when necessary. We configure per-link, in contrast to the regular per-path, Bidirectional Forwarding Detection (BFD) sessions to quickly detect link failures. This limits the number of traversing BFD sessions to be linear to the network size and minimizes session RTT, enabling us to further decrease the BFD sessions' window intervals to detect link failures faster. By integrating each interface's link status into Open vSwitch' Fast Failover Group Table implementation, we further optimize the recovery time. Where we reach recovery times as low as 3.3 ms, we show that sub 50 ms recovery times, an industrial rule of thumb, are feasible in SDNs without the necessity of implementing non-standardized components. The achieved recovery time is shown to be independent of path length and network size.

Complementary to the techniques implemented in chapter 4, in chapter 5 *we have derived, implemented and evaluated techniques and algorithms computing an all-to-all network forwarding configuration capable of circumventing link and node failures.* Our algorithms compute forwarding rules that include failure-disjoint backup paths offering preprogrammed protection from future topology failures. Through packet labelling we guarantee correct and loop-free detour forwarding. The labeling technique allows packets to return on primary paths unaffected by the failure and carries information used to upgrade link-failures to node-failures when applicable. Furthermore, we have implemented a proof-of-concept network controller that configures OpenFlow-based SDN switches according to this approach, showing that these types of failover techniques can be applied to production networks.

Compared to traditional link- or node-disjoint paths, our method shows to have significantly shorter primary and backup paths. Furthermore, we observe significantly less

crankback routing when backup paths are activated in our approach. Besides shorter paths, our approach outperforms traditional disjoint path computations in terms of respectively the needed Flow and Group table configuration entries by factors up to 20 and 1.9. Our approach allows packets that encounter a broken link or node along their path, to travel the second-to-shortest path to their destination taken from the node where the link or node failure is detected. We apply Software-Defined Networking, specifically the OpenFlow protocol, to configure computer networks according to the derived protection scheme, allowing them to continue functioning in case of a failure without (slow) controller intervention. Combined, the techniques presented in chapters 4 and 5 give a fast and correct failure protection scheme.

In chapter 6, we focus on Information-Centric Networking (ICN), a future Internet architecture paradigm focusing on OSI Layer 3 standardization of content distribution. In particular, we have investigated a prototype implementation of Named Data Network (NDN), implementing routing of content by hierarchical human-readable names describing the content. *To contain the complexity of global NDN routing tables, we have proposed to decouple context-related NDN names from routable names.* By using routable names aggregating to Autonomous Systems and their internal structure, the size of global routing tables becomes upper bounded by the number of ASes in a system. By using a LISP-style late binding of context-related names to location-aggregated names using the DNS directory service, we enable applications to communicate based on context-related names and maintain the benefits of location-agnostic routing. Our implementation of mapping and renaming context describing names to routable names, enables the NDN strategy layer to facilitate multipath routes and multihomed applications.

Realizing the need for centralized network control is not limited to the current Internet architecture, in chapter 7 *we proposed and designed a mechanism realizing application-specific forwarding schemes in OpenFlow-controlled Software-Defined Networks (SDNs).* Specifically, we have implemented a prototype for Named Data Networking and its implementation CCNx, an architecture which we also worked on in chapter 6. Compared to other application-specific SDN implementations, we argue that our implementation is architecturally less complex to implement, easier to extend and furthermore applicable to multiple application-specific forwarding schemes due to the stricter separation of functionalities. Furthermore, we have shown through experiments that this technique can be used to deploy application-specific forwarding in partially upgraded networks through tunneling. Since at introduction of an application-specific forwarding scheme not all forwarding devices need to be replaced or upgraded, it lowers the cost of introducing new forwarding schemes within one's network. With this implementation, we provide the tools to control and manage application-specific flows in SDNs.

Finally, in chapter 8 *we analyzed the Domain Name System's Security Extensions (DNS and DNSSEC) implementation.* DNS is a critical infrastructure to the Internet, a property that does not change using future Internet architectures such as SDN and ICN. In fact, the proposal of chapter 6 relies heavily on DNS. Intially analyzing the DNSSEC misconfigurations of four zones (.bg, .br., co. and .se), our measurements show that implementing DNSSEC is not trivial and that misconfigurations exist in large numbers. From the 282,766 gathered domain names, only 7.93% show the intent to implement DNSSEC.

Furthermore, over 4% of DNSSEC-enabled domains show a form of misconfiguration, emphasizing the configuration complexity. Where one might expect expiration of keys to be a significant means of misconfiguration, categorization of the errors found in the .se domains shows its impact to be neglectable. Instead, most DNSSEC-related misconfigurations are caused by an inconsistency concerning the DNSKEY, the main public key of a domain. In more than 99% of the cases of a missing DNSKEY or an error in the two-stage ZSK and KSK DNSKEY signing process, the error led to an unreachable domain and thus unreachable website or other network service. User availability shows to vary per type of misconfiguration. On average, 73.86% of the misconfigured domains appeared unreachable from a DNSSEC-aware resolver. Hence, organizations implementing DNSSEC need to frequently verify the correct configuration of DNSSEC parameters and perhaps implement mechanisms to guarantee continuous correctness of configuration and authentic availability of their resources.

Measurements on a second dataset, including domains from .bg, .br, .co, .com, .nl and .se, show that many more domains have the intent of implementing DNSSEC (over 20% excluding .com) but fail to communicate an essential part to their respective TLD. We show that a limited number of faulty configured webhosting parties cause most of these misconfigurations. Furthermore, we observe that where signature expirations were initially practically absent, now we exclusively find expired signatures. This implies that key invalidation almost always coincides with signature expiration, both results of a neglected zone. Finally, we show that a 3 and 2 week validity period between inception and expiration are most popular, followed by half-year and 1 month renewal periods.

Overall, we found that the researched computer network architectures provide great benefits to the Internet. *Through our work, we have significantly contributed to enable actual implementation of network architecture improvements.* Through abstract and theoretical analysis and practical improvement of these systems, we have solved complicated problems that previously prevented functional deployment. Although more work needs to be performed to achieve overall applicability, we have shown that these architectures can indeed provide a practical solution for the future Internet architecture.

9.1. FUTURE WORK

Based on the work in [96], *we further suggest to remove the overhead introduced by microflows in OpenNetMon* (see chapter 3), by categorizing microflows into one greater stream until recognized as an elephant flow. This prevents potential overloading of the controller by insignificant but possibly numerous flows. In future work, OpenNetMon can be used as an input generator for a responsive real-time QoS controller that recomputes and redistributes paths when necessary. OpenNetMon's measurements may further be used to perform multiple constraints routing, compute link-independent multiple paths based on actual usage allowing further exploitation of available bandwidth, or as a network health monitoring tool in general.

Our work on failure detection, recovery and protection of topology failures in SDNs, discussed in chapters 4 and 5, considers single link or node failures in unicast shortest-path routed networks. *We suggest researching the protection of multi-link and -node failures as well as strictly guaranteeing QoS constraints under failure.* Furthermore, one could extend the work to cover protection of multicast routed networks.

Overall, much of our work improves the robustness of computer networks, or considers the robustness of its proposed systems in general. Where chapters 4 and 5 focus on protection and recovery from topology failures, robustness of controller failure remains unnoticed. Although distributed network controller configurations are common, where 2 or more controllers cooperate to provide a control layer, their behavior in the event of failures is not well researched. Protocols exist to provide network state database synchronization, but little work is found on the detection and recovery times when failures occur. *We think additional protection from controller failures as well as an optimization of detection and recovery times can be achieved by implementing a proxy-like layer between switches and controllers.* Agnostic to the intent of the message, the proxy-like layer can first of all load-balance between distributed controllers and monitor the responsiveness of communication channels. Given that OpenFlow requests and replies share a unique identifier, the proxying layer may furthermore administer missing replies to requests and repeat lost requests to other available controllers authoritative for the specific network area.

9.2. RECOMMENDATIONS

Throughout our work on Software-Defined Networking (besides our work in chapters 3, 4, 5 and 7 also including our work in [216] and [217]) we found that the optional nature of OpenFlow's *Apply-Action* to apply packet-rewriting immediately instead of writing it to a list to be applied at output of a packet (the default *Write-Action*) often leads to implementational problems. In particular, absence of the Apply-Action on a switch complicates working with multiple VLAN tags and disables the use of between-table packet rewriting and matching functionality on the new values, both actions that are useful when trying to implement complex routing schemes. Hence, *implementing the Apply-Action should become compulsory in the OpenFlow specification.*

In chapter 8 we found that a limited number of webhosting parties cause most DNSSEC misconfigurations, possibly through subtennants deploying their own DNS servers. Besides registry operators sanctioning individual authoritative name servers misconfiguring DNSSEC, for example by declining zone recursion, the serving webhosting parties need to be addressed to solve the high overrepresentation found in their autonomous systems. Through public ranking and potential additional sanctions by their respective RIRs, *webhosting parties should be forced to properly implement their own DNS servers and educate their respective customers.*

REFERENCES

[1] C. Sutton, *Internet Began 35 Years Ago at UCLA ; Forum to Mark Anniversary Oct. 29,* http://newsroom.ucla.edu/releases/Internet-Began-35-Years-Ago-at-5464 (2004), Accessed: 2016-08-18.

[2] J. M. McQuillan and D. C. Walden, *The ARPA Network Design Decisions,* Computer Networks (1976) **1**, 243 (1977).

[3] F. E. Heart, R. E. Kahn, S. Ornstein, W. Crowther, and D. C. Walden, *The interface message processor for the ARPA computer network,* in *Proceedings of the May 5-7, 1970, spring joint computer conference* (ACM, 1970) pp. 551–567.

[4] B. Cosell, *IMP System change notification,* RFC 213 (1971).

[5] C. Botnet, *Internet census 2012—port scanning/0 using insecure embedded devices,* http://internetcensus2012.bitbucket.org/paper.html (2013), Accessed: 2016-10-13.

[6] M. Kende, *GLOBAL INTERNET REPORT 2015,* https://www.internetsociety.org/globalinternetreport/ (2015), Accessed: 2016-08-18.

[7] T. B. C. for Digital Development, *The State of Broadband 2015,* http://www.broadbandcommission.org/Documents/reports/bb-annualreport2015.pdf (2015), Accessed: 2016-08-18.

[8] I. Standardization, *ISO/IEC 7498-1: 1994 information technology–open systems interconnection–basic reference model: The basic model,* International Standard ISOIEC (1994).

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

[10] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, P. Crowley, C. Papadopoulos, L. Wang, B. Zhang, *et al.*, *Named Data Networking,* ACM SIGCOMM Computer Communication Review **44**, 66 (2014).

[11] N. Fotiou, P. Nikander, D. Trossen, and G. Polyzos, *Developing Information Networking Further: From PSIRP to PURSUIT,* BROADNETS (2010).

[12] C. Dannewitz and T. Biermann, *Prototyping a Network of Information,* IEEE Local Computer Networks **34** (2009).

[13] D. Katz and D. Ward, *Bidirectional Forwarding Detection (BFD),* RFC 5880 (Proposed Standard) (2010), Updated by RFCs 7419, 7880.

[14] W. H. L. L. P. W. Group, *IEEE Standard for Local and Metropolitan Area Networks Virtual Bridged Local Area Networks Amendment 5: Connectivity Fault Management,* IEEE 802.1ag (2007).

[15] S. Deering and R. Hinden, *Internet Protocol, Version 6 (IPv6) Specification,* RFC 2460 (Draft Standard) (1998), Updated by RFCs 5095, 5722, 5871, 6437, 6564, 6935, 6946, 7045, 7112.

[16] *IPv6 Statistics,* https://www.google.com/intl/nl/ipv6/statistics.html (2016), Accessed: 2016-08-18.

[17] J. Czyz, M. Allman, J. Zhang, S. Iekel-Johnson, E. Osterweil, and M. Bailey, *Measuring IPv6 Adoption,* ACM SIGCOMM Computer Communication Review **44**, 87 (2015).

[18] P. H. Salus and G. Vinton, *Casting the Net: From ARPANET to Internet and Beyond...* (Addison-Wesley Longman Publishing Co., Inc., 1995).

[19] J. Naughton, *A Brief History of the Future* (Weidenfeld & Nicolson, 2015).

[20] P. Van Mieghem, *Data Communications Networking,* 2nd ed. (Piet Van Mieghem, ISBN 978-94-91075-01-8, Delft, 2010).

[21] J. F. Kurose, *Computer Networking: A Top-Down Approach Featuring the Internet, 3/E* (Pearson Education India, 2005).

[22] J. Day, *Patterns in Network Architecture: A Return to Fundamentals* (Pearson Education, 2007).

[23] J. Simpson and E. S. Weiner, *Oxford English dictionary online,* Oxford: Clarendon Press. Retrieved March **6**, 2008 (1989).

[24] S. Bradner, *Key words for use in RFCs to Indicate Requirement Levels,* RFC 2119 (Best Current Practice) (1997).

[25] J. Klensin and D. Thaler, *Independent Submissions to the RFC Editor,* RFC 4846 (Informational) (2007), Updated by RFC 5744.

[26] S. Bradner, *The Internet Standards Process – Revision 3,* RFC 2026 (Best Current Practice) (1996), Updated by RFCs 3667, 3668, 3932, 3978, 3979, 5378, 5657, 5742, 6410, 7100, 7127, 7475.

[27] R. Housley, D. Crocker, and E. Burger, *Reducing the Standards Track to Two Maturity Levels,* RFC 6410 (Best Current Practice) (2011).

[28] A. S. Tanenbaum, *Computer Networks,* Vol. 3 (Prentice Hall New Jersey, 1996).

[29] R. M. Metcalfe and D. R. Boggs, *Ethernet: Distributed Packet Switching for Local Computer Networks,* Communications of the ACM **19**, 395 (1976).

[30] W. V. T. T. Force, *VLAN TAG,* IEEE 802.3ac (1998).

[31] D. Plummer, *Ethernet Address Resolution Protocol: Or Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware,* RFC 826 (INTERNET STANDARD) (1982), Updated by RFCs 5227, 5494.

[32] P. Srisuresh and M. Holdrege, *IP Network Address Translator (NAT) Terminology and Considerations,* RFC 2663 (Informational) (1999).

[33] J. Postel, *Internet Protocol,* RFC 791 (INTERNET STANDARD) (1981), Updated by RFCs 1349, 2474, 6864.

[34] E. Nordmark and R. Gilligan, *Basic Transition Mechanisms for IPv6 Hosts and Routers,* RFC 4213 (Proposed Standard) (2005).

[35] B. Carpenter and C. Jung, *Transmission of IPv6 over IPv4 Domains without Explicit Tunnels,* RFC 2529 (Proposed Standard) (1999).

[36] B. Carpenter and K. Moore, *Connection of IPv6 Domains via IPv4 Clouds,* RFC 3056 (Proposed Standard) (2001).

[37] C. Huitema, *Teredo: Tunneling IPv6 over UDP through Network Address Translations (NATs),* RFC 4380 (Proposed Standard) (2006), Updated by RFCs 5991, 6081.

[38] C. Bao, C. Huitema, M. Bagnulo, M. Boucadair, and X. Li, *IPv6 Addressing of IPv4/IPv6 Translators,* RFC 6052 (Proposed Standard) (2010).

[39] A. Durand, R. Droms, J. Woodyatt, and Y. Lee, *Dual-Stack Lite Broadband Deployments Following IPv4 Exhaustion,* RFC 6333 (Proposed Standard) (2011), Updated by RFC 7335.

[40] X. Zhou, M. Jacobsson, H. Uijterwaal, and P. Van Mieghem, *IPv6 delay and loss performance evolution,* International Journal of Communication Systems **21**, 643 (2008).

[41] J. Postel, *Transmission Control Protocol,* RFC 793 (INTERNET STANDARD) (1981), Updated by RFCs 1122, 3168, 6093, 6528.

[42] J. Postel, *User Datagram Protocol,* RFC 768 (INTERNET STANDARD) (1980).

[43] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications,* RFC 3550 (INTERNET STANDARD) (2003), Updated by RFCs 5506, 5761, 6051, 6222, 7022, 7160, 7164.

[44] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure, *TCP Extensions for Multipath Operation with Multiple Addresses,* RFC 6824 (Experimental) (2013).

[45] S. Barré, C. Paasch, and O. Bonaventure, *Multipath TCP: From Theory to Practice,* in *International Conference on Research in Networking* (Springer, 2011) pp. 444–457.

[46] L. R. Ford Jr, *Network Flow Theory,* Tech. Rep. (DTIC Document, 1956).

[47] R. Bellman, *On a Routing Problem,* Quarterly of Applied Mathematics **16**, 87 (1958).

[48] E. F. Moore, *The Shortest Path Through a Maze,* in *Proceedings of an International Symposium on the Theory of Switching* (Cambridge: Harvard University Press, 1959) pp. 285–292.

[49] A. Schrijver, *On the History of the Shortest Path Problem,* Doc. Math , 155 (2012).

[50] D. P. Bertsekas, *Network Optimization: Continuous and Discrete Models,* (1998).

[51] U. Pape, *Implementation and efficiency of Moore-algorithms for the shortest route problem,* Mathematical Programming **7**, 212 (1974).

[52] S. Pallottino, *Shortest-path methods: Complexity, interrelations and new propositions,* Networks **14**, 257 (1984).

[53] A. V. Goldberg and T. Radzik, *A heuristic improvement of the Bellman-Ford algorithm,* Applied Mathematics Letters **6**, 3 (1993).

[54] R. W. Floyd, *Algorithm 97: Shortest Path,* Communications of the ACM **5**, 345 (1962).

[55] S. Warshall, *A Theorem on Boolean Matrices,* Journal of the ACM **9**, 11 (1962).

[56] E. W. Dijkstra, *A Note on Two Problems in Connexion with Graphs,* Numerische Mathematik **1**, 269 (1959).

[57] M. Leyzorek, R. Gray, A. Johnson, W. Ladew, S. Meaker Jr, R. Petry, and R. Seitz, *Investigation of Model Techniques–First Annual Report–6 June 1956–1 July 1957–A Study of Model Techniques for Communication Systems,* Case Institute of Technology, Cleveland, Ohio (1957).

[58] M. L. Fredman and R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms,* Journal of the ACM (JACM) **34**, 596 (1987).

[59] P. Jakma and D. Lamparter, *Introduction to the Quagga Routing Suite,* IEEE Network **28**, 42 (2014).

[60] G. Malkin, *RIP Version 2,* RFC 2453 (INTERNET STANDARD) (1998), Updated by RFC 4822.

[61] Y. Rekhter, T. Li, and S. Hares, *A Border Gateway Protocol 4 (BGP-4),* RFC 4271 (Draft Standard) (2006), Updated by RFCs 6286, 6608, 6793, 7606, 7607, 7705.

[62] L. Bosack, *Method and apparatus for routing communications among computer networks,* (1992), US Patent 5,088,032.

[63] J. Moy, *OSPF Version 2,* RFC 2328 (INTERNET STANDARD) (1998), Updated by RFCs 5709, 6549, 6845, 6860, 7474.

[64] I. Standardization, *ISO/IEC 10589: 2002 Information technology – Telecommunications and information exchange between systems – Intermediate System to Intermediate System intra-domain routeing information exchange protocol for use in conjunction with the protocol for providing the connectionless-mode network service (ISO 8473),* International Standard ISOIEC (2002).

[65] J. Case, M. Fedor, M. Schoffstall, and J. Davin, *Simple Network Management Protocol (SNMP),* RFC 1157 (Historic) (1990).

[66] R. Enns, M. Bjorklund, J. Schoenwaelder, and A. Bierman, *Network Configuration Protocol (NETCONF),* RFC 6241 (Proposed Standard) (2011), Updated by RFC 7803.

[67] A. Doria, J. H. Salim, R. Haas, H. Khosravi, W. Wang, L. Dong, R. Gopal, and J. Halpern, *Forwarding and Control Element Separation (ForCES) Protocol Specification,* RFC 5810 (Proposed Standard) (2010), Updated by RFCs 7121, 7391.

[68] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, *OpenFlow: Enabling Innovation in Campus Networks,* ACM SIGCOMM Computer Communication Review **38(2)**, 69 (2008).

[69] B. S. Networks, *Floodlight OpenFlow Controller,* http://www.projectfloodlight.org/floodlight/ (2013), Accessed: 2016-10-13.

[70] M. McCauley, *About POX,* http://www.noxrepo.org/pox/about-pox/ (2016), Accessed: 2016-10-13.

[71] M. McCauley, *About NOX,* http://www.noxrepo.org/nox/about-nox/ (2016), Accessed: 2016-10-14.

[72] J. Medved, R. Varga, A. Tkacik, and K. Gray, *Opendaylight: Towards a Model-Driven SDN Controller Architecture,* in *Proceeding of IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks 2014* (2014).

[73] *Ryu SDN Framework,* http://osrg.github.io/ryu/ (2015), Accessed: 2015-12-22.

[74] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow, *et al.*, *ONOS: Towards an Open, Distributed SDN OS,* in *Proceedings of the third workshop on Hot topics in software defined networking* (ACM, 2014) pp. 1–6.

[75] R. Sherwood, G. Gibb, K.-K. Yap, G. Appenzeller, M. Casado, N. McKeown, and G. Parulkar, *Flowvisor: A Network Virtualization Layer,* OpenFlow Switch Consortium, Tech. Rep , 1 (2009).

[76] J. Vasseur and J. L. Roux, *Path Computation Element (PCE) Communication Protocol (PCEP),* RFC 5440 (Proposed Standard) (2009), Updated by RFC 7896.

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

[78] S. Jain, A. Kumar, S. Mandal, J. Ong, L. Poutievski, A. Singh, S. Venkata, J. Wanderer, J. Zhou, M. Zhu, *et al.*, *B4: Experience with a Globally-Deployed Software Defined WAN,* ACM SIGCOMM Computer Communication Review **43**, 3 (2013).

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

[80] Q. Zhao, Z. Ge, J. Wang, and J. Xu, *Robust Traffic Matrix Estimation with Imperfect Information: Making Use of Multiple Data Sources,* in *ACM SIGMETRICS Performance Evaluation Review*, Vol. 34(1) (ACM, 2006) pp. 133–144.

[81] C. Doerr, R. Gavrila, F. A. Kuipers, and P. Trimintzios, *Good Practices in Resilient Internet Interconnection,* ENISA Report (2012).

[82] C. Doerr and F. A. Kuipers, *All Quiet on the Internet Front?* Communications Magazine, IEEE **52**, 46 (2014).

[83] N. L. M. van Adrichem, C. Doerr, and F. A. Kuipers, *TUDelftNAS/SDN-OpenNetMon,* `https://github.com/TUDelftNAS/SDN-OpenNetMon` (2013), Accessed: 2016-10-13.

[84] B. Claise, *Cisco Systems NetFlow Services Export Version 9,* RFC 3954 (Informational) (2004).

[85] V. Sekar, M. K. Reiter, W. Willinger, H. Zhang, R. R. Kompella, and D. G. Andersen, *cSamp: A System for Network-Wide Flow Monitoring.* in *NSDI* (2008) pp. 233–246.

[86] P. Phaal, S. Panchen, and N. McKee, *InMon Corporation's sFlow: A Method for Monitoring Traffic in Switched and Routed Networks,* RFC 3176 (Informational) (2001).

[87] B. Huffaker, D. Plummer, D. Moore, and K. Claffy, *Topology discovery by active probing,* in *Applications and the Internet (SAINT) Workshops, 2002. Proceedings. 2002 Symposium on* (IEEE, 2002) pp. 90–96.

[88] C. Fraleigh, S. Moon, B. Lyles, C. Cotton, M. Khan, D. Moll, R. Rockell, T. Seely, and C. Diot, *Packet-level traffic measurements from the Sprint IP backbone,* Network, IEEE **17(7)**, 6 (2003).

[89] A. Tootoonchian, M. Ghobadi, and Y. Ganjali, *OpenTM: Traffic Matrix Estimator for OpenFlow Networks,* in *Passive and Active Measurement* (Springer, 2010) pp. 201–210.

[90] J. R. Ballard, I. Rae, and A. Akella, *Extensible and Scalable Network Monitoring Using OpenSAFE,* Proc. INM/WREN (2010).

[91] M. Yu, L. Jose, and R. Miao, *Software Defined Traffic Measurement with OpenSketch,* in *Proceedings 10th USENIX Symposium on Networked Systems Design and Implementation, NSDI*, Vol. 13 (2013).

[92] S. Hemminger, *Network Emulation with NetEm,* in *Linux Conf Au* (Citeseer, 2005) pp. 18–23.

[93] V. Paxson, M. Allman, J. Chu, and M. Sargent, *Computing TCP's Retransmission Timer,* RFC 6298 (Proposed Standard) (2011).

[94] R. van der Pol, M. Bredel, A. Barczyk, B. Overeinder, N. L. M. van Adrichem, and F. A. Kuipers, *Experiences with MPTCP in an intercontinental multipathed OpenFlow network,* in *Proceedings of the 29th Trans European Research and Education Networking Conference*, edited by D. Foster (TERENA, 2013).

[95] P. Van Mieghem and F. A. Kuipers, *Concepts of Exact QoS Routing Algorithms,* Networking, IEEE/ACM Transactions on **12**, 851 (2004).

[96] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, *DevoFlow: Scaling Flow Management for High-Performance Networks,* in *ACM SIGCOMM Computer Communication Review*, Vol. 41(4) (ACM, 2011) pp. 254–265.

[97] A. F. de la Cruz, J. P. Muñoz-Gea, P. Manzanares-Lopez, and J. Malgosa-Sanahuja, *Network Failures Support for Traffic Monitoring Mechanisms in Software-Defined Networks,* in *NOMS 2016-2016 IEEE/IFIP Network Operations and Management Symposium* (IEEE, 2016) pp. 691–694.

[98] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, *Fast Recovery in Software-Defined Networks,* in *IEEE Third European Workshop on Software Defined Networks (EWSDN'14)* (2014).

[99] B. Niven-Jenkins, D. Brungard, M. Betts, N. Sprecher, and S. Ueno, *Requirements of an MPLS Transport Profile,* RFC 5654 (Proposed Standard) (2009).

[100] L. Wang, R.-F. Chang, E. Lin, and J. C.-s. Yik, *Apparatus for link failure detection on high availability Ethernet backplane,* (2007), US Patent 7,260,066.

[101] D. Levi and D. Harrington, *Definitions of Managed Objects for Bridges with Rapid Spanning Tree Protocol,* RFC 4318 (Proposed Standard) (2005).

[102] V. Jacobson, *Congestion Avoidance and Control,* in *ACM SIGCOMM Computer Communication Review*, Vol. 18(4) (ACM, 1988).

[103] D. Clark, *Window and Acknowledgement Strategy in TCP,* RFC 813 (Historic) (1982), Obsoleted by RFC 7805.

[104] *OpenFlow Switch Specification version 1.1.0,* https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.1.0.pdf (2011), Accessed: 2015-12-22.

[105] M. Reitblatt, N. Foster, J. Rexford, and D. Walker, *Consistent Updates for Software-Defined Networks: Change you can believe in!* in *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (ACM, 2011) p. 7.

[106] *Open vSwitch GIT Web Front-End,* http://git.openvswitch.org (2014), Accessed: 2014.

[107] N. L. M. van Adrichem, B. J. van Asten, and F. A. Kuipers, *TUDelftNAS/SDN-OpenFlowRecovery,* https://github.com/TUDelftNAS/SDN-OpenFlowRecovery (2014), Accessed: 2016-10-13.

[108] R. Olsson, *pktgen the linux packet generator,* in *Proceedings of the Linux Symposium, Ottawa, Canada* (2005).

[109] S. Sharma, D. Staessens, D. Colle, M. Pickavet, and P. Demeester, *OpenFlow: Meeting Carrier-Grade Recovery Requirements,* Computer Communications (2012).

[110] J. Kempf, E. Bellagamba, A. Kern, D. Jocha, A. Takacs, and P. Skoldstrom, *Scalable Fault Management for OpenFlow,* in *Communications (ICC), 2012 IEEE International Conference on* (2012).

[111] E. Bellagamba, J. Kempf, and P. Skoldstrom, *Link Failure Detection and Traffic Redirection in an Openflow Network,* (2011), US Patent App. 13/111,609.

[112] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, *Effective Flow Protection in Open-Flow Rings,* in *Optical Fiber Communication Conference* (Optical Society of America, 2013).

[113] A. Sgambelluri, A. Giorgetti, F. Cugini, F. Paolucci, and P. Castoldi, *OpenFlow-Based Segment Protection in Ethernet Networks,* Optical Communications and Networking, IEEE/OSA Journal of **5** (2013).

[114] B. Lantz, B. Heller, and N. McKeown, *A Network in a Laptop: Rapid Prototyping for Software-Defined Networks,* in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks* (2010).

[115] N. L. M. van Adrichem, F. Iqbal, and F. A. Kuipers, *Backup rules in Software-Defined Networks,* in *Network Function Virtualization and Software Defined Network (NFV-SDN), 2016 IEEE Conference on* (IEEE, 2016).

[116] N. L. M. van Adrichem, F. Iqbal, and F. A. Kuipers, *Computing backup forwarding rules in Software-Defined Networks,* arXiv preprint arXiv:1605.09350 (2016).

[117] F. A. Kuipers, *An Overview of Algorithms for Network Survivability,* ISRN Communications and Networking **2012**, 24 (2012).

[118] J. Suurballe, *Disjoint paths in a network,* Wiley Networks **4**, 125 (1974).

[119] J. W. Suurballe and R. E. Tarjan, *A Quick Method for Finding Shortest Pairs of Disjoint Paths,* Netw. **14**, 325 (1984).

[120] D. A. Dunn, W. D. Grover, and M. H. MacGregor, *Comparison of K-Shortest Paths and Maximum Flow Routing for Network Facility Restoration,* IEEE J. Selec. Areas in Commun. **12**, 88 (1994).

[121] L. Guo and H. Shen, *On Finding Min-Min Disjoint Paths,* Springer Algorithmica **66**, 641 (2013).

[122] M. Shand and S. Bryant, *IP Fast Reroute Framework,* RFC 5714 (Informational) (2010).

[123] L. Ford and D. R. Fulkerson, *Flows in Networks* (Princeton Princeton University Press, 1962).

[124] B. Owens, *OpenFlow Switching Performance: Not All TCAM Is Created Equal,* http://packetpushers.net/openflow-switching-performance-not-all-tcam-is-created-equal/ (2013), Accessed: 2016-03-23.

[125] P. Erdős and A. Rényi, *On Random Graphs,* Publicationes Mathematicae Debrecen **6**, 290 (1959).

[126] B. M. Waxman, *Routing of multipoint connections,* IEEE Journal on Selected Areas in Communications **6**, 1617 (1988).

[127] A. R. Moral, P. Bonenfant, M. Krishnaswamy, and O. In, *The optical Internet: Architectures and protocols for the global infrastructure of tomorrow,* IEEE/ACM Transactions on Networking **39**, 152 (2001).

[128] M. Naldi, *Connectivity of Waxman topology models,* Elsevier Computer Communications **29**, 24 (2005).

[129] *OpenFlow Switch Specification version 1.3.5,* https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-switch-v1.3.5.pdf (2015), Accessed: 2015-12-22.

[130] D. A. Schult and P. Swart, *Exploring Network Structure, Dynamics, and Function using NetworkX,* in *Proceedings of the 7th Python in Science Conferences (SciPy 2008)*, Vol. 2008 (2008) pp. 11–16.

[131] N. L. M. van Adrichem, F. Iqbal, and F. A. Kuipers, *TUDelftNAS/SDN-OpenFlowBackupRules,* https://github.com/TUDelftNAS/SDN-OpenFlowBackupRules (2016), Accessed: 2016-03-23.

[132] R. Bhandari, *Optimal Physical Diversity Algorithms and Survivable Networks,* in *IEEE Symposium on Computers and Communications* (1997) pp. 433–441.

[133] J. Roskind and R. E. Tarjan, *Note on Finding Minimum-Cost Edge-Disjoint Spanning Trees.* Mathematics of Operations Research **10**, 701 (1985).

[134] M. Médard, S. Finn, R. Barry, and R. Gallager, *Redundant Trees for Preplanned Recovery in Arbitrary Vertex-Redundant or Edge-Redundant Graphs,* IEEE/ACM Transactions on Networking **7**, 641 (1999).

[135] C. M. Machuca, S. Secci, P. Vizarreta, F. Kuipers, A. Gouglidis, D. Hutchison, S. Jouet+, D. Pezaros+, A. Elmokashfi, P. Heegaard++, *et al.*, *Technology-related Disasters: A Survey towards Disaster-resilient Software Defined Networks,* in *Reliable Networks Design and Modeling (RNDM), 2016 8th International Workshop on* (IEEE, 2015) pp. 181–185.

[136] A. Capone, C. Cascone, A. Q. Nguyen, and B. Sanso, *Detour Planning for Fast and Reliable Failure Recovery in SDN with OpenState,* in *Design of Reliable Communication Networks (DRCN), 2015 11th International Conference on the* (IEEE, 2015) pp. 25–32.

[137] C. Cascone, L. Pollini, D. Sanvito, A. Capone, and B. Sansò, *SPIDER: Fault Resilient SDN Pipeline with Recovery Delay Guarantees,* arXiv preprint arXiv:1511.05490 (2015).

[138] G. Bianchi, M. Bonola, A. Capone, and C. Cascone, *OpenState: Programming Platform-independent Stateful OpenFlow Applications Inside the Switch,* ACM SIGCOMM Computer Communication Review **44**, 44 (2014).

[139] O. Tilmans and S. Vissicchio, *IGP-as-a-Backup for Robust SDN Networks,* in *Network and Service Management (CNSM), 2014 10th International Conference on* (IEEE, 2014) pp. 127–135.

[140] W. Braun and M. Menth, *Loop-Free Alternates with Loop Detection for Fast Reroute in Software-Defined Carrier and Data Center Networks,* Journal of Network and Systems Management , 1 (2016).

[141] N. L. M. van Adrichem and F. A. Kuipers, *Globally Accessible Names in Named Data Networking,* in *Proc. IEEE INFOCOM Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN)* (IEEE, 2013).

[142] 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 (2007).

[143] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh, *Named Data Networking (NDN) Project NDN-0001*, Tech. Rep. (Palo Alto Research Center, 2010).

[144] *Internet Grows to More Than 240 Million Domain Names in the Second Quarter of 2012,* https://investor.verisign.com/releasedetail.cfm?releaseid=710803 (2012), Accessed: 2016-10-13.

[145] *CIDR Report,* http://www.cidr-report.org/as2.0/ (2012), Accessed: 2012-12-09.

[146] R. A. Steenbergen and R. Mosher, *An Inconvenient Prefix: Is Routing Table Pollution Leading To Global Datacenter Warming,* (NANOG50, 2010).

[147] L. Wang, A. Hoque, C. Yi, A. Alyyan, and B. Zhang, *OSPFN: An OSPF Based Routing Protocol for Named Data Networking,* Palo Alto Reserch Centers, Tech. Rep (2012).

[148] Y. Wang, K. He, H. Dai, W. Meng, J. Jiang, B. Liu, and Y. Chen, *Scalable Name Lookup in NDN Using Effective Name Component Encoding,* in *Distributed Computing Systems (ICDCS), 2012 IEEE 32nd International Conference on* (2012).

[149] H. Yuan, T. Song, and P. Crowley, *Scalable NDN Forwarding: Concepts, Issues and Principles,* in *Computer Communications and Networks (ICCCN), 2012 21st International Conference on* (2012).

[150] P. Mockapetris, *Domain names - concepts and facilities,* RFC 1034 (INTERNET STANDARD) (1987), Updated by RFCs 1101, 1183, 1348, 1876, 1982, 2065, 2181, 2308, 2535, 4033, 4034, 4035, 4343, 4035, 4592, 5936.

[151] D. Farinacci, *draft-ietf-lisp-23 Locator/ID Separation Protocol (LISP),* (2012).

[152] D. Meyer and K. Patel, *BGP-4 Protocol Analysis,* RFC 4274 (Informational) (2006).

[153] T. Bates, Y. Rekhter, R. Chandra, and D. Katz, *Multiprotocol Extensions for BGP-4,* RFC 2858 (Proposed Standard) (2000), Obsoleted by RFC 4760.

[154] R. Editor, *Internet Official Protocol Standards,* RFC 5000 (Historic) (2008), Obsoleted by RFC 7100.

[155] A. Narayanan, S. Previdi, and B. Field, *draft-narayanan-icnrg-bgp-uri-00 BGP advertisements for content URIs,* (2012).

[156] J. D. Clercq, D. Ooms, S. Prevost, and F. L. Faucheur, *Connecting IPv6 Islands over IPv4 MPLS Using IPv6 Provider Edge Routers (6PE),* RFC 4798 (Proposed Standard) (2007).

[157] G. Lutostanski and B. Zhang, *NDN-Routing/ccnx-dhcp,* `https://github.com/NDN-Routing/ccnx-dhcp` (2011), Accessed: 2016-10-13.

[158] R. Cox, A. Muthitacharoen, and R. Morris, *Serving DNS using a Peer-to-Peer Lookup Service,* Peer-to-Peer Systems , 155 (2002).

[159] L. Jakab, A. Cabellos-Aparicio, F. Coras, D. Saucez, and O. Bonaventure, *LISP-TREE: A DNS Hierarchy to Support the LISP Mapping System,* IEEE Journal on Selected Areas in Communications **28**, 1332 (2010).

[160] V. Fuller and G. Wiley, *draft-fuller-lisp-ddt-04 LISP Delegated Database Tree,* (2012).

[161] D. Farinacci, V. Fuller, D. Meyer, and D. Lewis, *The Locator/ID Separation Protocol (LISP),* RFC 6830 (Experimental) (2013).

[162] N. L. M. van Adrichem, *TUDelftNAS/CCNx-RenamingFaces,* https://github.com/TUDelftNAS/CCNx-RenamingFaces (2012), Accessed: 2016-10-13.

[163] Palo Alto Research Center, *CCNx Protocol,* https://github.com/ProjectCCNx/ccnx/blob/master/doc/technical/CCNxProtocol.txt (2012), Accessed: 2016-10-13.

[164] N. L. M. van Adrichem, *TUDelftNAS/CCNx-DNSRelay,* https://github.com/TUDelftNAS/CCNx-DNSRelay (2012), Accessed: 2016-10-13.

[165] C. Yi, *NDN-Routing/ccnping,* https://github.com/NDN-Routing/ccnping (2016), Accessed: 2016-10-13.

[166] N. L. M. van Adrichem, *TUDelftNAS/CCNx-DHCNGP,* https://github.com/TUDelftNAS/CCNx-DHCNGP (2012), Accessed: 2016-10-13.

[167] Palo Alto Research Center, *CCNx Basic Name Conventions,* https://github.com/ProjectCCNx/ccnx/blob/master/doc/technical/NameConventions.txt (2012), Accessed: 2016-10-13.

[168] N. L. M. van Adrichem and F. A. Kuipers, *NDNFlow: Software-Defined Named Data Networking,* in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (IEEE, 2015).

[169] J. Suh, *OF-CCN: CCN over OpenFlow,* http://netlab.pkusz.edu.cn/wordpress/wp-content/uploads/2012/07/Content-Networking-over-OpenFlow.pdf (2012), Accessed: 2016-10-13.

[170] N. Blefari-Melazzi, A. Detti, G. Mazza, G. Morabito, S. Salsano, and L. Veltri, *An OpenFlowlow-based Testbed for Information Centric Networking,* Future Network & Mobile Summit , 4 (2012).

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

[172] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker, *P4: Programming Protocol-independent Packet Processors,* SIGCOMM Comput. Commun. Rev. **44**, 87 (2014).

[173] H. Song, *Protocol-Oblivious Forwarding: Unleash the Power of SDN through a Future-Proof Forwarding Plane,* in *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking* (ACM, 2013).

[174] Open Networking Foundation, *OF-PI: A Protocol Independent Layer,* https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/OF-PI__A_Protocol_Independent_Layer_for_OpenFlow_v1-1.pdf (2014), Accessed: 2016-10-13.

[175] Palo Alto Research Center, *CCNx,* http://www.ccnx.org/ (2012), Accessed: 2016-10-13.

[176] N. L. M. van Adrichem and F. A. Kuipers, *TUDelftNAS/SDN-NDNFlow,* https://github.com/TUDelftNAS/SDN-NDNFlow (2015), Accessed: 2016-10-13.

[177] N. L. M. van Adrichem, A. R. Lúa, X. Wang, M. Wasif, F. Fatturrahman, and F. A. Kuipers, *DNSSEC Misconfigurations: How incorrectly configured security leads to unreachability,* in *Intelligence and Security Informatics Conference (JISIC), 2014 IEEE Joint* (IEEE, 2014) pp. 9–16.

[178] N. L. M. Adrichem, N. Blenn, A. R. Lúa, X. Wang, M. Wasif, F. Fatturrahman, and F. A. Kuipers, *A measurement study of DNSSEC misconfigurations,* Security Informatics **4** (2015).

[179] P. Mockapetris, *Domain names - implementation and specification,* RFC 1035 (INTERNET STANDARD) (1987), Updated by RFCs 1101, 1183, 1348, 1876, 1982, 1995, 1996, 2065, 2136, 2181, 2137, 2308, 2535, 2673, 2845, 3425, 3658, 4033, 4034, 4035, 4343, 5936, 5966, 6604, 7766.

[180] D. Migault, C. Girard, and M. Laurent, *A Performance view on DNSSEC migration,* in *Network and Service Management (CNSM), 2010 International Conference on* (IEEE, 2010) pp. 469–474.

[181] S. M. Bellovin, *Using the Domain Name System for System Break-ins,* in *Proceedings of 5th USENIX UNIX Security Symposium, USENIX Association, Berkeley, CA* (1995).

[182] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, *DNS Security Introduction and Requirements,* RFC 4033 (Proposed Standard) (2005), Updated by RFCs 6014, 6840.

[183] F. Cambus, *StatDNS - DNS and Domain Name statistics.* http://www.statdns.com (2015), Accessed: 2015-03-02.

[184] ICANN and VeriSign Inc., *Root DNSSEC,* http://www.root-dnssec.org/ (2015), Accessed: 2015-03-02.

[185] ICANN, *DNSSEC Securing the Internet: Benefits to Companies and Consumers,* http://www.icann.org/en/news/in-focus/dnssec/dnssec-card-03dec12-en.pdf (), Accessed: 2015-03-02.

[186] C. Deccio, J. Sedayao, K. Kant, and P. Mohapatra, *Quantifying and Improving DNSSEC Availability,* in *Computer Communications and Networks (ICCCN), 2011 Proceedings of 20th International Conference on* (IEEE, 2011) pp. 1–7.

[187] D. York, *HBO NOW DNSSEC Misconfiguration Makes Site Unavailable From Comcast Networks (Fixed Now) Deploy360 Programme,* http://www.internetsociety.org/deploy360/blog/2015/03/hbo-now-dnssec-misconfiguration-makes-site-unavailable-from-comcast-networks-fixed-now/ (2015), Accessed: 2015-03-12.

[188] Comcast, *DNSSEC News,* http://dns.comcast.net/index.php/categories/listings/dnssec-news (), Accessed: 2015-07-22.

[189] J. Livingood, *Responsibility for Authoritative DNSSEC Mistakes,* Working Draft (2015).

[190] Comcast, *gov Failing DNSSEC Validation,* http://dns.comcast.net/index.php/entry/gov-failing-dnssec-validation-1 (), Accessed: 2015-07-22.

[191] ICANN, *Applicant Guidebook | ICANN New gTLDs.* http://newgtlds.icann.org/en/applicants/agb (), Accessed: 2015-03-02.

[192] Internet Infrastructure Foundation, *DNSSEC - The path to a secure domain.* https://www.iis.se/english/domains/tech/dnssec/ (), Accessed: 2015-03-02.

[193] SIDN, *SIDN Annual Report 2013,* https://www.sidn.nl/annualreport/dot-nl (2013), Accessed: 2015-03-04.

[194] IANA, *Root Zone DNSSEC Trust Anchors,* http://data.iana.org/root-anchors/ (2015), Accessed: 2015-03-02.

[195] Y. Schaeffer, B. Overeinder, and M. Mekking, *Flexible and Robust Key Rollover in DNSSEC,* in *Proceedings of the Workshop on Securing and Trusting Internet Names (SATIN 2012)* (NPL, 2012).

[196] J. Schlyter, *DNS Security (DNSSEC) NextSECure (NSEC) RDATA Format,* RFC 3845 (Proposed Standard) (2004), Obsoleted by RFCs 4033, 4034, 4035.

[197] B. Laurie, G. Sisson, R. Arends, and D. Blacka, *DNS Security (DNSSEC) Hashed Authenticated Denial of Existence,* RFC 5155 (Proposed Standard) (2008), Updated by RFCs 6840, 6944.

[198] G. Huston and G. Michaelson, *Measuring DNSSEC performance,* http://impossible.rand.apnic.net/ispcol/2013-05/dnssec-performance.pdf (2013), Accessed: 2015-07-22.

[199] W. Lian, E. Rescorla, H. Shacham, and S. Savage, *Measuring the Practical Impact of DNSSEC Deployment,* in *USENIX Security* (2013) pp. 573–588.

[200] S. Ariyapperuma and C. J. Mitchell, *Security vulnerabilities in DNS and DNSSEC,* in *Availability, Reliability and Security, 2007. ARES 2007. The Second International Conference on* (IEEE, 2007) pp. 335–342.

[201] J. Bau and J. C. Mitchell, *A Security Evaluation of DNSSEC with NSEC3,* IACR Cryptology ePrint Archive **2010**, 115 (2010).

[202] C. Deccio, *Quantifying the Impact of DNSSEC Misconfiguration,* in *DNS-OARC Workshop (2)* (2010).

[203] Verisign Labs and Internet Innovations, *Verisign jdnssec-tools,* http://www.verisignlabs.com/jdnssec-tools/ (), Accessed: 2015-03-02.

[204] NLnet Labs, *ldns,* http://www.nlnetlabs.nl/projects/ldns/ (2015), Accessed: 2015-03-02.

[205] C. Perez, *dnsrecon,* https://github.com/darkoperator/dnsrecon (2014), Accessed: 2014-01.

[206] Verisign Labs and Internet Innovations, *DNSSEC Analyzer,* http://Dnssec-debugger.verisignlabs.com (), Accessed: 2015-03-02.

[207] Google, *Public DNS - Google Developers,* https://developers.google.com/speed/public-dns/ (2015), Accessed: 2015-03-02.

[208] BIND, *dig (domain information groper),* ftp://ftp.isc.org/isc/bind9/cur/9.10/doc/arm/man.dig.html (2016), Accessed: 2016-10-13.

[209] Anonymous, *DNS Census 2013,* https://dnscensus2013.neocities.org/ (2013), Accessed: 2016-10-13.

[210] D. Kalchev, *DNSSEC Implementation in .BG,* http://www.cctld.ru/files/pdf/Presentations_09-09/17-45_dnssec%20Bulgaria.pdf (2008), Accessed: 2014-01.

[211] Brasil Dominios, *Home - Dominios - Estatisticas,* http://registro.br/estatisticas.html (2014), Accessed: 2014-01.

[212] G. Romero, *DNSSEC Deployment in .CO,* in *ICANN48 DNSSEC Workshop* (2013).

[213] Internet Infrastructure Foundation, *Growth .se | .SE,* https://www.iis.se/english/domains/domain-statistics/growth/ (), Accessed: 2014-01.

[214] T. Cymru, *IP to ASN Mapping,* https://www.team-cymru.org/IP-ASN-mapping.html (2015), Accessed: 2015-03-10.

[215] R. Elz and R. Bush, *Serial Number Arithmetic,* RFC 1982 (Proposed Standard) (1996).

[216] M. P. V. Manthena, N. L. van Adrichem, C. van den Broek, and F. Kuipers, *An SDN-based Architecture for Network-as-a-Service,* in *Network Softwarization (NetSoft), 2015 1st IEEE Conference on* (IEEE, 2015) pp. 1–5.

[217] N. L. M. v. A. Hedi Krishna and F. A. Kuipers, *Providing Scalable Bandwidth Guarantees with OpenFlow,* in *Communications and Vehicular Technology in the Benelux (SCVT), 2016 IEEE Symposium on* (IEEE, 2017).

ACKNOWLEDGEMENTS

When first approached by Fernando whether I was interested in doing a PhD, I had no clue how such a long research project would look like. Hence, my first reaction being "I am not sure whether I am the right person for that." I think I am the only person alive whose potential PhD supervisor gave him 3 weeks consideration time with the advice to inform himself so he can better motivate the answer. A classical example of rock-solid advice by Fernando, of which, after a few weeks of researching into what a PhD implies and careful deliberation, were many to follow. 4 years seem like a long time to commit oneself, though, as Piet frequently and truthfully reminded me, those years fly by! Piet and Fernando, thank you for giving me the opportunity to work with you, my years in Delft have given me a broader view on life and on the world.

During my PhD, I have worked with many people and enjoyed support in many ways from an equal number of people. I would like to thank you all individually, instead I will have to find rest in these finite and probably incomplete acknowledgements. If you feel you have been omitted, please accept my apologies and rest assured that I do value your contribution.

First of all, I would like to show gratitude to all my direct colleagues from the NAS group at TU Delft. Aleksandr, Annalisa, Bo, Christian, Chuan, Cong, Dajie, Dongchao, Ebisa, Edgar, Evangelos, Farabi, Fernando, Hale, Huijuan, Javier, Jil, Luxing, Marcus, Marloes, Martijn, Nico, Norbert, Piet, Qiang, Rani, Remco, Rob, Rogier, Ruud, Song, Stojan, Wendy, Wynand, Xiangrong, Yakup, Zhidong and Zongwei, thank you all for making the NAS group such a close group to stay in. My special greetings go to my fellow clusterbusters, Christian, Marcus, Norbert and Ruud, like you, I hope I never have to fix a broken TORQUE cluster ever again. Cheers to my Friday afternoon[1] /Pub drinking buddies Christian, Jil, Norbert and Ruud, you have been an extremely enjoyable clique to hang out with.

Throughout my PhD time I have worked with many pleasant people. In particular, I would like to thank the people who made it possible for me to perform research. Bart, Robbert and Ruud, thank you very much for your technical support and collegiality, I greatly admire your team spirit and enthousiasm. Aad and Peter, you are the best datacenter managers I have met, your professional approach made me feel that "my" servers were in good hands. Ronald, thank you for sharing your knowledge and working with me at the testbed facilities at your disposal. Frans and Lolke, thank you for thinking outside of the box when problems needed unconventional networking solutions. To the Master students who got to enjoy my personal supervision, Ben, Ficky, Harmjan, Hedi, Jorik, Prashanth, Sarina, Sulabh and Vassil, I wish you all the best. Furthermore, I would like to thank the members of my PhD commitee for their time and effort, your comments and questions are well appreciated and have improved the quality of this dissertation.

[1] Though not exclusively Friday afternoon.

I would like to thank all my teachers throughout my education, but especially my teachers from the Prins Johan Friso, Wolfert van Borselen and Technische Hogeschool Rijswijk. Your devotion has schooled me and taught me to acquire the academic skills and competences I own today.

Throughout my life I have been surrounded by a steady number of close friends, thank you for being there. I trust you know who you individually are when you are reading this. If you are reading this from the audience at my defence, please think about the most absurd thing you have encountered together with me.

My foremost gratitude goes towards my family. I know my decisions and moods often seem a bit tangled to you, and perhaps they are, but I wouldn't know where in the world I would be without you. Mum and Dad, I think I can say you successfully raised a giant, thank you for your unconditional support. Dion, special thanks to you for sticking together through our studies and ITCall. I would do it all over again without changing a thing. Arjun, thank you for your pragmatic approach to solving problems, often you offer me a point of view I had not thought of. I love you all equally.

Additionally, I am blessed with a very caring family-in-law, thank you for your unending hospitality. Remy, thank you for designing the cover art.

Finally, I would like to thank my lovely wife Tine for all her love, care and devotion. I hope I may love you as much as I do today for a very long time to come.

BIOGRAPHY

Nicolaas Leonardus Maria VAN ADRICHEM
Niels L. M. VAN ADRICHEM
Niels VAN ADRICHEM
Niels

Niels van Adrichem was born on the 14th of November 1985 in Rotterdam, the Netherlands, both a happy day and a beautiful city. In line with his ancestry of strong family names, he turned out to become a somewhat curious, slightly hyperactive, wee-bit distracted, now-and-then worrying, heretically stubborn as a mule, though relatively intelligent, always honest, frequently spontaneous, tendingly compassionate and fairly creative, but most-of-all giant of a man of over 2 meters tall. With feet of European shoe size 48 2/3, his grandmother-in-law does not dare not ask him how large his feet are, experiencing utmost awe comparing her shoe size while curiously informing whether it is cold up there.

Niels his early education started at the Prins Johan Friso primary school in 1997, where he was taught the necessary skills to attend University-prepatory school. In extension, he followed a bilingual secondary school program at the Wolfert van Borselen. Over here he learned to combine his creativity with technical skills and was taught a near-to-native level of English through the International Baccalaureate program by the time of 2004.

After a short leave-year in 2005 when he had fooled himself into thinking mechanical engineering would suit his profession, Niels soon moved his educational interests towards computer science and engineering related areas. In 2009, after receiving his Bachelor of Engineering in Electrical Engineering at the Technische Hogeschool Rijswijk and while founding his own company ITCall together with his brother Dion, Niels decided to pursue an academic education and graduated in 2012 receiving a Master of Science in Computer Engineering from the Delft University of Technology (TU Delft). Soon after, he started working on his PhD in the Network Architectures and Services group at TU Delft, from the resulting dissertation you are currently reading this biography.

In his work, Niels addresses computer networking problems that show both practical relevance and theoretical complexity. The results of which, besides academic recognition through peer-reviewed publications and research projects for partners from industry, also show in a significant number of open-source code contributions.

In his personal life, Niels often continues employing his technical skills in benefit of his household. As a true DIYer, there may soon not be a single wire in the house untouched or digitally controllable by remote or smartphone. A property not always understood and accordingly valued but almost always tolerated by his loving wife Tine.

List of Publications

12. **N.L.M. van Adrichem, F. Iqbal and F.A. Kuipers**, *Backup rules in Software-Defined Networks*, IEEE Conference on Network Function Virtualization and Software Defined Networks (IEEE NFV-SDN 2016), IEEE, Palo Alto, California, United States, 7-9 November 2016

11. **H. Krishna, N.L.M. van Adrichem and Fernando A. Kuipers**, *Providing Scalable Bandwidth Guarantees with OpenFlow*, IEEE Symposium on Communications and Vehicular Technologies (IEEE SCVT 2016), IEEE , Mons, Belgium, 22 November 2016

10. **N.L.M. van Adrichem, F. Iqbal and F.A. Kuipers**, *Computing Backup Forwarding Rules in Software-Defined Networks*, arXiv:1605.09350 (30 May 2016)

9. **Niels L. M. van Adrichem, Norbert Blenn, Antonio Reyes Lúa, Xin Wang, Muhammad Wasif, Ficky Fatturrahman and Fernando A. Kuipers**, *A measurement study of DNSSEC misconfigurations*, Security Informatics, 4:8, SpringerOpen (19 October 2015, invited article, Dutch Cyber Security best Research paper Award [DCSRA] nomination)

8. **Niels L. M. van Adrichem and Fernando A. Kuipers**, *NDNFlow: Software-Defined Named Data Networking*, IEEE Conference on Network Softwarization (NetSoft 2015), IEEE, London, United Kingdom, 13-17 April 2015

7. **M. P. V. Manthena, Niels L. M. van Adrichem, Casper van den Broek, and Fernando A. Kuipers**, *An SDN-based Architecture for Network-as-a-Service*, IEEE Conference on Network Softwarization (NetSoft 2015), London, United Kingdom, 13-17 April 2015

6. **Niels L. M. van Adrichem, A. Reyes Lúa, X. Wang, M. Wasif, F. Fatturrahman and Fernando A. Kuipers**, *DNSSEC Misconfigurations: How incorrectly configured security leads to unreachability*, IEEE Joint Intelligence and Security Informatics Conference (JISIC 2014), IEEE, the Hague, the Netherlands, 24-26 September 2014 (best paper award nomination)

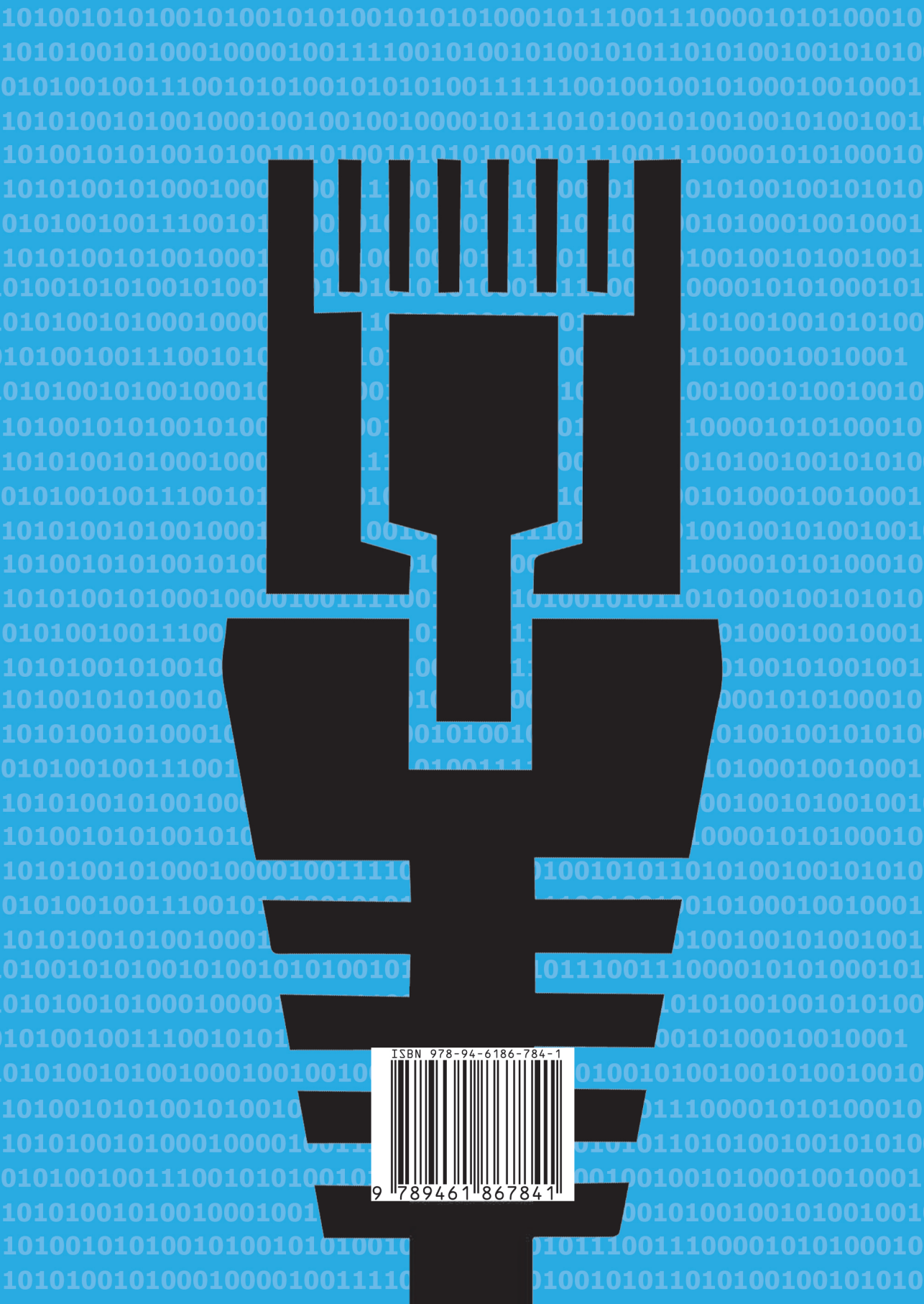
5. **Niels L. M. van Adrichem, Benjamin J. van Asten and Fernando A. Kuipers**, *Fast Recovery in Software-Defined Networks*, European Workshop on Software Defined Networking (EWSDN 2014), IEEE, Budapest, Hungary, 1-3 September 2014

4. **B. van Asten, N. van Adrichem, and F.A. Kuipers**, *Scalability and Resilience of Software-Defined Networking: An Overview*, arXiv:1408.6760 (28 August 2014)

3. **Niels L. M. van Adrichem, Christian Doerr and Fernando A. Kuipers**, *OpenNetMon: Network monitoring in OpenFlow Software-Defined Networks*, IEEE/IFIP Network Operations and Management Symposium (NOMS 2014), IEEE/IFIP, Poland, Kraków, 5-9 May 2014

2. **Ronald van der Pol, Michael Bredel, Artur Barczyk, Benno Overeinder, Niels van Adrichem and Fernando Kuipers**, *Experiences with MPTCP in an intercontinental OpenFlow network*, Trans European Research and Education Networking Conference (TNC2013), TERENA the Netherlands, Maastricht, 3-6 June 2013

1. **Niels L. M. van Adrichem and Fernando A. Kuipers**, *Globally Accessible Names in Named Data Networking*, IEEE International Workshop on Emerging Design Choices in Name-Oriented Networking (NOMEN 2013), workshop of IEEE INFOCOM, IEEE, Italy, Turin, 19 April 2013


ISBN 978-94-6186-784-1
9 789461 867841