Extension and Evaluation of Routing with Hints in NetInf
Information-Centric Networking

Christos Christodoulos Tsakiroglou

Master’s thesis
Supervisor: Dr. Bengt Ahlgren, SICS
Examiner: Prof. Peter Sjödin, KTH

Stockholm, 21st November, 2014
Abstract

Content distribution is the main driver for Internet traffic growth. The traditional networking approach, focused on communication between hosts, cannot efficiently cope with the evolving problem. Thus, information-centric networking (ICN) is a research area that has emerged to provide efficient content distribution solutions by shifting the focus from connecting hosts to connecting information. This new architecture defines named data objects as the main network entity and is based on a publish/subscribe-like paradigm combined with pervasive caching. An open challenge is a scalable routing mechanism for the vast number of objects in the global network.

The Network of Information (NetInf) is an ICN architecture that pursues a scalable and efficient global routing mechanism using name resolution service, which maps the content publisher to a set of routing hints. The routing hints aid at forwarding the request towards a source of the content, based on a priority system. Topological aggregation on the publisher authority names and on the location-independent routing hints provide a scalable solution.

This thesis extends the routing and forwarding scheme by forming partially ordered sets of routing hints, in order to prevent routing loops. In addition, the system has to meet the routing scalability and high performance requirements of a global solution. A dynamic routing service is investigated through an interface to open source routing software, which provides implementations of the existing routing protocols, in particular Quagga with BGP. The experimental evaluation of the forwarding scheme measures the execution times of the functions in the forwarding process by collecting timestamps. The results identify the most expensive functions and potential bottlenecks under high workload.
Acknowledgements

I would like to profoundly thank my supervisor at the Swedish Institute of Computer Science (SICS), Bengt Ahlgren. His guidance, support and feedback during the course of this project was invaluable. I would also like to express my gratitude to my academic supervisor from KTH Royal Institute of Technology, Peter Sjödin, for providing constructive feedback and guidance.

I want to thank all the fellow colleagues and friends at SICS with whom I shared interesting discussions, coffee breaks and lunch. I would also like to thank EIT ICT Labs for their support during this master program.

I would like to express my sincere thanks to my family for their constant and unconditional encouragement and support. Finally, a thanks goes to my friends for their support during my studies.

Christos Christodoulou Tsakiroglou
## Contents

Abstract i  
Acknowledgements ii  
Contents iii  
List of Figures vi  
List of Tables viii  
List of Acronyms ix  

1 Introduction 1  
1.1 Problem statement 2  
1.2 Purpose 3  
1.3 Goals 3  
  1.3.1 Ethics 4  
  1.3.2 Sustainability 4  
1.4 Methodology 4  
  1.4.1 Research methods 5  
  1.4.2 Development methods 5  
  1.4.3 Evaluation methods 5  
1.5 Delimitations 6  
1.6 Thesis outline 6  

2 Literature Study 8  
2.1 Information Centric Networking 8  
  2.1.1 Design commonalities and differences 9  
  2.1.2 Benefits and challenges 11  
  2.1.3 Routing and forwarding 12  
2.2 NetInf architecture 13  
2.3 Routing and forwarding in NetInf 15  
  2.3.1 Partial orders 17  
  2.3.2 Related work 18
5.4.1 Comparison with NNRP 80

6 Conclusion and future work 82
   6.1 Conclusion 82
   6.2 Future work 84

Bibliography 86

Appendices 91
   A System specifications 92
   B Experiment topology 94
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>ICN communication mechanisms</td>
<td>9</td>
</tr>
<tr>
<td>2.2</td>
<td>Name-based routing in NetInf</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>NRS routing in NetInf</td>
<td>16</td>
</tr>
<tr>
<td>2.4</td>
<td>Hasse diagram of a poset</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Priority-based routing system in NetInf</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Routing loops in priority-based routing system in NetInf due to NRS and routing tables</td>
<td>26</td>
</tr>
<tr>
<td>3.3</td>
<td>Routing loops in priority-based routing system in NetInf due to priority levels</td>
<td>26</td>
</tr>
<tr>
<td>3.4</td>
<td>Hasse diagram of poset of routing hints</td>
<td>29</td>
</tr>
<tr>
<td>3.5</td>
<td>NetInf GET request processing</td>
<td>37</td>
</tr>
<tr>
<td>3.6</td>
<td>NetInf GET request forwarding process</td>
<td>38</td>
</tr>
<tr>
<td>3.7</td>
<td>NetInf inter-domain request forwarding</td>
<td>41</td>
</tr>
<tr>
<td>4.1</td>
<td>NetInf router tables and Quagga</td>
<td>46</td>
</tr>
<tr>
<td>4.2</td>
<td>Interface between NetInf and Quagga components</td>
<td>48</td>
</tr>
<tr>
<td>4.3</td>
<td>NetInf overlay network topology</td>
<td>49</td>
</tr>
<tr>
<td>5.1</td>
<td>Experiment topology</td>
<td>57</td>
</tr>
<tr>
<td>5.2</td>
<td>Experiment posets of routing hints</td>
<td>59</td>
</tr>
<tr>
<td>5.3</td>
<td>Gateway scenario: Execution time of <code>forwarding table lookup</code> function versus forwarding table size, for authority A (2 hints)</td>
<td>62</td>
</tr>
<tr>
<td>5.4</td>
<td>Gateway scenario: Execution time of <code>forwarding table lookup</code> function versus forwarding table size, for authority C (4 hints)</td>
<td>62</td>
</tr>
<tr>
<td>5.5</td>
<td>Gateway scenario: Execution time of <code>NetInf GET request</code> function versus NDO size</td>
<td>64</td>
</tr>
<tr>
<td>5.6</td>
<td>Gateway scenario: Execution time of <code>forward request</code> function versus NDO size</td>
<td>65</td>
</tr>
<tr>
<td>5.7</td>
<td>Gateway scenario: Execution time of <code>HTTP CL functions</code> versus NDO size</td>
<td>66</td>
</tr>
</tbody>
</table>
5.8 Gateway scenario: Execution time of HTTP CL functions with caching of the NDO versus NDO size [steps 10a, 10b, 11 of section 5.1.4] .................................................. 67
5.9 Gateway scenario: Execution time of all forwarding functions, median values over all authorities (2-6 hints) ................. 69
5.10 Gateway scenario: Execution time of forwarding process functions, median values over all authorities (2-6 hints) [steps 4-8 of section 5.1.4] ........................................ 69
5.11 Gateway scenario: Execution time of forwarding table lookup function versus poset of hints (authority) [step 6 of section 5.1.4] .................................................... 71
5.12 Gateway scenario: Execution time of update hints function versus poset of hints (authority) [step 8 of section 5.1.4] ..... 71
5.13 Gateway scenario: Execution time of DNS (NRS) lookup function versus poset of hints (authority) [step 5 of section 5.1.4] 72
5.14 Gateway scenario: Execution time of forward request function versus number of request processes and inter-arrival time [step 9 of section 5.1.4] .................................................. 74
5.15 Gateway scenario: Execution time of GET response function versus number of request processes and inter-arrival time [step 11 of section 5.1.4] ...................................................... 74
5.16 Gateway scenario: Execution time of forwarding table lookup function versus number of request processes and inter-arrival time [step 6 of section 5.1.4] .................................................. 75
5.17 Experiment topology: In-network scenario .......................... 76
5.18 In-network scenario: Execution time of forwarding process functions, median values over all authorities (2-6 hints) [steps 4-8 of section 5.1.4] ........................................ 76
5.19 In-network scenario: Execution time of extract hints function versus poset of hints (authority) [step 4 of section 5.1.4] .... 77
5.20 In-network scenario: Execution time of NRS lookup with merging of hints function versus poset of hints (authority) [step 5 of section 5.1.4] .................................................. 78

B.1 Physical layer experiment topology ................................. 94
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Partially ordered set of routing hints</td>
<td>28</td>
</tr>
<tr>
<td>3.2</td>
<td>NRS Cache</td>
<td>30</td>
</tr>
<tr>
<td>3.3</td>
<td>Hint forwarding table</td>
<td>33</td>
</tr>
<tr>
<td>3.4</td>
<td>Next hop table</td>
<td>33</td>
</tr>
<tr>
<td>3.5</td>
<td>Ni name forwarding table</td>
<td>33</td>
</tr>
<tr>
<td>3.6</td>
<td>Recommended scheme for assignment of priority levels</td>
<td>43</td>
</tr>
<tr>
<td>4.1</td>
<td>FPM Header</td>
<td>49</td>
</tr>
<tr>
<td>4.2</td>
<td>GRE tunnels at NetInf router 10.0.0.1</td>
<td>50</td>
</tr>
<tr>
<td>5.1</td>
<td>Parameters for forwarding table size experiment</td>
<td>61</td>
</tr>
<tr>
<td>5.2</td>
<td>Parameters for NDO size experiment</td>
<td>63</td>
</tr>
<tr>
<td>5.3</td>
<td>Parameters for routing hints experiment</td>
<td>68</td>
</tr>
<tr>
<td>5.4</td>
<td>Parameters for request rate experiment</td>
<td>73</td>
</tr>
<tr>
<td>A.1</td>
<td>Systems specifications</td>
<td>92</td>
</tr>
<tr>
<td>A.2</td>
<td>Switch specifications</td>
<td>93</td>
</tr>
</tbody>
</table>
# List of Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AS</td>
<td>Autonomous System.</td>
</tr>
<tr>
<td>BGP</td>
<td>Border Gateway Protocol.</td>
</tr>
<tr>
<td>BIRD</td>
<td>Bird Internet Routing Daemon.</td>
</tr>
<tr>
<td>CCN</td>
<td>Content-Centric Networking.</td>
</tr>
<tr>
<td>CDN</td>
<td>Content Delivery Network.</td>
</tr>
<tr>
<td>CL</td>
<td>Convergence Layer.</td>
</tr>
<tr>
<td>COMET</td>
<td>COntent Mediator architecture for content-aware nETworks.</td>
</tr>
<tr>
<td>DFZ</td>
<td>Default Free Zone.</td>
</tr>
<tr>
<td>DNS</td>
<td>Domain Name System.</td>
</tr>
<tr>
<td>DONA</td>
<td>Data-Oriented Network Architecture.</td>
</tr>
<tr>
<td>FIB</td>
<td>Forwarding Information Base.</td>
</tr>
<tr>
<td>FPM</td>
<td>Forwarding Plane Manager.</td>
</tr>
<tr>
<td>GRE</td>
<td>Generic Routing Encapsulation.</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol.</td>
</tr>
<tr>
<td>ICN</td>
<td>Information-Centric Networking.</td>
</tr>
<tr>
<td>ICNRG</td>
<td>Information Centric Networking Research Group.</td>
</tr>
<tr>
<td>IETF</td>
<td>Internet Engineering Task Force.</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol.</td>
</tr>
<tr>
<td>IRTF</td>
<td>Internet Research Task Force.</td>
</tr>
<tr>
<td>JSON</td>
<td>JavaScript Object Notation.</td>
</tr>
<tr>
<td>NBR</td>
<td>Name-Based Routing.</td>
</tr>
<tr>
<td>NDN</td>
<td>Named-Data Networking.</td>
</tr>
<tr>
<td>NDO</td>
<td>Named Data Object.</td>
</tr>
<tr>
<td>NetInf</td>
<td>Network of Information.</td>
</tr>
<tr>
<td>ni</td>
<td>named information.</td>
</tr>
<tr>
<td>NLSR</td>
<td>Named-data Link State Routing.</td>
</tr>
<tr>
<td>NRS</td>
<td>Name Resolution Service.</td>
</tr>
<tr>
<td>OLSA</td>
<td>Opaque Link State Advertisement.</td>
</tr>
<tr>
<td>OSPF</td>
<td>Open Shortest Path First.</td>
</tr>
<tr>
<td>OSPFN</td>
<td>OSPF for Named-data.</td>
</tr>
<tr>
<td>P2P</td>
<td>peer-to-peer.</td>
</tr>
<tr>
<td>PKI</td>
<td>Public-Key Infrastructure.</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>poset</td>
<td>Partially Ordered Set.</td>
</tr>
<tr>
<td>PSIRP</td>
<td>Publish-Subscribe Internet Routing Paradigm.</td>
</tr>
<tr>
<td>PURSUIT</td>
<td>Publish-Subscribe Internet Technologies.</td>
</tr>
<tr>
<td>RIB</td>
<td>Routing Information Base.</td>
</tr>
<tr>
<td>RIP</td>
<td>Routing Information Protocol.</td>
</tr>
<tr>
<td>SAIL</td>
<td>Scalable &amp; Adaptive Internet solutions.</td>
</tr>
<tr>
<td>SDN</td>
<td>Software-Defined Networking.</td>
</tr>
<tr>
<td>Siena</td>
<td>Scalable Internet Event Notification Architectures.</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol.</td>
</tr>
<tr>
<td>TRIAD</td>
<td>Translating Relaying Internet Architecture integrating Active Directories.</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol.</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator.</td>
</tr>
<tr>
<td>VoD</td>
<td>Video on Demand.</td>
</tr>
<tr>
<td>XIA</td>
<td>eXpressive Internet Architecture.</td>
</tr>
<tr>
<td>XORP</td>
<td>eXtensible Open Router Platform.</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Information-Centric Networking (ICN) is a research area in the field of future Internet architecture that started over a decade ago but has gained more popularity during the last five years. As computer networks and the Internet evolve, new approaches are proposed to provide solutions for emerging problems, in this case efficient and scalable content distribution. Internet traffic keeps increasing and according to forecasts [1], global IP traffic will surpass one zettabyte (1000 exabytes) in 2016 and reach 1.6 zettabytes by 2018. Video is the dominant form of traffic, as 80% to 90% of the consumer traffic will consist of forms of video distribution, such as TV, Video on Demand (VoD), Internet video and peer-to-peer (P2P). Nowadays there are typically different proprietary technologies used for each application, which cannot directly communicate using the same principles and which are usually implemented as an overlay of the existing technologies, leading to inefficiencies [2].

The Internet was initially designed to provide a means of communication between named hosts, with hosts being the main entity of the network. The current host-centric approach suggests that knowing the name (or address) of a computer enables a client to reach a host and communicate with it. The new information-centric approach puts information (which is also referred to as content, or data, by some proposals) as the basic entity of the network, suggesting that hosts are now able to request and receive uniquely named information from the network [3]. Thus, the host-to-host communication is replaced by a host-to-network communication, decoupling the senders from the receivers. In other words, the focus shifts from connecting hosts to connecting information. These pieces of information are called Named Data Objects (NDOs), they are the main network abstraction and they can be audio, video, text, image and any other kind of data files. The basic communication patterns include senders publishing the NDOs and receivers requesting them. Every network node, whether in the core or edge network, can potentially cache an NDO and then satisfy requests for this NDO itself.
The expected benefits of this new architecture include improved network efficiency and scalability, which translates into lower response latency and simpler network load balancing, by exploiting the trade-off between storage and bandwidth. The content can be self-certified without the need of external authorities. Furthermore, mobility support and ad hoc communication without infrastructure provide more robustness in challenging communication environments [2, 4].

Network of Information (NetInf) [5] is an ICN architecture that aims at global-scale communication, a challenging task considering the vast number of data objects in comparison to the number of hosts. It was developed during the course of the 4WARD and SAIL EU FP7 projects. NetInf is based on a location-independent flat naming scheme, in which the NDO name includes a hash digest of the content, and it uses a hybrid of name resolution services and name-based routing for retrieving data objects. NetInf nodes forward the NDO requests to locate the objects and transfer the objects (or pointers to them) back to the requester.

NetInf’s routing scheme consists of Name-Based Routing (NBR) and Name Resolution Service (NRS). The first is hop-by-hop forwarding of the requests until the NDO is found. However, if there is no routing information for the location of the NDO, then the NetInf router asks the NRS for routing hints, in order to continue the routing process of the request. Routing hints are then put into the NDO request, so that the router finds the next hop to forward the request to. These hints do not indicate the final destination that will serve the request, but a way towards it.

1.1 Problem statement

The NetInf routing scheme is designed to achieve global routing scalability but has issues related to its robustness and deployability.

The global routing scheme, as it is described in the NetInf routing IETF draft [6], assigns priorities to the routing hints. This priority-based system is used to route the requests, as it defines an order, where higher priority hints indicate a location closer to the source of the content. The name resolution service replies to a router’s request with multiple routing hints, to aggregate routes and provide multiple sources. Despite the total ordering, the routing system is susceptible to routing loops, if the name resolution service and the routing tables are not consistently configured. Thus, the routing scheme needs to be redesigned, without losing its scalability, to provide loop-free routing based on the idea of partial ordering of the hints [7]. Because of the introduced overhead, not only the functionality but also the forwarding performance of the newly designed scheme must be evaluated.

Furthermore, NetInf can currently be evaluated only at small scale experiments, since the hint forwarding tables have to be statically configured.
Thus, an interface to an open source routing daemon is investigated in order
to take advantage of the current Internet routing protocols. This enables to
dynamically populate the forwarding tables. It also allows easy deployment
of the architecture in larger scale, without the need to develop new routing
protocols from scratch.

The problem that is investigated in this thesis is summarized by the
following question:

*How should the routing scheme with hints be designed in order to be
scalable, high-performance, loop-free and incrementally deployable?*

*Scalable* means it should maintain the global routing scalability by keep-
ing the routing tables to a manageable size (metrics such as resource con-
sumption and convergence time are not considered). *High-performance*
means it should perform efficiently with respect to response time, so that it
would be able to forward sufficiently fast in large networks with high traf-
fi c. In the experimental performance evaluation of this work, the measured
entity is the forwarding performance of a single node in the network rather
than the system performance of a network of multiple nodes. The evalua-
tion consists mainly of the comparison of the execution times among the
functional chains of the router and the identification of bottlenecks. *Loop-
free* refers to the elimination of the routing loop problem when forwarding
a request. Finally, *incrementally deployable* indicates that users don’t have
to change their system architecture in order to deploy NetInf, as it is built
over existing solutions, for instance the current Internet routing protocols
like BGP.

### 1.2 Purpose

This thesis describes the design of NetInf’s routing and forwarding scheme,
its extension with partially ordered routing hints and its evaluation. The
purpose of this work is to design, implement and evaluate a scalable, high-
performance, loop-free and incrementally deployable router for a global ICN.

### 1.3 Goals

The goals of this thesis are the following:

- Investigate the current research in ICN architectures, with respect to
  routing and forwarding challenges and solutions.

- Redesign, implement and evaluate NetInf’s routing and forwarding
  scheme to use partial ordering of routing hints rather than just prior-
  ities. The routing hints pursue scalability by providing explicit aggre-
  gation, but the priority-based system is not loop-free. Therefore, the
routing hints are redefined to form partial orders and the solution is examined with regard to functionality and performance.

- Design, implement and evaluate an interface to an open source routing daemon for populating the hint forwarding tables. Quagga is examined as the daemon to provide routing services, in order to develop an incrementally deployable NetInf routing solution that can also be tested in larger scale.

- Evaluate the forwarding performance of the router by conducting experiments. The costs of the forwarding functions have to be measured to provide an overview regarding the router performance in large scale.

1.3.1 Ethics

One of the prominent ethical issues in the field of ICN and particularly the NetInf design explained in this thesis is the privacy of the users, as the requests for content are visible to the ICN network, however tracing back the requests to individuals may be prevented. Furthermore, ubiquitous caching of the content raises legal issues, as content owners need some form of access control over their data [3]. There have been suggestions towards these issues, such as attribute-based encryption [8] and NetInf extensions for controlling caching behaviour [9], however further discussion is out of the scope of this work.

1.3.2 Sustainability

ICN has features that could be used towards energy-efficient communication networks, but this aspect is still in a primitive research stage, as discussed in [2]. More specifically, end-to-end communication is not mandatory, eliminating energy consuming signaling. Second, ubiquitous caching decreases the distance of the end user from the data, thus reducing the traffic in the core network. Third, routers are aware of the requests in the network layer and are able to optimize their power consumption patterns accordingly.

In addition, ICN is a suitable candidate for multiple scenarios and applications that consider sustainability, like vehicular networking, Internet of things, transportation and energy networks, and network infrastructure sharing.

1.4 Methodology

This project is based on recent works in the field of routing in ICN, in particular the NetInf routing IETF draft [6] that sets the specifications for the basic routing mechanism and the presentation by Narayanan and Oran [7] that suggests using partially ordered sets to aggregate and scale NDN’s
Chapter 1. Introduction

Routing scheme. The interface to the routing daemon and the use of routing protocols resembles the work in RouteFlow [10] and OSPFN [11]. Next, the methodologies regarding research, development and evaluation are explained in detail.

1.4.1 Research methods

With regard to the portal of research methods and methodologies [12], this is a quantitative research project that follows the philosophical assumption of positivism, in which reality is objectively given and is independent of the observer. Essentially, it is experimental research, since experiments are conducted in an attempt to find relationships between the controlled variables and the router’s performance measurements. A deductive approach evaluates the design that was developed in order to refine and improve it.

1.4.2 Development methods

Incremental and iterative development [13] is used for this project to provide flexibility and work on the interesting and challenging problems that come up during the process. The work is divided into tasks based on the goals, discussed previously, the tasks are then again divided into smaller subtasks, and these are developed incrementally to build up the resulting goal. The iterative method suggests a closed-loop system where every task is designed, implemented, evaluated and refined, until it meets the set requirements. The requirements are also in the loop and may be modified based on the feedback from the evaluation process. The requirements are discussed in detail in the chapters 3 and 4 for the routing system and the routing daemon interface respectively.

1.4.3 Evaluation methods

Initially, the evaluation does not concern NetInf as a whole ICN architecture, but the individual components developed, which are the routing and forwarding system and the routing daemon interface.

The routing scheme functionality is validated by testing it on relevant scenarios, like multi-homing networks or content providers from multiple locations. The routing daemon interface functionality is also validated by small-scale experiments to verify its correctness. In this case, the process of using an existing solution to provide routing in NetInf is evaluated itself regarding its effectiveness and cost.

Apart from the qualitative evaluation of the systems, the performance of the forwarding process is evaluated quantitatively through experimentation. The main metric used is the execution time of the router’s functional chain. Code profiling provides a preliminary overview of the performance.
Then, experiments are designed to measure the real system. The experiment topology, scenarios, workload, traffic, metrics, factors and all other parameters are carefully chosen with respect to ICN evaluation methodology [14], to make the experiments as controlled and reproducible as possible. The main goal is to quantify the performance of the new routing scheme, to compare the costs of different functions and identify potential bottlenecks [15]. Chapter 5 discusses the experiment design, measurements and results in detail.

1.5 Delimitations

The scope of this thesis is limited by the following facts:

- The experimental evaluation of the prototype concerns not only the design but also its implementation, two aspects which are very difficult to be decoupled, if possible. Thus, the results evaluate the functional blocks in a relative way. The comparison of results between two different designs, namely this project and [16], implemented in different programming languages, respectively Python and C++, is also a challenge.

- The mechanism of mapping an authority to a set of routing hints is taken as granted in this project. However, the registration of routing hints at the name resolution service for an authority is not a trivial issue. Here, the focus is limited to how the routing hints are used in order to find a path towards the named data object in question.

1.6 Thesis outline

The remainder of this thesis is structured as follows:

Chapter 2 is a literature review of the related concepts, namely ICN principles, the dominant ICN architectures, NetInf and its routing mechanism, open source routing daemons and related work.

Chapter 3 reasons about the motivation and design of routing in NetInf with partially ordered sets of routing hints, the implementation as well as a qualitative evaluation.

Chapter 4 explains the motivation, design, implementation and the challenges of the NetInf routing system with interface to Quagga open source routing daemon.

Chapter 5 describes the experiment design and measurements of the router’s forwarding performance, followed by the results in two scenarios: An edge router-gateway to the NetInf network and an in-network NetInf router.
Finally, chapter 6 concludes this research work and discusses the main results, suggesting future work.
Chapter 2

Literature Study

This chapter discusses the main concepts and benefits of Information-Centric Networking with focus on routing and forwarding. The NetInf architecture, its routing mechanisms and related work are discussed to set the foundation for extending it in the next chapters. Finally, overlay routing is discussed as the base for developing the NetInf Quagga routing system.

2.1 Information Centric Networking

Information-Centric Networking may be the next important paradigm in the evolution of networks, starting from the first circuit-switched telephone networks roughly one hundred years ago through the packet-switched networks that were introduced fifty years ago [17]. Nowadays, ICN is driven from the change of focus from communication between hosts to information retrieval, but it is not a very recent idea.

The Translating Relaying Internet Architecture integrating Active Directories (TRIAD) paper [18], published in 2000, is considered as one of the first next generation architectures and a precursor of the recent ICN work. The IETF draft published in 2002 by Baccala [19] introduced data-oriented networking and argued for such an Internet architecture. After a long pause, in 2007 Data-Oriented Network Architecture (DONA) provided the first detailed clean-slate ICN architecture, but it didn’t manage to attract the interest of the research community. However, the Content-Centric Networking (CCN) proposal of 2009, now developed into Named-Data Networking (NDN), captured the attention of the broader community, supported by the US National Science Foundation’s Future Internet Architecture project [20]. As a result, more projects and proposals followed, some supported by the European Union, including 4WARD and SAIL (proposing NetInf), PSIRP and PURSUIT, COMET, and some by the USA, including XIA and MobilityFirst [21]. These projects have led to a large number of research publications related to ICN. In addition, the Internet Research Task Force
(IRTF) that focuses on the long-term evolution of the Internet has created the Information Centric Networking Research Group (ICNRG), which aims at promoting collaboration in research and connecting it to the evolution of the Internet at a larger scope.

Before discussing the ICN design in detail, figure 2.1 demonstrates the basic communication mechanisms of an ICN. Every router is an ICN router that also has a data cache implemented. In this example, node A requests object X from the network. The publisher of content X is node F. Based on the routing scheme, each node forwards the request on a hop-by-hop basis, until a copy of the object is available in a node’s cache, in this case node F. The response contains object X, which is also cached by the nodes along the path. Afterwards, node B requests object X, and node C satisfies the request, since it holds a copy of object X. There are security mechanisms within the data objects that ensure their authenticity and integrity, so that the serving nodes need not be trusted. Routing of the requests is more complicated and is discussed in section 2.1.3.

![Figure 2.1: ICN communication mechanisms](image)

2.1.1 Design commonalities and differences

All recent ICN architectures, despite their differences, share some common principles, which are not easily identified because of different terminology. These are the basic principles that essentially define ICN [4, 20]:

- **Basic communication messages.** The network applications and protocols follow a communication paradigm similar to publish/subscribe in a global scale. The basic messages are PUBLISH, which is used by content providers to advertise their content, and GET (or SUBSCRIBE), which is used by consumers to request content. GET is usually a synchronous operation that takes place once, as opposed to
the asynchronous notification service of subscriptions. This architecture decouples the sender-receiver communication in both space and time.

- **Pervasive caching.** Every network element is potentially a content cache that may satisfy an incoming request if it already has the data cached. If it doesn’t, it will forward the request and cache the data when the response carrying the data is sent back to the requester. In the limit, every network node is a cache for content from any user and any protocol.

- **Content-oriented security model.** Data security is inherent in the content itself. Instead of securing the connection, the original content providers sign their NDOs, so that the receivers are able to verify the authenticity by their signature.

However, there are fields where the ICN proposals differ [20]:

- **Naming.** The content name is bound to the intent of the content publisher in all proposals. However, there are two different schemes for the naming system. First, the objects have hierarchical human-readable names, the same way it works in DNS today. This enables the consumer to find the desirable content by knowing its name and the hierarchical structure improves scalability, but it needs an external system to communicate the producer’s public key to ensure data integrity. Second, the objects have self-certifying names, eliminating the need for Public-Key Infrastructure (PKI), but they are not human-readable, so a system is needed for the users to be able to find the name of the desirable content.

- **Inter-domain Name-Based Routing.** There are many different approaches in both intra-domain and inter-domain routing, but the real challenge is the inter-domain routing in the Default Free Zone (DFZ) because of the great volume of data objects. This is the main problem that is discussed in this thesis. Alternatives include routing over current solutions like BGP, following current routing models in own protocols, or developing new protocols following another paradigm.

- **Narrow waist.** The narrow waist of the current Internet is IP at the network layer. In some proposals, ICN is deployed as an overlay architecture over IP, which remains the narrow waist, enabling incremental deployment of the ICN services. In other approaches, ICN becomes the narrow waist itself, having extensive implications on the current Internet architecture. In all designs, however, the ICN elements communicate in a hop-by-hop basis.
Chapter 2. Literature Study

2.1.2 Benefits and challenges

The basic principles of ICN claim to introduce a number of benefits, as discussed in previous work [3, 4]:

- **Scalable and efficient content distribution.** Lower response latency is the immediate result of pervasive caching, as the content is stored closer to the consumer, so that the requests do not have to reach the origin server. Besides, services of overlay technologies like peer-to-peer (P2P) networks and Content Delivery Networks (CDNs), which provide caching solutions in a content-based model, can become inherent in ICN.

- **Simplified traffic engineering.** Content hotspots are alleviated as a result of pervasive caching, thus aiding network load balancing.

- **Security.** Data authenticity and integrity are inherently provided by the ICN naming and security model. The communication channels need not be secured by external authorities.

- **Mobility and multihoming.** Mobile clients are inherently supported, as they do not rely on an end-to-end connection needing handovers, but simply continue issuing requests to the network, which is responsible to satisfy them. Multihomed consumers or producers can also communicate through multiple access networks.

- **Ad hoc mode.** ICN enables devices to communicate without any infrastructure.

Despite these potential advantages, there are many challenges ahead that are yet to be addressed. The most important include [3]:

- **Scalability.** Because of the vast number of NDOs as compared to the number of hosts, routing requests is much harder, and this work also resides in the field of scalable routing solutions in ICN, introducing the concept of routing hints.

- **Privacy.** The NDO requests are visible to all the nodes that process them, so there is a loss of privacy, even though tracing it back to the consumer may be prevented.

- **Legal issues.** Pervasive caching means the content owners lose access control over their data, which raises issues regarding property rights.

- **Deployment.** The traditional business models including internet service providers and content providers are challenged, so the incentives have to be clear for them in order to deploy ICN. Incremental deployment over existing systems is important to simplify the transition and allow growth.
Apart from these challenges, there are also arguments that question ICN in its foundation, more specifically whether pervasive caching improves performance. A study in 1999 revealed through measurements that "the distribution of page requests follows a Zipf-like distribution, where the relative probability of a request for the \( i \)\(^{th} \) most popular page is proportional to \( 1/i^\alpha \), with \( \alpha \) typically taking on some value less than unity" [22]. A recent study of request logs from CDNs [4] confirms that his heavy-tailed workload still holds for requests popularity distribution. Past studies suggested that multi-layer or cooperative caching provided limited improvement for Zipf workloads. Based on this motivation, the paper concludes through simulations that an edge caching deployment has almost the same benefits to users and the network as a pervasive caching deployment with nearest replica routing. More analysis suggests that the best case ICN performance can be matched by increasing the size of the edge caches or by simple cooperative caching strategies. Thus, an incrementally deployable ICN that is limited to the edge of the network is proposed, as a simple yet effective solution that does not need extensive re-engineering and is built with available tools.

### 2.1.3 Routing and forwarding

Routing and naming in ICN are two closely connected fields, as aggregatable names simplify routing. Hierarchical names enable aggregation by definition, while flat names need to know the publisher of the content in order to aggregate routing. Aggregation is a fundamental concept of a scalable routing mechanism, since the number of NDOs is huge compared to the number of nodes in the Internet.

The two main routing approaches differ in whether they use a Name Resolution Service (NRS) or not. The NRS maps object names (or publisher names) to network locators that specify the location of the NDOs. The NRS approach uses *name resolution*, it routes the NDO request towards an NRS node to retrieve network locators. Then, it uses these locators to route the request towards the publisher (or a copy of the object). Finally, the located NDO is routed back to the requester.

The second approach is called *name-based routing*, it omits the first NRS step, thus routing directly the NDO request based on its name towards a copy of the object. NRS can provide off-path caching, by providing locators for object copies that might not be on the forwarding path. It can also simplify incremental deployment, as it does not mandate changes to the routing and forwarding processes. On the other hand, name-based routing completely eliminates the NRS step, resulting in lower latency and simpler communication scheme.

Every ICN proposal deploys its own mix of approaches and features. Four major ICN proposals are briefly mentioned, more comparison details are provided in [3]. Both Data-Oriented Network Architecture (DONA)
and Content-Centric Networking (CCN), which has now developed into Named-Data Networking (NDN) [24], use name-based routing and longest-prefix matching for name matching. In DONA naming is flat, and an object name has the form of $P:L$, where $P$ stands for the principal who publishes content (in fact, a cryptographic hash of its public key) and $L$ stands for a unique object label. DONA supports the REGISTER operation for publishing content and the FIND operation for requests. Resolution Handlers are responsible for routing the requests in a hierarchical manner, using anycast to find the nearest copy of the object.

In NDN naming is hierarchical, in the form of, for instance, /example.com/videos/videoA.mpg. This scheme supports routing aggregation, as /example.com/videos can be aggregated to /example.com. NDN routers announce name prefixes, which they can satisfy the requests for. Requests are called interests, which can also be aggregated by keeping state of the issued but not yet satisfied interest packets, at the pending interest table.

On the other hand, Publish-Subscribe Internet Routing Paradigm (PSIRP), a project which developed into Publish-Subscribe Internet Technologies (PURSUIT) [25], uses a rendezvous system to match the publications with the subscriptions. Information is organized within scopes, and every information object is identified by a scope identifier and a rendezvous identifier, which is unique for each item within a scope. Both identifiers are flat and location-independent, but scopes can be hierarchically structured to provide routing aggregation. Content is forwarded from the source to the consumer using a source routing approach with Bloom filters.

Network of Information (NetInf) [5] uses a hybrid approach to satisfy different purposes, that is NRS mechanisms for the global core network and name-based routing for edge or access networks, but it is explained in detail in section 2.3.

### 2.2 NetInf architecture

The main components of NetInf [5] are the named information (ni) URI naming scheme, the NDO object model with its metadata, and the NetInf protocol [26] messages, namely PUBLISH, GET, and SEARCH.

The ni naming scheme has the format shown in listing 2.1, where example.com is the authority, the publisher of the content, usually a domain name, followed by the hash algorithm, here sha-256, and the hash value of the data itself. Names may also contain a query string with routing or other information. This format also has binary and human-friendly forms that may be used depending on the purpose. The naming scheme is flat, as there is no hierarchy related to topology or organization in the names. This enables persistent names that don’t have to change because of internal changes in the organization file systems or network topologies. Routing aggregation
is achieved explicitly by using the authority field. Name-data integrity is provided by the scheme itself, as the message digest of the received data can verify the content validity by matching the hash value. Authenticity, however, needs external PKI. Even though static data may well be supported by this scheme, for dynamically changing data, the hash value of the data is replaced by the hash value of the publisher’s public key, thus providing access to multiple data from the same owner, and authenticity.

\texttt{ni://example.com/sha-256;nwOdQqo10...97efheaXkWnXJD6bcw}

Listing 2.1: NDO name format example

NetInf is designed to be independent of the lower layer, therefore it introduces a Convergence Layer (CL) to map its message fields to the corresponding lower layer, which can be HTTP, UDP, IP, Ethernet etc. The CL provides framing and message integrity, but it can also support transport protocol functions such as fragmentation and reassembly, flow control, congestion control etc. Communication at the convergence layer is hop-by-hop. NetInf can be deployed over heterogeneous networks using one or more CLs. The CLs that have been specified so far are HTTP and UDP in [26] and Bluetooth in [27].

The three fundamental NetInf messages are implemented by every specific CL:

\textbf{GET} messages request an NDO from the network. If a node has the requested NDO in its cache, it responds with a GET-RESP message. If a node holds related information but not a copy of the object, as in the case of NRS, it responds with locators or alternative names for that object. Otherwise, it forwards the request towards a node holding a copy of the NDO.

\textbf{PUBLISH} messages allow content publishers to register the name of their data or their authority to an NRS, and optionally store locators, alternative names, or a copy of the object data or metadata. PUBLISH-RESP messages include a status code.

\textbf{SEARCH} messages contain keywords to be searched in a node’s cache. SEARCH-RESP messages include a status code or object metadata.

NetInf supports caching on the forwarding path, with nodes caching NDOs while forwarding responses, thus being able to respond to requests themselves, providing the benefits of ICN in content dissemination and traffic engineering. It also supports off-path caching, as in dedicated data caches that reduce inter-domain traffic and latency. Peer caching allows user devices to act as on- or off-path caches, proving crucial for networks in challenging environments.
2.3 Routing and forwarding in NetInf

With scalable and efficient content distribution as the main requirement, NetInf is designed to use a hybrid of name-based routing and name resolution services to be highly configurable and cover different parts of a network with a matching routing mechanism. The forwarding mechanism needs the routing information to decide which next hop should the request be forwarded to.

Name-based routing in combination with NetInf’s flat naming scheme is not a very attractive solution for a vast number of objects, as aggregation based on the name itself is not possible. It is, though, a viable solution for edge or access networks with limited number of objects. As shown in figure 2.2, when NetInf router A receives a request for NDO X, it looks up its ni name forwarding table to find next hop C and forward the request. Node D finally has a copy of the object and sends back a GET-RESP including object X. When router C requests the same object, it is served by router B that now has the NDO in its cache. Because this thesis work aims at scalable global routing solutions, the remainder of this work focuses on the NRS.

The NRS is provided by one or more NetInf nodes, which may be dedicated for this service, and it is the means to provide explicit aggregation. NRS translates the authority of an NDO to network or host locators of a different namespace. These locators are called *routing hints* and indicate where to find copies of the requested NDO. The routing hints are used by a NetInf node during the forwarding process to determine the next hop towards a source of the NDO. As shown in figure 2.3, when NetInf router A receives a request for NDO X, it does not have any routing information available, so it asks the associated NRS for routing hints for NDO’s authority, *company.com*. The routing hints indicate nodes D and B, so it looks up its hint

![Figure 2.2: Name-based routing in NetInf](image-url)
forwarding table to find next hop B and forward the request. The routing hints are attached to the request, so the next nodes on the path won’t have to do an NRS lookup again. As a result, node B receives the request and after looking up the hints and its forwarding table, it forwards the request to node D. From that point on, node D satisfies the request by sending the response back on the same path.

Figure 2.3: NRS routing in NetInf

The routing hints are locators that do not necessarily need to specify the final destination, but they can point towards it. They don’t even need to identify hosts, but may have the desired level of abstraction. There might be different NRS nodes for different domains, resulting in multiple NRS lookups by different hops in the forwarding path. Different domains might also employ different routing schemes or routing protocols to populate the hint forwarding tables, but in a global level one protocol is expected for the Default Free Zone (DFZ), like Border Gateway Protocol (BGP) for the Internet nowadays. These NRS configurations allow network providers to improve traffic engineering, reduce their traffic and balance network load.

This is how NRS aids the forwarding process, but how is the NRS table built? Content providers advertise the authority names of the published NDOs to the NRS through *PUBLISH* messages, providing also the routing hints. The object names can also be published at the NRS, but that is not a scalable solution for a global network. NRS can then reply with all the routing hints, a set of them, a prioritized list, or a partially ordered set as proposed in this work. There have been two proposals on a global NRS for NetInf, both of which are essentially hierarchical distributed hash tables, but further discussion is out of the scope of this thesis. However, a short discussion can be found in [5].

The routing of the response depends on the CL. If TCP (or a protocol over it like HTTP) is used, then the response can be returned through the
same socket from which the request arrived. Generally, either the routers
have to maintain state for each request or labels have to be attached to the
request, to use them as a reverse path (e.g. the routing hints).

An in-depth presentation of the routing components and processes is
found in chapter 3. It includes not only the design specified in [6] but also
the further developments of this thesis work.

2.3.1 Partial orders

Partial orders are here introduced in a mathematical as well as a practical
way, since they are later used for the representation of the routing hints.

A set \( S \) with a binary relation \( \preceq \) such, that certain elements of the set can
be compared in the sense that one precedes the other, is called a Partially
Ordered Set (poset). In other words, there are pairs of elements where
neither element precedes the other. If every pair of elements is comparable,
then the partial order becomes a total order. For instance, the set of natural
numbers over the relation of \( \leq \) (less than or equal) is a totally ordered set.

More formally, for two elements \( a, b \) of a set \( S \), if \( a \) precedes \( b \) it is
denoted as \( a \preceq b \). If the relation \( \preceq \) on a set \( S \) is reflexive, antisymmetric
and transitive, it is called a partial order. A set \( S \) together with a partial
order \( \preceq \) is called a partially ordered set. Two elements are comparable if
either \( a \preceq b \) or \( b \preceq a \). If every two elements of a poset \( (S, \preceq) \) are comparable,
then \( S \) is a totally ordered set [28].

Partial orders are used to order sets that do not have a natural order.
A real life example is the genealogical tree, where not every pair of persons
can be compared with the ancestor/descendant relation. Posets have various
applications in computer science, from databases to distributed systems [29].
In the related field of publish/subscribe systems, posets are used for message
filtering, e.g. by content-based routers for storing client subscriptions.

In this thesis, posets are used to represent the routing hints, more specif-
ically the set of nodes that define a path towards the publisher. The paths
may be of arbitrary topology, as they depend on how the publisher adver-
tises them, but they are required to be loop-free. Thus, posets are a suitable
structure to employ and they are graphically visualized by Hasse diagrams.
A sample Hasse diagram is shown in figure 2.4, where node A precedes node
B, which precedes nodes C and D etc, in other words the graph is directed
from the bottom upwards but directions are not shown for simplification. A
possible forwarding path from a node to the publisher could be A-B-C-D1.

Siena poset filters are a popular poset implementation used in Scalable
Internet Event Notification Architectures (Siena) project [30] that aims at
a scalable publish/subscribe event-notification service. For each node in
the Siena poset, successor and predecessor sets are maintained. For the
NetInf project, in order to simplify the scheme and reduce the structure
size, each node of our routing hints poset maintains only the immediate
predecessors set, i.e. the "parents" of a node. The priority level of a node is also maintained to provide a parallel total ordering, where priority 1 is the lowest, indicating the nodes that are the furthest away from the source. More implementation details follow in chapter 3.

2.3.2 Related work

Information-Centric Networking has been a hot topic in the networking research community for the past five years, so there is a long list of publications, especially in the challenging field of routing and forwarding. Here, only a few closely-related works are discussed.

Narayanan and Oran [7] have motivated the need for compression of the routing table size to achieve routing scalability for NDN, proposing explicit aggregation with posets of routing labels. This is also the main motivation for this thesis work, so it is explained in detail in chapter 3.

The NDN project has conducted extensive research and development on the area of routing and forwarding. Even though NDN is based on different design choices, such as hierarchical naming that enables routing aggregation, longest-prefix matching on the name prefixes, and stateful forwarding to name a few, the challenges are valid for any ICN architecture [31]: forwarding strategy and scalable forwarding to reach wire-speed operations with fast table lookup and packet processing. An industrial Cisco team has developed an NDN-based router prototype [32] that achieved forwarding NDN traffic at 20Gbps or higher. Its forwarding scheme aims at efficient hash table-based name lookup with fast collision-resistant hash computation and efficient FIB lookup with 2-stage longest-prefix matching algorithm.

Other relevant NDN challenges include routing protocol design and routing paradigms. The first attempts on routing were to adapt IP-based protocols to work with name prefixes, as in the OSPFN extension to Open Shortest Path First (OSPF) [11]. However, this approach introduced IP-related
problems, such as GRE tunneling and IP address management, which are also discussed in the routing daemon interface in chapter 4. Next is the Named-data Link State Routing (NLSR) protocol, a new name-based design which uses names to identify networks, routers, data etc. and is currently in use. It can work over any underlying communication channel, using it to exchange routing messages. In addition, since the NDN forwarding plane performs fault detection and recovery, it reduces the workload of the routing plane, which now has more resources to examine more scalable routing approaches, such as hyperbolic routing.

NDN report [33] discusses scalable routing in particular. The authors argue that the routing scalability problem is essentially the same in both IP and NDN. IP address space is already too large for the routing tables, so the addresses are aggregated into prefixes to compress the routing table size. However, the need for provider-independent (PI) address prefixes led to the use of Map-n-Encap. This is a system that maps provider-independent addresses to provider-aggregatable (PA) addresses, and then tunneling the packets to the destinations through the PA addresses, enabling aggregation by keeping only a few PA addresses in the DFZ.

Since NDN object space is unlimited and the names are PI, the problem is even greater, but the same idea can be used. There is a mapping system from application names (in NetInf terminology, authority names) to ISP name prefixes, providing aggregation at the edge networks. Then, there is a forwarding mechanism based on encapsulation, i.e. the ISP name prefix. ISPs are networks that provide transit service for their customers. Thus, the DFZ routing tables have a manageable size by containing only ISP name prefixes. The scalability problem is moved from the routing system to the mapping system, which can be handled by DNS.

NetInf routing is essentially built on a similar idea. The mapping system maps authority names to posets of routing hints, using DNS. The PI addresses are the authority names and the PA addresses, instead of ISP names, are the routing hints, more specifically the lowest priority hints that provide the highest level of aggregation and are the only ones advertised in the DFZ. The forwarding mechanism is also based on the PA routing hints, decreasing the forwarding table size.

Finally, Baskaravel implemented in his master thesis [16] a global routing solution for NetInf, based on the same principles and reference documents as this one. Yet not all processes are identical. The NRS resolves the authority of an NDO into a set of routing hints, each with a priority value. Aggregation is provided in two levels: First, by the authority name and second, by aggregating high priority hints on low priority hints. A NetInf testbed was built over the Internet and the results show that hop-by-hop transport has the highest impact on the forwarding process. Encoding and decoding binary objects into/from routing hints is another costly step.

However, this work extends the routing scheme proposal by routing based
on posets of routing hints rather than prioritized lists of routing hints. This
modification has effects in the whole forwarding process, from the forward-
ing table lookup to extra steps added for updating the routing hints of
a request. Furthermore, it builds on the proposed future work by imple-
menting the NetInf BGP routing system with Quagga and also conducting
performance evaluation experiments on a different topology. The results
from these experiments, which measure execution times, depend heavily on
the implementation, thus an absolute comparison between the two designs is
difficult, as this project was developed in Python while the former in C++. 
Nevertheless, the results can be relatively compared.

2.4 Overlay routing

The routing schemes of NetInf need information in their routing tables to
make forwarding decisions based on the routing hints. Populating the rout-
ing tables needs employing a routing protocol at the NetInf layer. NetInf is
deployed over IP (in fact, over HTTP), thus constructing an overlay network
that forms a virtual topology via tunneling. The existing routing protocols
are investigated in order to provide overlay routing, as a quick and incre-
mentally deployable solution.

Open source implementations of common routing protocols are provided
by open source routing software. Some popular open source routing daemons
include Bird Internet Routing Daemon (BIRD), Quagga, and eXtensible
Open Router Platform (XORP). They support the most popular TCP/IP
routing protocols, like Routing Information Protocol (RIP) and Open Short-
est Path First (OSPF) for intra-domain routing and Border Gateway Pro-
tocol (BGP) for inter-domain routing. In order to use a routing daemon for
populating the NetInf hint forwarding table, an interface has to be devel-
oped between the daemon and NetInf. The selection of Quagga is argued in
section 3.2.1.

2.4.1 Related work

Quagga is the selected routing daemon to provide NetInf with routing ser-
dices. Therefore, some related work using Quagga for overlay routing in ICN
or Software-Defined Networking (SDN) is presented. The NetInf Quagga
routing system is based on these premises.

The NDN project has developed the Named-data Link State Routing
(NLSR) protocol [34], which runs on top of NDN. It uses names instead of
IP addresses to identify routers and interfaces, so it can be deployed over any
communication channel. However, the first attempts were to adapt the IP-
based OSPF protocol to NDN. Thus, OSPF for Named-data (OSFNN) [11]
was developed as an extension to Quagga’s OSPF implementation. OSPF’s
Opaque Link State Advertisement (OLSA) future extensibility field was used
by OSPFN to announce name prefixes. Using the OLSAs and Router IDs, OSPFN finds routes to name prefixes. OSPF still runs normally on the overlay topology and computes the shortest path tree. OSPFN does not calculate a shortest path tree itself, but it asks OSPF for the next hop to the origin router of a name prefix.

In NetInf the routing hints are used for computing the routing tables, and they are deliberately designed to follow the IPv4 addressing format, so that they can directly use the current routing protocols. Thus, the problem of mapping name prefixes to IP addresses is overcome by NetInf NRS, which maps authority names to routing hints. Nevertheless, NetInf follows a path similar to NDN for the overlay network configuration. In NDN, every router is identified by an ID address and has one address for each of its interfaces. The routing table, calculated by OSPF, keeps a next hop entry (router interface address) for each router ID, resembling NetInf hint forwarding table.

The major issue that emerged during the deployment of OSPFN on the NDN testbed was the overlay network configuration, in particular “setting up and configuring GRE tunnels in different OSes”. Another problem was the management of private IP addresses. The upcoming pure NDN-based routing protocol, NLSR, was expected to solve these problems.

The RouteFlow project started as QuagFlow [35], combining the Quagga routing system with the low-level vendor-independent OpenFlow, aiming at interoperability between software-defined and legacy networks. In QuagFlow, the Quagga forwarding table is monitored for updates, which are then translated to flow entries and pushed to the OpenFlow switches. RouteFlow [10] is the evolution of QuagFlow and it works transparently to the underlying routing engine (Quagga, BIRD, XORP). It is an ongoing open source project that provides virtual IP routing services over OpenFlow enabled hardware. The routing engine generates the forwarding table and pushes it to the Linux kernel table, which is pulled by RouteFlow to install it in OpenFlow. This is relevant to the NetInf project primarily due to the interface between the Quagga-generated routing tables and the RouteFlow server.
Chapter 3
Routing with hints in NetInf

This chapter discusses the NetInf routing and forwarding scheme extension with partially ordered sets of routing hints. The requirements, system components and design choices are followed by implementation details. Quagga is chosen as the daemon for the routing system, but the implementation is discussed in short (more in chapter 4). The forwarding process is explained from start to finish, focusing on the distinct forwarding functions. Finally, this chapter’s work is qualitatively evaluated with regard to the set requirements.

3.1 Design

The routing mechanism is based on the Global Information Network (GIN) architecture [36] and Narayanan and Oran’s ideas [7] and it is defined in the unpublished IETF draft [6]. The extension of the mechanism and the design choices made during this work are particularly emphasized.

The NetInf routing scheme consists of the following components:

**Routing hints** are locators which provide aggregation for routing information.

**Name Resolution Service (NRS),** also referred to as routing hint lookup service, is based on DNS and it maps an ni name (actually the authority field) to a Partially Ordered Set (poset) of routing hints.

**NetInf routing system** is used to populate the forwarding tables, it may be static or use dynamic routing protocols, like BGP, through Quagga.

**Forwarding tables** consist of the ni name forwarding table, hint forwarding table and next hop table.

Before explaining these components and the respective mechanisms in detail, the challenges that essentially set the requirements for the NetInf routing scheme are discussed.
Chapter 3. Routing with hints in NetInf

3.1.1 Requirements

This work investigates a scalable solution for global routing in the Default Free Zone (DFZ), the global network where the routing tables have no default route, but entries for any destination. In 2014, the biggest routing table carries approximately $5.2 \times 10^5$ routes in BGP [37]. Edge networks have more freedom in choosing a suitable routing scheme.

Scalability

In this work, scalability is studied with regard to the size of the routing tables. Narayanan and Oran [7] analyze the routing scalability challenges in the design of a routing system for NDN, debating on the use of current IP routing protocols in ICN. An architecture like NetInf that uses flat namespace would need $O(10^{12})$ entries in the routing table, based on the number of unique URLs indexed by Google in 2008. This is six orders of magnitude higher than the maximum capability of current BGP routers ($3 - 4 \times 10^6$ entries on high-end route reflectors). Thus, a compression of the routing tables is desirable, in particular through an unambiguous and location-independent naming scheme, since location is not static any more due to mobility and path changes.

The Domain Name System (DNS) is an already established location-independent system that in 2011 consisted of $2 \times 10^8$ top-level domain names. Assuming there are providers who want to have second- and third-level prefixes for load balancing, multihoming or other purposes, and assuming that the sub-domain names follow scale-free growth, the number of routes reaches $6 \times 10^8$ routes, which is 200 times greater than BGP today. As a result, the routing table needs more compression to reach an operational level.

The authors stress the need to compress the routing tables down to the order of $10^6$, especially on a flat namespace. Topological aggregation of location-independent labels is hard, but desirable by routing algorithms. The scalability challenge stems from the combination of the huge routing tables and the great number of routers in global scale.

The proposed solution exploits the topology of the location-independent labels to compress the routing table and enable aggregation. These labels do not have to identify hosts, but can be data centers, networks or even autonomous systems, to achieve higher level of aggregation. The authority names (domain names) are translated by the NRS into these location-independent labels, i.e. the routing hint locators. The longest-prefix match is replaced by exact match of the highest-priority hint in a poset. Aggregation based on the priority (precedence) takes place, as lower priority hints are advertised in backbone networks, while the higher priority hints are limited to the edge networks. Aggregation on the levels of the authority names and their organization into posets can also compress the NRS table in a sim-
ilar manner. The approximately $1.3 \cdot 10^8$ second-level domain names (e.g. example.com) are considered a good estimation for the size of the authority namespace.

**High performance**

Besides scalability, the performance of the routing and forwarding processes must not be ignored, as the scheme is destined for high-speed networks in the DFZ. Aggregation allows treating sets of NDOs the same way in processes such as NRS lookup or request forwarding, improving the overall performance. However, the overhead of building and maintaining posets of routing hints has to be taken into account, so that the final scheme is feasible. Thus, all elements and functions of the routing scheme are designed and implemented with high-performance as a requirement.

**Loop-free**

The structure and ordering of the routing hints must ensure that the routing algorithm avoids routing loops. It is assumed that the routing protocol populating the routing tables is of course loop-free. Deciding the next hop solely by the priority of a routing hint may result in a routing loop, as the exact topology of the hints is not evident only by the priorities.

An example of how requests are routed in the priority-based system is discussed in the topology of figure 3.1. Each routing hint consists of a NetInf locator and a priority. The forwarding process selects the highest priority hint it has a next hop for. Node A receives a request for example.com and it resolves the authority name to the routing hints shown in the NRS table. These hints are now attached to the request. Each node looks for exact matches of the routing hints in its forwarding table, starting from the highest priority to the lowest, and when it finds a next hop, it forwards the request. Let’s assume that node A first finds B (priority-2 hint) as the next hop and forwards the request. Now B searches again the set of routing hints from highest to lowest priority, and forwards to C (priority-3 hint). Finally, the request is satisfied at node D.

Routing loops in the priority-based system may emerge as a result of the NRS configuration and its interconnection with the routing protocol. Two possible causes of routing loops include:

- Inconsistency between routing hints and routing (forwarding) tables. The locators that are advertised by each publisher to the routing protocol should be consistent with the routing hints provided to the NRS. For example, each hop belonging to a path from the DFZ to the content source should have suitable entries in its routing table to be able to follow that path. In figure 3.2, routing hint C is removed from the
Inconsistent priority levels among different domains. NRS may provide different sets of routing hints in different domains, for the same authority. When a request traverses two domains, it may look up the domain-specific NRS for routing hints, and then merge the reply with the hints that it already carries, in order to provide alternative content sources. If they don’t follow the same pattern, the result is a merged set of routing hints with incorrect order. In figure 3.3, a request arrives at node A carrying hints from a domain using priorities 1, 5, 10 for the three highest aggregation levels. If node A has an entry for the hint of priority 5 (even though it corresponds to 2), it forwards back to that next hop, creating a loop. This problem can be solved if all NetInf users follow a standard, clearly defined scheme.

If the priority-based hints are correctly configured, they avoid routing loops. However, there is no systematic way to ensure that the set of routing hints at the NRS is loop-free.

An alternative solution to avoid loops while using the priority-based system is keeping a list of the traversed nodes in the request. When the request traverses a node, its NetInf locator is appended to this traversed path, much like the AS_PATH attribute of BGP. The idea is that the request should never traverse a node twice. Thus, when checking the forwarding table, the router never forwards to a next hop that has already been traversed, but attempts a lower priority hint, until there is no next hop to select and the request fails. This solution avoids loops but at the cost of possibly extra forwards, before finally failing the request. It also introduces overhead, not only transferring but also decoding, processing, and encoding a second list.
of locators as part of a NetInf request. However, it is less complex and easier to implement than the partial ordering solution.

In the partial ordering solution, which is pursued in this work, hints keep track of their predecessors, thus explicitly forming the topology of the locators, providing loop-free routing but adding complexity and increasing the execution time. Notably, the routing hints cannot solve the routing problem themselves, as it depends on the routing protocol and the NRS routing hint configuration. However, the posets manage to avoid routing loops, provided that they are also correctly configured. The difference is that the poset-based system can be systematically validated, in contrary to the priority-based system. In addition, the explicit path selection provides advanced traffic engineering capabilities.
Incremental deployability

Finally, the NetInf architecture should be incrementally deployable so that new users do not have to change all their systems in order to deploy a NetInf network. This means the routing and forwarding mechanisms should use existing proven technologies for difficult problems, such as the routing protocol. Besides, running NetInf on top of HTTP requires a form of overlay routing, that does not influence the underlay core network at all.

3.1.2 Routing Hints

Routing hints allow routing of the NDO requests by providing information to find the next hop rather than the destination. They are looked up at the NRS (or the NRS cache) and then attached to the request. To each request correspond multiple routing hints that use exact matching to provide explicit aggregation with the location-independent locators. The levels of aggregation can be selected as desired.

The IPv4 address namespace is used for the locators of the routing hints, to provide a compatible format to use the current routing protocols like BGP. However, there is no explicit relationship between a 'real' IP address of a router interface and a routing hint locator, which may in fact represent a host, a router, a data center, a subnet, an enterprise network, an autonomous system or any other aggregation entity.

Besides the locator, each routing hint has a priority. Higher priority hints are more specific and they are advertised only close to the destination, while the DFZ normally consists only of the lowest priority hints that provide maximum aggregation. This scheme also allows retrieving data from multiple sources or selecting the best source (multihoming or CDN scenarios). However, the total ordering of hints by priority is not robust enough for selecting the next hop from the forwarding tables. The request may end up in a loop among hints that cannot forward towards a copy of the object, for example because of inconsistent priority levels in different domains.

Therefore, the more explicit structure of poset of hints is investigated in this work. Each routing hint also has a parents field, keeping the lower priority locators that are directly connected to it. This explicitly forms the topology of the path that is selected towards the source of the content. Instead of keeping full successors and predecessors set for every hint, keeping only the parents is sufficient combined with the priority ordering. Furthermore, it keeps the format simple and reduces the overhead of hints attached to the request. When a path is chosen by a NetInf node during forwarding, only the hints that form this path are kept on the request, to ensure loop-free routing with reduced overhead.

The new definition of routing hints includes the following fields:

**Locator** has the format of an IPv4 address but in the NetInf namespace
and is represented in binary or ASCII form.

**Priority** is an integer value in the range of [1, 255]. Higher value means higher priority, representing lower aggregation.

**Parents** is a list of locators, containing all the parents of the hint. For priority-1 hints with no parents, it is the empty list.

**Flags** (optional field) is a list of flags. The defined flags are *NDOSPECIFIC* ("S") meaning the hint is specific for an individual NDO and *LOCAL* ("L") meaning the hint is only valid in the advertised local network. The list contains none or more of these flags. These are not used in the remainder of this work.

It is recommended that routing hints should be encoded in ASCII strings, consisting of comma-separated values of locator, priority, parents, flags. In this work though, a routing hint is a list of fixed length of three or four, as in listing 3.1.

```
["locator", priority, ["parent1", "parent2"], ["S", "L"]]
```

Listing 3.1: Routing hint format

Keeping hints as strings would have been inconvenient, as it would be a string including a list of strings, requiring escape characters and complicated encoding. Therefore, the list solidifies the hint structure and simplifies JSON encoding and processing. Examples of routing hints are presented in table 3.1.

```
"10.0.0.1", 1, []
"10.10.0.1", 2, ["10.0.0.1"]
"10.10.10.1", 3, ["10.10.0.1"]
"10.10.20.1", 3, ["10.10.0.1"], ["L"]
```

Table 3.1: Partially ordered set of routing hints

The hints poset is encoded as a list of hints, i.e. a list of lists. The partial ordering lies within the parents field. The poset of the routing hints of table 3.1, visualized in figure 3.4, would be as in listing 3.2, sorted by decreasing priority.

```
[["10.10.20.1", 3, ["10.10.0.1"], ["L"]],
 ["10.10.10.1", 3, ["10.10.0.1"]],
 ["10.10.0.1", 2, ["10.0.0.1"]],
 ["10.0.0.1", 1, []]]
```

Listing 3.2: Poset of routing hints format

Routing hints are carried by NDO requests in the NetInf *GET* messages. The GET message contains the *ext* field, which is designed to handle future
extensibility and is a JSON-encoded object [26]. JavaScript Object Notation (JSON) is a popular data-interchange format that is easy both for humans and machines to process. Routing hints are encoded in a JSON field of `ext` named `hint`. Because of the new routing definition, the field value is now the poset itself. In other words, it is an array of JSON-encoded arrays (hints), each containing three or four elements. The JSON-encoded poset of routing hints in the `ext` field of the GET message is presented in listing 3.3.

```
ext = { "hint" : [
  ["10.10.20.1", 3, ["10.10.0.1"], ["L"]],
  ["10.10.10.1", 3, ["10.10.0.1"]],
  ["10.10.0.1", 2, ["10.0.0.1"]],
  ["10.0.0.1", 1, []]
]
}
```

Listing 3.3: JSON-encoded poset of routing hints in the ext field

## 3.1.3 Name Resolution Service

The Name Resolution Service (NRS) maps the authority field of the ni names to a poset of routing hints. It is provided by one or more NetInf nodes, which may be dedicated for this service. In addition, every NetInf node has an NRS cache, to keep the known translations in local storage.

As the authority names are in practice first- and second-level domain names, the existing DNS system is used to provide the NRS. The NRS entries are stored in a DNS server, possibly of a DNS hierarchical distributed database, as DNS TXT resource records. DNS TXT records have name-value text strings, and they are used to store arbitrary string attributes as explained in the RFC 1464 [38] and shown in listing 3.4. Thus, newly-defined information can be stored in the scalable DNS system, without any change to the implementation.

```
<owner> <class> <ttl> TXT "<attribute name>=<attribute value>"
```

Listing 3.4: DNS TXT record format
Attribute name is selected to be netinfhint, while the attribute value has the format of the routing hint as defined in listing 3.1, aiming at a common design to simplify processing. However, keeping the JSON format of the routing hint and following the recommendations of the RFC 1464 complicates the formatting problem. In fact, only double quotes are valid for JSON strings. At the same time, the DNS TXT RFC also needs the attribute string inside double quotes. Thus, the locator strings inside the attribute value have to be formatted with escape characters, as shown in listing 3.5. Multiple such TXT records, each containing one hint, compose a poset of routing hints.

```
example.com IN TXT "netinfhint=["10.0.20.1", 3, 
\"10.10.0.1\", \"L\"]"
```

Listing 3.5: Routing hint encoded as DNS TXT record

Content publishers need to provide routing hints for their authority fields, hints that are routable in the respective domain of the NRS, but this issue is not dealt with here. In addition, using the NRS requires all ni names to have a non-empty authority field, otherwise a different routing hint lookup service needs to be defined, for example distributed hash tables. This design emphasizes the incremental deployability, as it is built with existing technologies like DNS.

The NRS cache stores hints retrieved from the NRS or from a local configuration file, sorted by priority. It maps authority names to a poset of routing hints, as shown in table 3.2. The cache policies regarding addition or deletion are not investigated, but all the new hints from DNS are written to the cache. The use of NRS cache prevents unnecessary costly NRS lookups for authorities already known, as it is common for a user to request more than one objects from the same publisher.

<table>
<thead>
<tr>
<th>Authority name</th>
<th>Routing hints</th>
</tr>
</thead>
<tbody>
<tr>
<td>example.com</td>
<td>[[&quot;10.0.0.1&quot;, 1, []]]</td>
</tr>
<tr>
<td>news.example.com</td>
<td>[[&quot;10.10.0.1&quot;, 2, [&quot;10.0.0.1&quot;], [&quot;10.0.0.1&quot;, 1, []]]</td>
</tr>
<tr>
<td>somedomain.com</td>
<td>[[&quot;192.168.0.1&quot;, 2, [&quot;192.0.0.1&quot;], [&quot;192.0.0.1&quot;, 1, []]]</td>
</tr>
</tbody>
</table>

Table 3.2: NRS Cache

An NRS lookup is a DNS lookup, i.e. a DNS request to the NRS/DNS server followed by a DNS response. If the NRS cache misses, then the DNS lookup is performed. DNS lookup may be performed even if there are hints in the cache, e.g. to get newer hints, in which case the hints are merged. It is usually done at either a gateway node or intermediate nodes. There are two main NRS configurations:

- Multiple NRS lookups. Gateway nodes from edge networks to the DFZ or vice versa may look up different NRS servers, with hints that
are valid only in that domain, for aggregation purposes. For example, the NRS of an edge network may only provide enough hints to reach the gateway to the DFZ. Then, a second NRS lookup is induced at the gateway node to provide hints for the DFZ, for example only priority-1 hints. After the DFZ traversal, a new NRS lookup at the destination gateway node may provide more specific hints for routing within the target domain.

- A different strategy is to provide hints up to the content source at the gateway NRS, so that only one lookup is needed, but more hints are carried by the GET message.

NRS lookup normally does not take place at an intermediate node where the GET message already contains routing hints, which the router has a next hop for. However, it may be forced, in order to provide capabilities such as CDN or multihoming. The authority domains may also be broad or more specific to compress the NRS tables (e.g. example.com, news.example.com, server1.news.example.com).

3.1.4 Routing system

The NetInf routing system is designed to take advantage of the current routing protocols, in accordance with the incremental deployability requirement, therefore routing hints use the IPv4 address namespace. It is used to populate the forwarding tables of the NetInf routers, in order to look them up for exact matches of routing hints. The forwarding tables map the routing hint locators to next hop locators. After routing hints for a GET message become available (by NRS or other means), routing the message towards the source is the responsibility of the routing system.

Both static and dynamic routing is possible. Static routing may be used for local domains or demonstration networks, but it is not flexible or scalable. Dynamic routing may use different routing protocols in different domains. Edge domains may use intra-domain protocols, such as RIP, OSPF or IS-IS, but for the DFZ BGP is recommended, the same way as for inter-domain routing in the current Internet. The investigated solution is to provide an interface to an open source routing daemon, like Quagga, and use any supported protocol for routing, to get the next hops. This solution is examined thoroughly in chapter 4.

NetInf BGP populates and maintains the forwarding tables among the NetInf routers in the DFZ. This could be compared to inter-Autonomous System (AS) routing in the current Internet architecture, where in this case routing is provided for the routing hints, meaning identifying the next hop for each routing hint. Only the highest aggregation (lowest priority) hints are advertised to the global NetInf BGP system, while the more specific hints
could be advertised only intra-domain, where each provider or organization can use its own routing protocol of preference.

The routing hint namespace is the IP address namespace in order to make it easier to use the existing implementation of the BGP protocol but the semantics are different. Every NetInf-layer IP address is not an IP-layer IP address, since not every IP router is also a NetInf router. This is in fact overlay routing, as it happens for instance in a P2P network. This enables caching among the NetInf routers, aggregating routing information in different levels, like data center or AS, and using multiple Convergence Layers in the DFZ.

3.1.5 Forwarding tables

There are two main forwarding tables in the process of forwarding an NDO request.

Ni name forwarding table is used for name-based forwarding. It maps specific ni names to next hops and is suitable for edge networks.

Hint forwarding table is used for routing with hints, in combination with NRS. It maps routing hint locators to next hops and is suitable for the DFZ.

The focus of this work is on routing with hints in the DFZ. In addition to these two tables, there is also a table of the NDOs that are available in the cache of the NetInf router, making it possible to satisfy the NDO request.

The hint forwarding table is populated by a routing system, such as NetInf BGP. A NetInf router can support all or a subset of Convergence Layers (CLs), as the CL is a choice between two NetInf neighbours. Routing is CL-agnostic, so the CL is selected after selecting the next hop. Thus, the hint forwarding table is split into two tables, one for selecting the next hop and one for selecting which CL to use and the CL-specific address for that next hop.

This is better explained in tables 3.3 and 3.4. The hint forwarding table maps routing hint locators to next hop locators of the NetInf namespace, with exact matching. After the next hop is selected, the locator is looked up in the next hop table, to find the CL-specific address to forward the request to. In this work, HTTP is exclusively used as the CL, simplifying the next hop table. Here the URL format is used, but a static IP address should be used if possible, to avoid unnecessary DNS requests. While the hint forwarding table is populated by the routing system, the next hop table is populated by the administrator, when configuring the NetInf neighbours, the same way it is done for BGP.

For the sake of completeness, table 3.5 shows an ni name forwarding table of an edge network NetInf router, mapping specific NDO names to
Table 3.3: Hint forwarding table

<table>
<thead>
<tr>
<th>Routing hint</th>
<th>Next hop locator</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>10.10.0.1</td>
<td>10.0.0.1</td>
</tr>
<tr>
<td>192.168.1.1</td>
<td>192.168.1.1</td>
</tr>
</tbody>
</table>

Table 3.4: Next hop table

<table>
<thead>
<tr>
<th>Next hop locator</th>
<th>CL-specific next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td><a href="http://global.example.com/netinfproto/get">http://global.example.com/netinfproto/get</a></td>
</tr>
<tr>
<td>192.168.1.1</td>
<td><a href="http://local.example.com/netinfproto/get">http://local.example.com/netinfproto/get</a></td>
</tr>
</tbody>
</table>

next hop addresses (it could also be used with next hop locators and the extra next hop table).

Table 3.5: Ni name forwarding table

<table>
<thead>
<tr>
<th>ni name</th>
<th>CL-specific next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>ni://example.com/sha-256;ABC</td>
<td><a href="http://local.example.com/netinfproto/get">http://local.example.com/netinfproto/get</a></td>
</tr>
<tr>
<td>ni://news.example.com/sha-256;XYZ</td>
<td><a href="http://local.example.com/netinfproto/get">http://local.example.com/netinfproto/get</a></td>
</tr>
<tr>
<td>default</td>
<td><a href="http://gw.example.com/netinfproto/get">http://gw.example.com/netinfproto/get</a></td>
</tr>
</tbody>
</table>

3.1.6 Forwarding process

From a high-level perspective, the following steps may be taken during the forwarding process of a GET request by a NetInf router in the DFZ. The steps that are actually executed depend on the router configuration, more details are found in the next section 3.2.

1. Receive and process the GET request.
2. Lookup object cache for NDO. If found, reply with GET response.
3. Lookup ni name forwarding table for ni name. If found, forward to next hop.
4. Lookup GET request and/or NRS for routing hints.
5. Lookup hint forwarding table for routing hint locators using exact match. If found, keep CL-specific next hop address(es).
6. If using hints, update them using the poset and encode new hints to the GET request.
7. Forward the GET request to the next hop (here using HTTP CL).
Chapter 3.  Routing with hints in NetInf

3.2 Implementation

The extension of routing with hints is implemented on the open source NetInf prototype in Python 2 (version 2.7), found in [39]. This consists of a NetInf HTTP CL server that supports the NetInf protocol messages *GET*, *PUBLISH* and *SEARCH* and also the ni naming system and a cache on the file system among others. A basic service for forwarding requests is also supported only through a default route.

Implementation choices are made with regard to the high performance requirement. The main components of the implementation with regard to the design section include:

- NRS cache, hint forwarding table, and next hop table. These are implemented as Python dictionaries, a native data type for associative array that is implemented with hash tables. This amortizes the lookup cost independent of the table size, which is a challenge in the case of the DFZ routers in the hint forwarding table and the DFZ authority names in the NRS cache. In the case of next hop table, it simply maintains a coherent design. The next hops themselves consist of the CL and the address of the next hop.

- Routing hints are implemented as lists of length 3 or 4 (locator, priority, and parents are mandatory fields, while flags is optional). The same format is kept in all places: The JSON-encoded hints that are carried by GET requests, the JSON-decoded hints that are processed, the NRS cache and, with the addition of escape characters for the double quote, in the DNS TXT records that serve as NRS and the tables configuration file.

- NRS is implemented as a DNS server. The hints for the authority names are statically populated, encoded as DNS TXT records.

- Routing system is either static, by the configuration file, or dynamic through the Quagga interface.

The configuration of a NetInf router instance consists of:

- Router configuration file, for configuring the parameters of the router, such as domain name, NetInf layer locator, NRS (DNS) server address, default gateway. The desired forwarding functions are also selected here, like name-based routing and routing hints lookup.

- Tables configuration file, for statically configuring the three tables. The next hop table must be configured here. NRS cache may be empty provided there is a DNS server to serve as NRS, and hint forwarding table may be populated by the Quagga interface.

Next, the routing system and the forwarding process are discussed.
3.2.1 Routing system

The routing system can be either static or dynamic. Static routing is an adequate solution for small demonstration networks or possibly even for small edge networks. However, dynamic routing has to be established as soon as the network starts to grow. The DFZ needs a system like the NetInf BGP routing system that provides inter-domain routing, like BGP in the Internet. Therefore, open source routing daemons were examined as the solution to provide ready, robust implementations of the current routing protocols.

Some popular routing daemons that were examined include Quagga, BIRD and XORP.

**Quagga** focuses to be a full routing solution, supports a multitude of routing protocols and operating systems and has a Cisco-like command-line interface (CLI). It is the most popular routing daemon, used by OpenFlow, SDNs, small ISPs and CDNs and it has an active developer community and large user community.

**BIRD (Bird Internet Routing Daemon)** is a project that started as an alternative to Quagga, it is a fast and efficient routing daemon that supports RIP, OSPF and BGP, runs on Linux and BSD systems, and is still developed and supported.

**XORP (eXtensible Open Router Platform)** also supports a multitude of routing protocols and operating systems, has a Juniper-like CLI but has not seen a major update since 2012.

Apart from its popularity, Quagga has the Cisco-like CLI which many users are familiar with configuring. Most importantly, though, it provides a push interface that communicates the forwarding information to an external component through a TCP connection. This component is referred to as the Forwarding Plane Manager (FPM) and it actually is a TCP server that listens for messages containing routes coming from Quagga. This interface allows to easily send the computed routes to a TCP server running on the NetInf router, thus allowing overlay routing. This is the main reason that led to choosing Quagga for implementing the interface between NetInf and Quagga.

The actual design and implementation of the Quagga routing system is extensive and includes multiple components, therefore it is discussed in detail in chapter 4. The main points are summarized:

- Quagga runs BGP in every NetInf router.
- Quagga kernel interface is disabled, so that it does not interfere with the actual Forwarding Information Base (FIB) of the system (overlay routing).
• The overlay NetInf network is constructed using GRE point-to-point tunnels. Every NetInf router (all its interfaces) is represented by only one NetInf locator.

• The FPM server is implemented as part of the NetInf router, listening for routes from Quagga and updating the NetInf hint forwarding table.

3.2.2 Forwarding process

The forwarding process is implemented with high performance as the main requirement. The main request processing flow, the forwarding flow and the consecutive forwarding functions are explained, to present an overview over the forwarding process. More details on each function are discussed during the performance evaluation in chapter 5.

Processing a NetInf GET request

The NetInf router is in fact an HTTP server, listening for requests, since this is a NetInf implementation of the HTTP CL. Despite the fact that the request processing was already implemented, it is explained for completeness. A NetInf GET request is received as an HTTP POST request, spawning a new handler for the specific request. The POST request is parsed and classified as a NetInf GET request, calling the corresponding routine. The GET request fields are extracted and validated. The URI field specifies the ni name of the NDO in request and the ext field contains the routing hints (the msgid field is used to link GET requests with GET responses).

Next, the NDO cache is checked for the NDO that is requested. If it is available, the request is satisfied and the NDO is returned along with the GET response. Otherwise, the forwarding process is executed. If forwarding is successful, the NDO is received, stored in the local cache and returned with the GET response. Otherwise, a GET response with an error code is returned. This process is presented in figure 3.5.

Determining next hops with routing hints

The process of forwarding a GET request is the main part that was designed and implemented in this work. A flow chart of the main functions called during forwarding is shown in figure 3.6, which details the Forward GET request to next hop black box of figure 3.5. The forwarding process depends on the configuration of the NetInf router, for example which routing schemes are used, name-based routing or NRS with routing hints. In this work, name-based routing is not addressed, as it is intended for edge networks rather than the scalable global routing scheme that is pursued. Thus, the forwarding process consists of NRS-based routing with hints.
First, the routing hints (if present) are extracted from the \textit{ext} field of the GET request (JSON decoding) and validated. Then, depending on the configuration, the NRS may be queried even if there are routing hints. An example case is a gateway router to the DFZ, which asks its own NRS for DFZ hints, which were not available at the edge.

The \textbf{NRS lookup} first checks the local NRS cache for hints of the requested authority. On a cache miss, it queries the DNS server for TXT records of the requested authority, decodes the routing hints and stores them in the NRS cache for future use. In case there are extracted hints from the GET request, it merges them to one poset. \textbf{Merging} has to be carefully treated in order to prevent loops. In the simple case, which is evaluated in chapter 5 of this work, the two posets are simply merged so that there are no duplicate entries. This case does not protect against loops due to the merged posets. In the more complex case, merging takes care to respect the
Figure 3.6: NetInf GET request forwarding process

chosen path from the poset in the message, in a similar way to the updating function. Therefore, it only merges higher-priority hints that belong to the same chosen path, to prevent taking a different route which might lead to a loop.

The hint forwarding table lookup has the priority-ordered poset of
routing hints as input and searches for exact locator match through all the hints with priority, higher than the current node’s hint priority. If a next hop locator is found, then the next hop table is looked up for the HTTP next hop address. This results in a list of possible next hops, ordered by priority. The advantage of this design is that it provides redundancy with the cost of looking up many hints. A faster but less robust design would stop searching as soon as there was a hit in the forwarding table.

If the previous process does not provide any next hops, the NRS is looked up, had it not been checked before. Then, the next hop table is looked up for a **default route**, as a last resort solution, normally used for edge networks rather than the DFZ. If no next hops are found, the forwarding process fails.

### Updating hints and forwarding the request

In the case that next hops are found, some processing is needed before the request is forwarded (unless next hop is the default route). The next hops are tested with decreasing priority order, until the NDO is successfully received.

The process of **updating the routing hints** is tightly connected to the poset structure and it has the logic to pursue loop-free routing. Besides, it reduces the overhead of carrying the routing hints, by keeping only the relevant ones.

First, a reverse mapping from the chosen next hop to the corresponding routing hint allows knowing which routing hint led to this next hop. Only the higher-priority routing hints that belong to the same chain with the current routing hint are kept, including the current hint. Same and lower priority hints are dropped to avoid routing loops. The chosen hint is kept, as there may be cases such that the lowest priority hint is the only one known in the DFZ and used by every router. Thus, the choice of a hint determines the path of the request. This does not mean there is a single path, as there may be multiple hints of the same priority level, to provide multihoming or serving from different places. Here, the **parents** field is used to determine which of the hints are successors of the chosen hint, by keeping a predecessors list of the current chain. This allows explicit control over the path taken by the publisher of the content.

After determining the new hints poset, the routing hints are **encoded into JSON**. The rest of the steps compose the transfer part, but they are separated into smaller functions for the purpose of experimental measurements. Since some include propagation or transmission delays, they are quite costly and identifying the execution times at this level is important for the performance evaluation. First, the GET request with the updated hints is **created and forwarded** to the next hop (including TCP connection establishment). Next, the **result is received**, the NDO is retrieved and the connection is closed. Finally, the **results are parsed**, containing the object metadata and the content. This includes a MIME (Multipurpose
Internet Mail Extensions) object parsing and JSON decoding. If all the checks are passed, the forwarding process successfully terminates, returning the NDO.

Finally, the hop-by-hop forwarding has to be mentioned. In this implementation, when a node forwards a GET request, if the next hop has a copy of the NDO, it satisfies the request. If it doesn’t, then the next hop forwards the request itself to another next hop etc, resulting in a chain of open HTTP connections with GET requests waiting for a response. This design introduces challenges such as determining the timeout of these requests and handling simultaneous pending requests.

**Example forwarding scenario**

The process is explained in an example scenario of inter-domain forwarding of a GET request, shown in figure 3.7. The DFZ routers compose a full mesh network, i.e. all routers have a specific route to each other. It is also assumed that all NRS caches and NDO caches are empty. The user from kth.se network sends a GET request for NDO X to her NetInf gateway router 20.20.0.1. All outbound traffic in this network is forwarded by default route to 20.0.0.1, which is the gateway of kth.se to the DFZ. The router finds no routing hints in the request, so it looks up the NRS for authority example.com and retrieves the poset of four routing hints. After it checks its forwarding table, it only finds a route to the lowest-priority next hop 10.0.0.1. The hints of the message are not modified, as the lowest-priority hint is used. Then, the request is forwarded to 10.0.0.1, with the HTTP connection kept open, waiting for the response. Router 10.0.0.1 does not check NRS as there are hints available. After the table lookups, it specifies next hop to be 10.10.0.1, and now processes the poset by deleting the lower priority hint (itself), as a level-2 hint is used.

Following the same procedure, router 10.10.0.1 updates the poset by deleting equal/lower priority hints and hints belonging to other chains before forwarding the request to 10.10.20.1. There, the request is satisfied and a GET response is sent along with the NDO. The NDO is cached in each one of the routers it traverses on its way to the requester. Furthermore, router 20.0.0.1 now has hints for example.com in its cache and can skip the expensive NRS lookup at the DNS server next time. This is just a sample configuration of the routers, and many other combinations of when to perform NRS lookup, which NRS to ask, when to cache NRS hints etc are possible.

More example scenarios that are supported by NetInf are discussed in chapter 5, where they compose the experimental poset topologies.
3.3 Evaluation

The **scalability** requirement with regard to table size is evaluated based on other works, such as the Global Information Network (GIN) [36], which argue that this design is scalable. Aggregation on the authority names reduces the size of the NRS table (mapping system). Aggregation on the low priority routing hints reduces the size of the DFZ hint forwarding table (routing system). An experimental evaluation would require a great amount of resources, which is out of the scope of this thesis.

To argue from a practical perspective, $O(10^{12})$ NDOs are aggregated by the approximately $1.3 \cdot 10^8$ authorities at the higher levels of DNS namespace, as discussed in section 3.1.1. DNS scales well for the same amount of domain names nowadays and the introduction of NetInf is not expected to increase this number of domains. It only depends on the desired level of granularity to add more specific domains to the NRS.

The routing system manages routing hints, where the lowest priority hints aggregate large network entities, such as an Internet service provider or an enterprise network. The number of lowest priority routing hints in the DFZ is expected to be less than the current $5.2 \cdot 10^5$ prefixes advertised in the BGP tables [40]. The routing hints depend on the global network size and topology, which does not change because of NetInf. Thus, the same number of lowest priority hints could suffice for the network, as it does nowadays for IP prefixes. By advertising only priority-1 routing hints in the global network (and higher priority hints within the domain), the number of routing hints could be further reduced to be smaller than today’s BGP
table size. For example the cases of large sites advertising multiple prefixes to the global BGP tables because of multihoming can now be replaced by one priority-1 routing hint.

The **deployability** requirement is pursued by using existing technologies, like DNS for the NRS, the IPv4 namespace for the routing hints and current routing protocols like BGP for the routing system. The highly configurable design allows edge network providers to create their custom networks for testing and drive the growth of NetInf from the edge.

The **high performance** is a core requirement of this work, and it is explained in its own chapter 5, where the forwarding performance is analyzed through extensive experimentation. That involves not only the poset extension but the whole forwarding process, to provide a big picture for all the functions.

### 3.3.1 Routing loops

The **loop-free** routing requirement is pursued by introducing the poset structure. Updating the poset is not sufficient to prevent loops by itself. Therefore, two assumptions are made to verify that the routing solution has no loops:

- The routing protocol populating the routing tables is also loop-free (e.g. BGP).
- A priority policy is enforced through a NRS validating mechanism.

If these assumptions hold true, then routing with posets of hints can be verified as loop-free with regard to these identified sources of loops:

- Inconsistencies between hints at NRS and the routing protocol.
- Inconsistent priority levels.

Two basic NRS configurations are studied, with regard to routing both in the DFZ and edge networks.

First, only priority-1 hints are available at the DFZ. In this case, the same routing hint is used throughout the DFZ to reach the destination domain (the rest are dropped). Provided that the routing protocol has correctly calculated the forwarding tables for this hint, there should be no loops. The case of misconfigured static routing, where the next hops for the same hint end up creating a loop, is not supported. It is actually covered by the assumption that the routing protocol should be loop-free. Finally, to enter the destination domain, hints from NRS may be merged. The merging function keeps only higher-priority hints that belong to the same chain as the chosen hint, so that loops are avoided.
Second, a full set of routing hints is used when routing at the DFZ. In this case, the first router that chooses a path towards the destination updates the poset, dropping the hints which belong to other chains. Furthermore, when merging hints, only hints of the same chain are kept.

Merging and updating posets of hints have the main logic to avoid loops. Next, some policy recommendations and the necessary NRS mechanism are described. These enforce the assumptions to make the proposed scheme loop-free.

An example scheme for assigning priorities is shown in table 3.6. A CDN scenario could be supported by multiple priority-1 hints. A multihoming scenario could be supported by multiple priority-2 hints. Redundancy and load balancing could be supported by multiple priority-3 or priority-4 hints. Higher priority hints are left to the local configuration scheme of the authority domain administrator. They are only advertised within the domain.

<table>
<thead>
<tr>
<th>Priority level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Default-Free Zone gateway</td>
</tr>
<tr>
<td>2</td>
<td>Internet Service Provider</td>
</tr>
<tr>
<td>3</td>
<td>Enterprise network</td>
</tr>
<tr>
<td>4</td>
<td>Data center</td>
</tr>
<tr>
<td>&gt; 5</td>
<td>Locally administered</td>
</tr>
</tbody>
</table>

Table 3.6: Recommended scheme for assignment of priority levels

The defined priority scheme should be enforced to deal with inconsistent priority levels. Therefore, a mechanism for the publishers to enable loop-free configuration can be built. This mechanism should verify that:

1. The poset of routing hints is loop-free itself. This is possible to verify because of the explicit paths formed by the partial ordering.

2. The priority levels conform to the defined policy scheme.

3. A suitable sub-poset of routing hints is advertised to different domain NRS nodes (based on the priority levels). It is recommended to use only priority-1 hints for the DFZ.

These restrictions could be implemented in the NRS itself. The NRS server could first validate the routing hints which are published from the content producers before using them to reply to requests.

A loop-free routing protocol, the update and merge poset functions, and the NRS validating mechanism enable a NetInf routing scheme without loops, with regard to the identified sources of loops. Inconsistent NRS and
routing tables should not lead to loops, as the updating of the poset limits the routing hints, also when merging posets, and the loop-free routing protocol should not have loops for hints that belong to the same chain.

These features force the requests which would otherwise create a loop to be dropped. Thus, loops are prevented, but inconsistent configuration may still cause requests to fail. A consistent NRS and routing configuration needs further research and practical input from experimentation. Agreement between the poset of routing hints advertised to the NRS and the set of NetInf locators advertised to the routing protocol could be checked at the administrator side during configuration.

Finally, besides tackling the routing loop problem, the partial ordering of hints allows explicit path selection, thus enhanced traffic engineering.
Chapter 4

NetInf Quagga Routing System

This chapter examines Quagga routing daemon as a solution for overlay routing to dynamically populate the forwarding table. It discusses the requirements, design and implementation of the interface between Quagga and NetInf, the overlay network configuration and the choice of a routing protocol. This dynamic routing solution enables NetInf deployments to extend beyond small-scale networks.

4.1 Quagga routing daemon

Quagga is a routing software suite for Unix platforms, providing implementations for popular routing protocols like RIP, OSPF, and BGP among others [41]. A Quagga-enabled system acts as a router, communicates routing information with other routers using routing protocols and installs it into the kernel routing table. Besides dynamic routing protocols, static routing can also be configured.

The architecture of Quagga consists of the core Zebra daemon and the peripheral protocol daemons. Zebra collects routing information from all protocol daemons, configures static routes itself, looks up interfaces, and redistributes routes among different routing protocols. It has an interface that enables updating the Linux kernel routing table.

More specifically, Zebra keeps the Routing Information Base (RIB), which consists of the best routes communicated by all routing protocols. It then computes the best route for each prefix across all protocols, building the Forwarding Information Base (FIB). Then, it pushes the FIB to the kernel through an OS-specific interface, which is Netlink for Linux. This enables the kernel to use the Quagga-provided routes for forwarding packets.

One of the reasons for selecting Quagga in this project is the Forwarding Plane Manager (FPM), which provides an interface to push the FIB to
external components through a cross-platform mechanism. Zebra acts as the client, periodically initiating a TCP connection to a well-known port to communicate the FIB to the FPM. The FPM is a TCP server, listening for connections from Zebra on that port. After the connection is established, the client sends messages with new or deleted routes over the socket, essentially a complete copy of the FIB, including routes from the kernel routing table. However, this process does not interfere with the Zebra-kernel interface, thus the FIB is pushed to both the kernel and the FPM server at all times.

4.2 Design

The NetInf routing and forwarding scheme is based on the tables presented in figure 4.1, as discussed in chapter 3. The NRS cache is populated either statically or by a DNS server, serving as the NRS. The next hop table has to be statically configured by the administrator, the same way as BGP neighbours. The hint forwarding table is populated either statically or by the Quagga routing daemon. The FPM server, spawned by the NetInf router, receives new or deleted routes from Quagga and, after filtering them, pushes them to the NetInf hint forwarding table.

![Figure 4.1: NetInf router tables and Quagga](image)

Quagga is employed to use a routing protocol implementation and provide dynamic forwarding tables to the NetInf router. OSPF and BGP are two protocols that are examined for intra- and inter-domain routing. The NetInf locators are deliberately chosen to follow the format of an IPv4 address, so that the current protocol implementations can be used as the NetInf
Chapter 4. **NetInf Quagga Routing System**

routing mechanism. However, there is no network or host part in a NetInf locator. Therefore, a NetInf locator is always represented as a /32 IPv4 address and the hint forwarding table is looked up with exact matching. NetInf locators do not necessarily represent a router, but they could also correspond to a subnet or enterprise network, for the purpose of route aggregation.

### 4.2.1 Requirements

First, an interface between NetInf and Quagga has to be developed, so that the routes calculated by Quagga are communicated to the NetInf forwarding table.

The use of intra-domain routing protocols like OSPF requires direct IP connections between neighbours. However, two NetInf routers may need to traverse multiple IP routers to communicate. In that case, the OSPF routers could be connected through virtual links and treated as if they were connected via an unnumbered point-to-point backbone network [42]. However, if the IP layer addresses of a NetInf router were used, it would be impossible to access the NetInf locator through the FPM interface. The FPM interface communicates entries from the FIB, meaning destinations and next hops, both of which would be IP layer addresses. OSPF-specific attributes which could be used for the NetInf locator, like *Router ID*, are not available at the FIB. Therefore, the construction of the NetInf overlay network is mandatory.

In BGP the neighbouring routers and the advertised networks are explicitly configured. Therefore, it is possible to avoid the overlay network, by advertising networks using the NetInf locators and configuring BGP peers with their IP addresses. Nevertheless, this would result in a mixed namespace hint forwarding table, where a NetInf routing hint locator maps to one of the IP addresses of the NetInf router host. This solution does not provide a clear separation of the NetInf locator and the IP address namespaces.

As a consequence, the challenge is configuring the NetInf overlay network and the routing protocol in order to enable Quagga only as an overlay routing daemon that provides information to the NetInf routers. Thus, Quagga has to be configured and modified to fulfil the following requirements:

1. Quagga should not interfere with the kernel routing table.
2. Quagga should be executed for an overlay network, using the NetInf locators namespace.
3. Quagga should respond to network changes and update the forwarding tables.
4. Quagga source code modifications should be minimal.
4.2.2 Interface between NetInf and Quagga

The interface between a NetInf router and the routing protocol includes the following components, as presented also in figure 4.2:

**FPM server** receives the routes from FPM client and pushes them to the forwarding table it shares with the NetInf router.

**FPM client** sends the calculated routes from Zebra FIB to the FPM server.

**Routing protocol** communicates the best routes to Zebra RIB, in order to calculate the FIB.

![Diagram of interface between NetInf and Quagga components](image)

Figure 4.2: Interface between NetInf and Quagga components

The FPM server is a TCP socket listening for connections from FPM clients and it is spawned by the NetInf router during its initialization. Once a connection is established, it receives FPM messages sent from the client. The FPM client, which is triggered by Zebra, sends new FPM messages whenever the FIB is modified. Every time a new FPM message is received and parsed, the hint forwarding table is updated by adding, modifying, or deleting a route. Before adding a new route to the forwarding table, it is verified that there is an entry for the next hop locator in the next hop table.

FPM is a protocol, defined by Quagga, that creates a point-to-point interface to communicate route changes. The FPM header is presented at table 4.1. Protocol version is currently 1. Type is the payload type, which, for Linux, is a Netlink message. Length is the total length of the message including the header. The FPM payload is a Netlink message of one of the two supported types: *New route* or *Delete route*. *New route* messages include the destination (hint locator) and the next hop as well as the output interface and a metric value among others. *Delete route* messages only include the destination to be deleted.

Netlink is an inter-process communication system between the kernel and the userspace. In the case of FPM, the payload of a Netlink message
is a route message. A route message contains multiple routing attributes. The routing attributes carry information like the destination and the next hop. A detailed description of the Netlink protocol can be found at IETF RFC 3549 [43].

### 4.2.3 Overlay network

Every NetInf router is identified by a NetInf locator, thus NetInf routers form an overlay network over IP. Besides, not every IP router is a NetInf router. The NetInf network is formed by creating tunnels among the NetInf routers. This allows custom network topologies, to enable the use of intra-domain routing protocols such as OSPF. Since one NetInf locator is considered sufficient for identifying one NetInf router due to exact matching, the same locator address is used for every tunnel interface.

An example is shown in figure 4.3. The NetInf locators are the 10.0.0.x addresses. They belong to the NetInf namespace, meaning they carry no network/host part semantics. The 192.0.2.x addresses are the public IP layer addresses (192.0.2.0/24 is an IPv4 address block reserved for documentation [44]). Table 4.2 further illustrates the topology. Generic Routing Encapsulation (GRE) protocol is used for creating the tunnels, which are IP-in-IP point-to-point tunnels with multicast enabled.

<table>
<thead>
<tr>
<th>Version</th>
<th>Type</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 bit</td>
<td>8 bit</td>
<td>16 bit</td>
</tr>
</tbody>
</table>

Table 4.1: FPM Header
<table>
<thead>
<tr>
<th>Local IP</th>
<th>Remote IP</th>
<th>Local tunnel IP</th>
<th>Remote tunnel IP</th>
</tr>
</thead>
<tbody>
<tr>
<td>192.0.2.1</td>
<td>192.0.2.2</td>
<td>10.0.0.1</td>
<td>10.0.0.2</td>
</tr>
<tr>
<td>192.0.2.5</td>
<td>192.0.2.6</td>
<td>10.0.0.1</td>
<td>10.0.0.3</td>
</tr>
</tbody>
</table>

Table 4.2: GRE tunnels at NetInf router 10.0.0.1

4.2.4 Routing protocols

Quagga supports both static routes and dynamic routing protocols, including RIP, OSPF, IS-IS, Babel, and BGP. Since Quagga is utilized to tackle NetInf’s static routing limitation, static routes will not be further discussed here. However, it should be noted that it is possible to set static routes through Quagga, providing some flexibility to NetInf, since the forwarding table can then be modified during runtime.

Babel is a wireless mesh routing protocol that is not interesting for the current NetInf scenario. IS-IS is not included in the official Quagga documentation. Finally, RIP, a distance-vector protocol, is not used because of less features and the routing loop and count-to-infinity problems.

OSPF is a link state protocol that can provide faster convergence times than RIP. It is used in enterprise networks as well as IP backbone networks. Besides, it was also selected for routing in the NDN project, with OSPFN [11]. OSPF’s Router ID attribute could also be used to represent the NetInf locator (however this is not accessible to the FPM). Therefore, OSPF is the selected protocol to test intra-domain routing.

BGP is selected for inter-domain routing, as the de facto standard for Internet routing. A scalable inter-domain routing solution is the main scenario of this thesis, so the focus is on BGP. It also provides more customization, as the advertised networks and the BGP neighbours can be configured. It also supports a keep-alive feature to monitor the status of its paired neighbours. Nevertheless, BGP has its own scalability and increased convergence time problems.

4.3 Implementation

The goal is to keep the NetInf-Quagga interface simple and to avoid unnecessary modifications to the Quagga daemon. The implementation consists of an FPM server in Python, its integration with the main NetInf router, modifications in the Quagga source code, the configuration of the overlay network and the specific protocol configuration.

The FPM server is implemented in Python, as the rest of the NetInf router. It has to be noted that the FPM listener is not part of the Quagga source code, but an example program (in C, by project OpenSourceRouting) is ported to Python. Since the Netlink interface is used for the FPM
messages in Linux, the necessary headers as well as Netlink message and route message parsers are also implemented.

The forwarding table is shared between the NetInf router and the FPM server by utilizing a lock. A valid received FPM message triggers an update to the forwarding table, which may be the addition of a new entry or the modification or deletion of an old entry. Thus, the NetInf router can only read, while the FPM server can read and write to the hint forwarding table. The shared memory design was preferred over a messaging passing system as a faster and simpler solution. The interface between NetInf and Quagga is now functional, but the following steps are still needed to provide dynamic routing services to NetInf.

Quagga modifies the kernel routing table by default, which is not desirable in this project, since Quagga is employed only to provide routing information to the overlay NetInf network. Therefore, Quagga is modified to disable updates from Quagga to kernel, to meet the first requirement. However, Quagga still receives updates from kernel, an issue dealt with after selecting the routing protocol.

The overlay network is created by establishing GRE tunnels between the NetInf routers, so that the network is independent of the underlying topology. As a result, routing protocols that require routers to belong to the same network, such as OSPF, can be deployed. Every NetInf router is identified by a unique NetInf locator, which is advertised as a /32 address, since the whole network uses exact matching. Therefore, Quagga runs using only interfaces and addresses of the overlay NetInf network, to meet the second requirement.

4.3.1 OSPF

With a functional interface between NetInf and Quagga and a constructed overlay network, OSPF is examined as the routing protocol. The NetInf locator could be accessible through the Router ID attribute of OSPF. However, the FPM interface only communicates destinations and routes, and essentially any information that can be delivered by Netlink messages, but no information coming from OSPF. Besides, the FPM client communicates with Zebra, and Zebra does not keep any OSPF-specific information in its database. Thus, the NetInf locator is chosen to be the tunnel address of the interface, and all the tunnel interfaces of the router have this same address. As a result, every router will advertise the same address to all its neighbours, constructing the desirable NetInf overlay network, as described in the second requirement.

The test network is similar to the one in figure 4.3. Even though the routing tables were correctly populated, the OSPF tests introduced a problem associated with link status and tunnels, as there is no keep-alive feature in GRE tunnels by default, hence no way to monitor the status of the links.
Thus, Zebra cannot be aware of a link that is broken, so it keeps providing the route even when its neighbour is no longer reachable, violating the third requirement. This issue requires either sophisticated tunnel software or Zebra hacks in order to manage the connected routes - routes which are read by Zebra directly when an interface is up and running - and give higher priority to the OSPF routes.

### 4.3.2 BGP

Next, BGP is examined, since it provides keep-alive detection between BGP peers and is suitable for a global inter-domain routing scheme. Indeed, BGP allows extensive configuration, since it is possible to choose which networks are advertised by a router to its neighbours. The only advertised network is the NetInf locator address of the router itself (as a /32 network). BGP also supports the Router ID attribute and allows explicit configuration of all BGP neighbours (peers). Autonomous System (AS) numbers reserved for private use can be used. The AS_PATH attribute can be used for loops detection, as it keeps the list of the ASes that a route traverses on the way to its destination. In addition, BGP responds to up/down link events by sending keep-alive messages between BGP peers through a TCP session, fulfilling the third requirement.

However, more modifications to Zebra source code are necessary, to manage the routes which Zebra receives from the kernel routing table. Only routes coming from the selected routing protocol, in this case BGP, are meaningful to NetInf. Therefore, the modifications for the NetInf BGP routing system, with respect to the fourth requirement, are the following:

- Only BGP routes are processed for updates into Zebra’s RIB.
- BGP routes are kept always active, so that they are redistributed.
- BGP routes have the highest priority to be selected, to make it into the FIB and be sent to the FPM client.
- Only BGP routes are sent to the FPM client, ensuring that no other routes will enter the NetInf forwarding tables.

Tests in the sample network of figure 4.3 indicated a correct routing mechanism and responsiveness to dynamic network topology. The convergence time is observed to be higher for BGP than OSPF, as expected.

### 4.4 Evaluation

The NetInf BGP routing system is designed and implemented to meet the set requirements. However, this solution is specific to BGP. A different
protocol solution would require respective changes to Zebra source code. Using multiple routing protocols is also not supported. Such a configuration may be required by a gateway router running BGP for the DFZ and OSPF for the edge network.

An issue that challenges the usability and deployability of this solution is the configuration complexity, making the routing system prone to errors. In order to add a new router to the NetInf network with BGP support, the following steps have to be followed:

**Tunneling.** Create the tunnels between the router and its neighbours.

**Zebra configuration.** Enable the tunnel interfaces.

**BGP configuration.** Configure the advertised network (NetInf locator) and the BGP peers (neighbours).

**Quagga.** Run Zebra daemon and BGP daemon.

Not only the new router but each one of its neighbours has to perform these actions, which can be automated by custom scripts. After setting up Quagga, there are two more configuration files for NetInf itself:

**NetInf router.** Authority name, NetInf locator, ports, forwarding functions.

**NetInf tables.**

- **NRS cache.** If NRS is provided by a DNS server, it can be omitted.
- **Hint forwarding table.** If Quagga is enabled, it can be omitted.
- **Next hop table.** The neighbour routers have to be specified here, too. These are the same NetInf locators as the BGP peers from the *BGP configuration*, so these two processes could be combined. The CL-specific (HTTP) next hop addresses have to be defined, too.

It has to be noted that tunneling is necessary for using the OSPF protocol. BGP can be deployed without the tunnels. In that case, the BGP peers are configured with their interface IP addresses in both BGP configuration and next hop table. However, this is not a recommended solution, as the NetInf abstraction is lost in the mixed namespace of the hint forwarding table.

In accordance with the OSPFN evaluation from the NDN project, the problem of managing private IP addresses to create the overlay network is expected. In addition, configuring tunnels between different operating systems may prove error-prone.
Chapter 5

Performance Evaluation

In this chapter, the forwarding performance of the NetInf router is measured by conducting experiments in two different scenarios. The method and parameters of the experiments are discussed. Next, by collecting timestamps during the experiments, the execution times of the functions are calculated and compared. The behaviour of the functions that display a correlation with the independent variables is discussed. The focus is on the forwarding process functions, which were implemented in this work.

5.1 Forwarding Process Evaluation

The purpose of the experiments is to evaluate the performance of the router when forwarding requests. The scenario is designed to be as simple and as controlled as possible, to be able to identify cause and effect relationships between the results and the parameters. The methodology recommendations of Le Boudec [15] for performance evaluation studies are used as the cornerstone of this work. These include first defining the load, metrics and goals of a performance study and identifying the factors which influence the system. One of the major evaluation methods is selected among measurement, simulation, and analytical, in order to proceed with the scientific method, where a hypothesis is considered valid only after thorough testing. Common performance patterns in computer systems simplify the evaluation process, some of which are bottlenecks and congestion collapse. Next, the evaluation components of this methodology are defined.

In order to evaluate the performance of NetInf router, first the system parameters have to be clearly defined. The workload consists of NetInf GET requests and is measured in requests per second. Interesting metrics include the execution time of the forwarding process as well as its components, router throughput in requests per second, and utilization as the CPU load. Throughput in bits per second is also important, as it depicts realistic requests carrying data, but not so relevant for the forwarding performance.
The execution time here is defined as the elapsed real time from the start to the end of a function, as measured by the system clock. This includes input/output time, such as transmission or propagation delays.

The goal of the router’s evaluation is a relative comparison of the system components execution time, with the aim to identify the bottlenecks on the pursuit of a scalable and efficient NetInf router. The forwarding capacity of the router would also be an interesting result of a throughput measurement, but this evaluation is limited to the execution time.

External factors which may affect the system’s performance include background activity, which may consume resources otherwise available to the router. For example, a forwarding function may be delayed by the scheduler and affect the measurements, which are in the order of µs. Network activity, of the router itself or other network elements, e.g. of a switch, may also affect measurements that include a transfer delay. Randomizing the experiments when necessary pursues to avoid hidden factors, which may influence the final results.

The experiment constants include the hardware and software of the computers and the network infrastructure used. These details are presented in appendix A. The experiment parameters include the NDO size, the forwarding table size, the number of hints per authority, the use of DNS server or local cache as the NRS, and the request inter-arrival time. The set of constants and parameters is explicitly defined for each experiment, as the measurement of some functions requires suitable parameters to fix the processing chain (code path).

The evaluation method of the system is the measurement of its performance during controlled experiments. As the measurements are taken by the system itself, the system under test is disturbed by the measurement system, an effect which has to be analyzed and accounted for. Therefore, the logging process is manipulated not to write to disk for each event but collect a higher amount of data in memory and then dump them to a file.

The scientific method requires extensive testing of a hypothesis before reaching a conclusion. An approach towards that is to formulate a hypothesis and then design experiments trying to invalidate it. Therefore, the expected results are discussed beforehand and then compared to the experimental results. The reproducibility of the results is pursued through the detailed description of the environment and the assumptions. The accuracy is not explicitly calculated, but because of the highly controlled experiments, there is limited variability in the measurements. Graphs like histograms and boxplots visualize and attempt to explain the observed variability.

The identification of system bottlenecks is a main goal of this forwarding process evaluation, in other words identifying the functions that have a high execution time, slowing down the overall process. Bottlenecks normally depend on the different parameters and the workload, so multiple experiments on different configurations are conducted. The NetInf router is not imple-
mented to handle load higher than its capacity, thus in a case of congestion, it is expected to collapse. Such an experiment would indicate the limitations of the router by identifying its overload behaviour.

5.1.1 Data collection

The data collected are timestamp values. The timestamp collection is hard coded in the entry/exit points of the forwarding functions. The resolution of the time system call (in Python, `time.time()`) is 1 µs, thus the resolution of the median values cannot be less than 1 µs either.

When taking a timestamp, besides the time system call, the value is written to a comma-separated string and then appended to a list. By having two consecutive timestamp commands, the overhead of this process is calculated in the range of 1 to 5 µs, with an average between 2.5 and 3 µs and median at 3 µs. In order to be conservative, the timestamp duration is set to 2 µs. When subtracting timestamps to calculate the function execution times, the duration of a timestamp command is also subtracted.

The measurements are taken using the `logging` module of Python. The influence of the measuring system on the system under test has to be minimized. Use of logging by writing to a file for each timestamp adds significant overhead. On the other end, a huge array with timestamps that is written to disk only a few times consumes a lot of memory and may slow down other processes. The chosen solution lies somewhere in the middle. When a request arrives, the handler initializes an empty timestamp list. At specified checkpoints, at the entry and exit of functions, timestamps are appended to this list. By the end of the request processing, the list of timestamps is dumped to a log file on the disk. Thus, this scheme avoids writing to the disk multiple times within one request, while keeping it simple. In addition, this is a thread-safe process, as a timestamp object belongs to each thread and the logging module handles concurrent access to the log file. In the case of parallel requests, logging may slow down the remaining threads. However, in the majority of the conducted experiments, the requests are sequential, therefore the logging process does not interfere with the measurements.

Normal runtime logging is avoided by disabling the informational and debugging log levels, but there is still the overhead of calculating the arguments of those logging calls, even though they are not executed. However, this keeps the measurements more realistic, since in a production deployment logging may be enabled, resulting in slower performance than the one measured here.

Other limitations include the cache lookup on the file system, that might add a random noise factor. Therefore, the cache memory size of the NDOs stored in memory is set large enough (20) to store all the files that are repeatedly requested (maximum 10), to alleviate the problem.

Furthermore, the process of copying a received NDO from the temporary
to the permanent NetInf cache and retrieving it again from the permanent cache, before forwarding the GET response, is disabled. However, it is enabled for the NDO size experiments, in order to measure its effect in the final step of sending a GET response.

In addition, Python’s garbage collection is a factor that may influence the results.

Regarding data analysis, the approach is based on the order statistics (median and other quantiles), because it is robust to outliers and it does not require any distributional assumption. Besides, the execution time histograms show skewed distributions. The function execution times are calculated by subtracting successive timestamps.

Before running the experiments, a good overview of the expensive function calls was given by a Python line profiler, executed locally, resulting in performance improvements in the code. Some test experiments were also run, to verify the correctness of the experiment design.

5.1.2 Experiment setup

The experiments took place locally at SICS Kista office. All the computers were connected to a Local Area Network (LAN). The simple client-router-server topology of figure 5.1 was considered suitable, since the experiments do not evaluate the performance of NetInf as an ICN architecture, but the forwarding performance of a NetInf router.

The specifications of the three computers are found in appendix A. Sproink operates only as the NetInf NDO server, where the NDOs are published and the GET requests are satisfied. Peng is the NetInf router, which receives GET requests from the client and forwards them to the server. Caching of NDOs is disabled, so that the received request is always forwarded. Measurements are taken only on peng. Laptop computer ccts is the client that sends GET requests to its NetInf gateway router, normally peng. In the in-network scenario, it also plays the role of the NetInf gateway router itself. The client is assumed to know the ni names of the NDOs it requests, including the authority field.

It must be noted that ccts and peng connect to the LAN through a 100Mbps switch, which limits the network throughput. The switch specifications can be found in appendix A, while the physical layer experiment topology is presented in appendix B.

Figure 5.1: Experiment topology
Chapter 5. Performance Evaluation

The request generator can be configured to have a variable inter-arrival time, and sequential or parallel requests by multiple clients. There are different experiment configurations designed to measure different functions of the router. The parameters include the forwarding table size, the number of hints and topology of the poset of hints, the DNS or local NRS lookup, the file size of the NDO, and the request rate. Each of these experiments is explained in the next sections.

Each experiment is repeated 100 times. In other words, 100 requests are sent to the NetInf router and the execution times are measured for each one of them.

Each of the NDOs is a file with random content of a specific size. It is also assigned to a specific authority, that translates into a particular poset of routing hints.

5.1.3 Routing hint topologies

The posets of routing hints that are examined are of different topology and number of hints. Each authority has 2-6 associated hints, which is realistic for the DFZ taking into account the level of aggregation that can be achieved with the routing hints. Each poset has its own topology, with either one or two chains:

- Chain 1 (10.0.0.x) is always the one taken.
- Chain 2 (10.0.13.x) is never taken.

This is achieved by configuring the hint forwarding table and next hop table of the router to include only the server. With respect to figure 5.1, locator 0.1 is reserved for the client (ccts), 0.2 is the router (peng), which is the measured node, and 0.3 is the server (sproink).

The topologies are shown in figure 5.2 and they serve the different scenarios A-E, which can be regarded as different authorities. Scenarios A, B, and C are simple linear topologies of 2-4 hints, that explicitly set the forwarding path for the request. Scenario D is a multihoming scenario, where the source of the content is connected through multiple ISPs or through multiple access networks (e.g. fixed and cellular), increasing reliability. Scenario E represents decentralized content dissemination, as in the case of CDNs, that enables serving the same content from different places.

5.1.4 Forwarding functions

With respect to the request processing flow chart 3.5, the NDO is never in the router’s cache. Regarding the forwarding flow chart 3.6, it is configured to follow the straight path from start to end (for the gateway scenario). Thus, the end-to-end forwarding process, from receiving a request to responding with the object, consists of the following consecutive functions:
Chapter 5. Performance Evaluation

Figure 5.2: Experiment posets of routing hints

1. Receive and process HTTP POST request
2. Process NetInf GET request
3. Check cache for NDO
4. Extract routing hints from message
5. Lookup NRS for routing hints
6. Lookup hint forwarding table for next hops
7. Check for default next hop (optional)
8. Update routing hints for selected next hop
9. Forward GET request to next hop
10. Receive NDO and parse result (in the NDO size experiment, this function is split in two parts a & b)
11. Send GET response with NDO (optionally store in permanent cache)

Functions 1-3 and 9-11 are called as pre- and post-forwarding process functions, respectively. They are specific to the HTTP Convergence Layer.
(CL), except for the cache lookup. The actual forwarding process functions are 4-8, and these are CL-independent.

5.1.5 Validity

The functions that were developed in this work are the forwarding process functions (4-8) and they involve manipulation of posets of routing hints. These functions are CL-independent, therefore the focus is on evaluating them, as the results can be generalized for any CL.

However, the HTTP-specific functions are measured in order to provide a full picture of the NetInf router processing delay from the moment a request is received until it is satisfied. This way, HTTP can be evaluated itself as a CL choice. HTTP is considered the most suitable CL for this scenario among the three fully specified CLs for NetInf (HTTP, UDP, Bluetooth). It offers all the necessary transport control functions of TCP and allows easy deployment as an overlay, to drive NetInf’s growth from the edge. Last, this HTTP prototype implementation can be compared to the the C++ implementation of the NEC NetInf Routing Platform (NNRP), which also is over HTTP.

A real world scenario of the evaluated NetInf routing scheme would be in global backbone Internet routers, as discussed in chapter 3. Therefore, the processing delay added by each router should be minimized, to be able to reach high-speed throughput in the order of Gbps at least. It is impossible to compare this prototype implementation to a dedicated hardware router with regard to processing delay. Even software routers using the Linux kernel forwarding module would outperform this application layer implementation. Thus, the results are useful to relatively compare the modules of the forwarding chain and quantify their contribution to the overall processing delay.

5.2 Gateway scenario

In this scenario, the topology is the same as depicted in figure 5.1. The NetInf router acts as the gateway to NetInf network for the users, in other words it is the first NetInf node that receives the GET request. Therefore, there are no routing hints carried by the GET request, forcing it to always look up the NRS for hints, either its local NRS cache or the DNS server. After that, hints and next hop are found, so no check is made for the default next hop.

As there are several parameters that influence one or more forwarding functions, the strategy is to first measure the effect of parameters such as the forwarding table size and the NDO file size. Then, these parameters are fixed, to allow deeper analysis of the most interesting parameter, the poset
of routing hints. Finally, the rate of requests is analyzed, with sequential and parallel requests.

5.2.1 Forwarding table size

The dependent variable in all experiments is the execution time. The independent variable of this experiment is the forwarding table size, i.e. the number of entries in the hint forwarding table. It must be noted that the next hop table only contains two next hop entries, as in reality the directly connected neighbours are less than the routes found in the forwarding table. In order to validate the result, the experiment is repeated for authorities A and C (2 and 4 hints respectively). The rest of the parameters are shown in table 5.1. The results are shown in figures 5.3 and 5.4.

The only function that is expected to be influenced by the table size is the forwarding table lookup (step 6). This work did not focus on efficient table lookup algorithms, however, the use of hash table (dictionary) provides decent performance for tables with up to \(10^5\) entries (this is the maximum that was tested, the average case for item access is \(O(1)\), indicating it may be efficient for larger tables).

<table>
<thead>
<tr>
<th>Constants</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations</td>
</tr>
<tr>
<td>Number of request processes</td>
</tr>
<tr>
<td>Requests inter-arrival time (sec)</td>
</tr>
<tr>
<td>NDO file size</td>
</tr>
<tr>
<td>(1KB = 1024B)</td>
</tr>
<tr>
<td>NRS lookup</td>
</tr>
<tr>
<td>NRS hint merging</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorities</td>
</tr>
<tr>
<td>Forwarding table size</td>
</tr>
</tbody>
</table>

Table 5.1: Parameters for forwarding table size experiment

As seen in the boxplot figure 5.3 for authority A, the forwarding lookup time remains almost constant irrespective of the table size. To further explain, the red line is the median value, the circle is the mean, and the box contains all values between the 25th and the 75th percentile. The length of the box is also called Inter-Quartile Range (IQR). The two 'whiskers' include the adjacent values. The upper value is the largest value that is less or equal to the upper quartile plus 1.5 times the IQR. In other words, all values larger than the 75th percentile plus \(1.5 \cdot IQR\) are considered upper outliers. The same applies for lower outliers. The value of 1.5 to mark the threshold for the outliers is based on John Tukey’s definition of the boxplot.
in 1977 [45] and is used as the default in most implementations. In the following figures there are few upper outliers, which contribute to the mean, but are not shown in the figure, so that it is easy to visually compare the plots. These values usually relate to the first time a request is received or a piece of code is executed. This type of graph was selected as it presents not only the median value but also the distribution of the execution times.
Figure 5.4 for authority C validates this result, despite a slight increase of 1 µs with increasing table size. This also shows that there is a high fixed cost besides the table lookup (initializing data structures, acquiring a lock, constructing next hop lists). The comparison of execution time against number of hints is an issue discussed in section 5.2.3.

5.2.2 NDO file size

The independent variable of this experiment is the size of the NDO in request. This is expected to influence only the functions that follow the forwarding of the request. Thus, these measurements actually regard the HTTP Convergence Layer part. The functions that are of interest are (steps refer to section 5.1.4):

- Forward GET request to next hop (step 9)
- Receive NDO (step 10a)
- Parse result (MIME object parsing, JSON decoding) (step 10b)
- Send GET response with NDO (optionally with caching) (step 11)

In order to measure the cost of caching in the final step, the NDO cache lookup (step 3) is disabled. As a result, the router can now store the NDOs in its permanent cache but it still forwards the GET requests to the server, as it does not check its own cache for the object. The parameters are presented in table 5.2.

<table>
<thead>
<tr>
<th>Constants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations</td>
<td>100</td>
</tr>
<tr>
<td>Number of request processes</td>
<td>1</td>
</tr>
<tr>
<td>Requests inter-arrival time (sec)</td>
<td>1</td>
</tr>
<tr>
<td>Forwarding table size</td>
<td>1000</td>
</tr>
<tr>
<td>Authority</td>
<td>A</td>
</tr>
<tr>
<td>NRS lookup</td>
<td>Local cache</td>
</tr>
<tr>
<td>NRS hint merging</td>
<td>Off</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NDO file size</td>
<td>100B, 1KB, 10KB, 100KB, 1MB, 10MB (1KB = 1024B, 1MB = 1024KB)</td>
</tr>
<tr>
<td>NDO caching</td>
<td>Off/On</td>
</tr>
</tbody>
</table>

Table 5.2: Parameters for NDO size experiment

The sizes are selected with respect to [14], which lists the mean object size of different traffic loads. Web traffic has a mean object size of 1-10
kB, file sharing has 250 kB - 4 MB (selected 1 MB), user-generated content (e.g. Internet videos) has 10 MB. Small 100B files are added, as well as 100 kB, to provide easier interpretation of the results in log scale. The files are generated with random content.

**Processing functions**

The results show that all of the functions before forwarding the request (no file handled yet) tend to be slower for NDOs larger than 100 kB. At 10 MB, they are 10-20% slower than at 100 kB. This observation may be explained by the large file dominating the CPU cache, forcing it to refetch the instructions from memory, thus slowing it down. An example is shown for NetInf GET request processing (step 2) in figure 5.5.

![Figure 5.5: Gateway scenario: Execution time of NetInf GET request function versus NDO size [step 2 of section 5.1.4]](image_url)

**Forward request**

*Forwarding the request* (step 9) to next hop is fairly constant at 6.4-6.8 ms, irrespective of the NDO size (except for the 10 MB case, as discussed before), as shown in figure 5.6. The variance can be attributed to the relatively small sample size and the external delays in the network elements. This function prepares and encodes the HTTP form data (including the NDO URI and new routing hints), then establishes a TCP connection with the next hop (normally 3 times the latency because of the handshake), forwards the HTTP POST request and receives an HTTP response. Thus, this is dominated by the propagation delay, including 4 or 5 times the one-way latency (depending on use of piggybacking at last step of TCP handshake).
For this experiment, this function dominates the execution times for sizes up to 10k, after which, one of the following HTTP functions becomes the most expensive.

![Forward request versus NDO size](image)

Figure 5.5: Gateway scenario: Execution time of forward request function versus NDO size [step 9 of section 5.1.4]

**HTTP CL functions**

The three last functions (steps 10-11) depend heavily on the NDO size, therefore they are presented together in figure 5.7. The execution time distribution is insignificant comparing to the range of the median values, so instead of the box plot, a bar chart of the medians is drawn for easier comparison.

*Receive NDO* (step 10a) retrieves the HTTP response object, including the NDO, and closes the TCP connection. This includes the transmission delay for the object (from the receiver’s perspective).

*Parse result* (step 10b) parses the object, which consists of the content and the metadata. It parses the MIME object and writes the content to a temporary file. Then, it decodes the metadata (JSON), validates the name and encodes the updated metadata.

*Send GET response* (step 11) constructs the GET response (MIME encoded object and JSON encoded metadata) and sends it to the hop that requested it, and finally terminates the request handler. This function also includes the transmission delay (from the sender’s perspective).

Looking first at *receive NDO*, the execution time is fixed for the two smallest sizes, implying the file has already been transferred at the previous function. At 10 kB, the execution time increases but the previous effect is still visible, as the throughput seems to surpass the link capacity, which is
limited by a 100 Mbps switch. At the larger sizes, from 100 kB to 10 MB, the link is saturated, as can be observed by computing the throughput (filesize over execution time), that reaches 95 Mbps (that is because execution time includes more than just the transmission time). Besides, since both of the graph axes are in log scale, the trend line for these sizes is almost linear.

*Parse results* is a processing-only function (includes writing to the disk), that depends however on the NDO size. It seems to have a fairly high fixed cost at 1 ms, but then scales better than the other two, reaching a processing rate of approximately 250 Mbps. However, since it does not depend on link bandwidth, it might become a real bottleneck in the case of faster links (e.g. 1 Gbps).

*GET response* shows a behaviour similar to *receive NDO*, having a very high fixed cost at approximately 1.5 ms, but then scaling marginally faster, reaching almost 100 Mbps.

Interestingly, different functions are dominant for different file sizes. For the two smallest sizes, *GET response* is the most expensive, because of its fixed cost. For the two medium sizes, *parse results* is marginally the most expensive. For sizes in the Megabyte range, *receive NDO* is the slowest function, closely followed by *GET response*. In fact, for the first three sizes, *forward request* is the most expensive function, but it mostly depends on the propagation delay. In reality, it may dominate even larger sizes, as the latency between next hops is expected greater than in the experiment LAN.
GET response with caching

Finally, the caching experiment is conducted. Therefore, the final *GET response* function (step 11) is modified to also store the received NDO in its permanent cache storage, before attaching it to the GET response and sending it. Actually, this is the common case for the current implementation, but it could be more efficient, so the previous results would probably be closer to reality. The results are shown in figure 5.8.

![Graph showing execution time vs NDO size for different HTTP CL functions with caching]{/static/5.8.png}

Figure 5.8: Gateway scenario: Execution time of *HTTP CL* functions with caching of the NDO versus NDO size [steps 10a, 10b, 11 of section 5.1.4]

The implications are mostly evident for small sizes, as expected. The fixed cost of *GET response* is now much higher, at over 4 ms (almost three times increased), dominating the cost from 100 B until 100 kB, marginally. For the Megabyte range, however, it adds only little overhead (2-3 ms) on the function’s execution time. The caching not only stores the NDO to the permanent cache, but also retrieves it again along with the metadata, resulting in unnecessary delays.

In summary, *receive NDO* is the most sensitive function to the NDO size and *parse result* is the least sensitive.

### 5.2.3 Posets of routing hints

This is the main experiment that evaluates the design and implementation work that has been discussed in chapter 3. The independent variable of this experiment is the poset of routing hints. There are five authorities, as shown in section 5.1.3, of different topologies and of 2-6 routing hints. The functions that are examined are those after the NDO cache check, which
initiates the forwarding process, and before the actual forwarding of the request. In particular:

- Extract routing hints from message (step 4)
- Lookup NRS for routing hints (step 5)
- Lookup hint forwarding table for next hops (step 6)
- Check for default next hop (step 7)
- Update routing hints for selected next hop (step 8)

In this scenario, there are no routing hints in the message, so the `extract hints` function will have a constant cost. Moreover, `default next hop` check is never executed, but still has a small fixed cost. The parameters are presented in table 5.3.

First, the router uses its local NRS cache for the NRS lookup function. All the routing hints are statically configured during initialization.

<table>
<thead>
<tr>
<th>Constants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations</td>
<td>100</td>
</tr>
<tr>
<td>Number of request processes</td>
<td>1</td>
</tr>
<tr>
<td>Requests inter-arrival time (sec)</td>
<td>1</td>
</tr>
<tr>
<td>Forwarding table size</td>
<td>1000</td>
</tr>
<tr>
<td>NDO size</td>
<td>1 kB</td>
</tr>
<tr>
<td>NRS hint merging</td>
<td>Off</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Authorities</td>
<td>A, B, C, D, E</td>
</tr>
<tr>
<td>NRS lookup</td>
<td>Local cache/DNS</td>
</tr>
</tbody>
</table>

Table 5.3: Parameters for routing hints experiment

Figure 5.9 is a bar chart of the medians of all the functions included in the process, in log scale. The medians are computed over all five authorities (ranging from two to six hints), thus the median number of hints is 4.

Before the forwarding process, the first three functions have constant cost. `HTTP POST request` is in the order of 300 µs, spawning a new request handler thread, initializing instance variables and processing the HTTP POST request to classify it as a NetInf GET request. Next, `NetInf GET request` extracts and validates the `URI` (ni name), `msgid`, and `ext` fields, with the last one containing the routing hints. It has a cost in the order of 100 µs. Next, the cache is checked for the NDO in request, with a median cost of 100 µs (actual values are in the range of either 50 or 100 µs). This cost will be higher if the NDO is in cache, likely depending on
Figure 5.9: Gateway scenario: Execution time of all forwarding functions, median values over all authorities (2-6 hints)

NDO size, to retrieve the object. In that case, all the following steps will be skipped, except for the last one of sending the GET response.

The last three functions have already been studied in the previous section. After this overview chart, the focus is shifted to the forwarding process functions, listed above. Figure 5.10 zooms in these functions for a comparative study.

Figure 5.10: Gateway scenario: Execution time of forwarding process functions, median values over all authorities (2-6 hints) [steps 4-8 of section 5.1.4]
Extract hints cost is constant (around 17 µs), as there are no routing hints yet, since the router is the NetInf gateway (see figure 5.1). The actual evaluation of this function with routing hints is presented in section 5.3. The function initiates the forwarding process, initializes variables and checks the GET request ext field for hints. This low cost for this function is expected at NetInf gateways, which connect different NetInf domains and are configured to look up the NRS.

NRS lookup checks the local NRS cache for the NDO authority, which is configured to be a hit, thus this is the fastest version of this function (around 13 µs). As there are no extracted hints, no merging is included. This function will also be examined in the in-network scenario (section 5.3). Default check does not look up the next hop table for a default entry, as there are next hops from the hints. In this case, it has a constant cost (around 4 µs).

Forwarding table lookup median is at 35 µs, but also depends on the number of hints, as shown in figure 5.11. To interpret the results, the cost is divided into fixed and variable cost. The fixed cost consists of logging, variables initialization, and building the list of higher-priority next hops. The variable cost is repeated for each hint and includes checking hint flags, checking if own locator is the hint locator (to stop from checking the lower priority hints), then acquiring a lock to look up the forwarding and next hop tables and keeping the next hops. For authority A, with two hints, the tables are looked up for one hint, as the last hint is the router itself, breaking the loop. Thus, for N hints, in this experiment, there are N-1 table lookups. Execution time increases by 4-6 µs per hint after B, but it has to be noted that those extra hint table lookups miss. Only one hint is a hit, which is also included in the first case. Thus, the 20 µs of A can be considered as the fixed cost plus one successful hint lookup cost. The cost of B is the cost of A plus one unsuccessful hint lookup cost. Then, 4-6 µs are added for each unsuccessful hint lookup.

Update hints is the most expensive function (so far), with median at 43 µs, and it also depends on the poset of hints, as shown in figure 5.12. The relationship is yet again not proportional, but 4-6 µs are added for each hint. However, this time it does not only depend on the number of hints, but also the poset topology, more specifically the number of higher priority hints belonging to the same chain as the selected hint. The difference, though, appears too small to be considered, as same-chain hints (cases B,C) add 5-6 µs, while other-chain hints add 4 µs. The largest cost is JSON encoding of the hints list (more than 50% of the time), rather than updating the list. The high fixed cost of more than 20 µs includes mainly logging (three times) and checking the CL (here HTTP).
Figure 5.11: Gateway scenario: Execution time of *forwarding table lookup* function versus poset of hints (authority) [step 6 of section 5.1.4]

Figure 5.12: Gateway scenario: Execution time of *update hints* function versus poset of hints (authority) [step 8 of section 5.1.4]

**DNS lookup**

Instead of using the NRS cache, it is common to use a DNS server for the NRS lookup of an authority to get the poset of hints. The DNS server is located at the client host (ccts), so the router has to send a request through the network, thus the dominating factor is the propagation delay. The only affected function is NRS lookup, as shown in figure 5.13.
Chapter 5. Performance Evaluation

In fact, the NRS cache check results in a miss, spawning a DNS lookup that returns a list of hints, which are then decoded, validated and sorted by priority (in the common case, they are finally stored in the NRS cache to avoid querying the DNS again, but here this is disabled for experimental purposes). Merging does not take place here either.

Instead of the 17 µs observed with the NRS cache lookup, the NRS DNS lookup ranges at 2.5 ms, a time that depends on the location of the DNS server (the RTT between client and server). However, it is noticeable that extra hints add 30-70 µs to the total time, not only because of the extra DNS TXT record that is transferred, but also because of JSON decoding and validation.

Figure 5.13: Gateway scenario: Execution time of DNS (NRS) lookup function versus poset of hints (authority) [step 5 of section 5.1.4]

5.2.4 Request rate

The execution times of the forwarding functions have been studied so far under optimal conditions, namely one request per second by one client. The purpose of this experiment is to increase the router load and observe the results, in an attempt to identify the problematic functions under heavy load. Therefore, one independent variable is the request inter-arrival time, which is decreased from 1 sec to 0 sec (back-to-back requests). Moreover, two, five, and ten parallel requests are examined. Two, five or ten authorities are used, without significant effect. All functions are examined to identify potential bottlenecks, especially among the forwarding process functions. The parameters of the experiment are presented in table 5.4.

The requester that sends N parallel requests is modelled as a fork/join process. It sends N requests back-to-back at once, then waits for all N to
finish, and then continues with the next batch of $N$ requests. Thus, instead of $N$ clients with each one sending back-to-back requests, there is one client that sends batches of $N$ requests back-to-back.

Not all results are presented here, but the observation is that bottlenecks arise in functions that share resources among concurrent requests. By analyzing the results, the shared resources, together with the function they affect, are:

- HTTP server request handler (HTTP request, step 1)
- NDO cache (cache check, step 3)
- Hint forwarding table (forwarding table lookup, step 6)
- Link bandwidth (HTTP-CL functions, steps 9-11)

Thus, the functions can be divided into the resource-sharing functions and those which allocate own resources for each request. The latter group includes NetInf GET request, extract hints, NRS lookup, default check, and update hints. High load has little effect on these functions, but they exhibit a small increase in the median and mean values when moving from 1 to 2, 5, and 10 concurrent requests. Due to processing limitations, there are cases with many slow execution times, increasing the mean value, but the median is normally kept at the same levels.

Concerning the resource-sharing functions, the effect is more evident at the mean value, too. However, the increasing number of parallel requests now increases the median value and shifts the whole distribution, as seen in the figures 5.14 and 5.15. The effect is substantial at five and ten requests.

In summary, the pre-forwarding process functions that struggle under heavy load are HTTP request, due to multithreading HTTP server request handler and the cache check, due to the lock acquisition for access from multiple threads. The post-forwarding process functions, all HTTP-CL specific,

### Table 5.4: Parameters for request rate experiment

<table>
<thead>
<tr>
<th>Constants</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of iterations</td>
<td>100</td>
</tr>
<tr>
<td>Forwarding table size</td>
<td>1000</td>
</tr>
<tr>
<td>NDO size</td>
<td>1kB</td>
</tr>
<tr>
<td>NRS lookup</td>
<td>Local cache</td>
</tr>
<tr>
<td>NRS hint merging</td>
<td>Off</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variables</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Requests inter-arrival time (sec)</td>
<td>1, 0</td>
</tr>
<tr>
<td>Number of request processes</td>
<td>1, 2, 5, 10</td>
</tr>
<tr>
<td>Authorities</td>
<td>2, 5, 10</td>
</tr>
</tbody>
</table>
struggle because of the HTTP/TCP internals, such as the TCP connection establishment for forward request and the TCP congestion control that balances resources among competing connections, during the NDO transmissions in receive, parse NDO and GET response. These last two functions face the greatest impact.

The only forwarding process function that experiences difficulties is for-
warding table lookup, as shown in figure 5.16. The distribution is not shifted a lot, though there are numerous exceptionally high execution times (in the order of hundreds of microseconds up to milliseconds). The reason is the lock used for concurrent access to the table. Even though concurrent read would not be a problem, writing to the table for updating the routes (e.g. from Quagga), has to be protected by a lock. The use of a readers-writer lock, which allows concurrent access to multiple readers but restricts access to a single writer, could alleviate the problem.

It is observed that the difference between one client sending 1 req/sec and one client sending requests back-to-back is almost negligible. In some cases, back-to-back exhibits even faster execution time, as in forward request, a function affected mostly by the network. The tested rate of 10 req/sec is also supported without significant additional delays.

![Figure 5.16: Gateway scenario: Execution time of forwarding table lookup function versus number of request processes and inter-arrival time](image)

5.3 In-network scenario

In this scenario, the topology is shown in figure 5.17. The request is sent by a client to the NetInf router running at the same host (ccts), which acts as the gateway to NetInf network. The gateway looks up NRS for hints, and attaches them to the request. The measured node peng is now an intermediate router inside the network, which receives a request already containing routing hints. Then, it is configured to always look up NRS, so it retrieves more hints and merges them with the existing hints.

This scenario is designed to measure:
5.3.1 Routing hints and merging

This experiment has essentially the same configuration as the routing hints experiment in section 5.2.3, with the parameters found in table 5.3, except for the fact that only NRS cache is used here and NRS merging is enabled. The difference is the position of the measured router in the network.

The pre- and post-forwarding process functions (steps 1-3 and 9-11) exhibit the same behaviour as in figure 5.9. The forwarding process functions median values are presented in bar chart figure 5.18 (steps 4-8). This bar chart can be compared to the equivalent gateway scenario bar chart figure 5.10.

While in the gateway scenario they were negligible, extract hints now dominates this set of functions (71 µs from 17 µs), and NRS lookup becomes
comparable to forwarding table lookup, due to merging (32 µs from 13 µs). There is also a notable 2-3 µs horizontal decrease in the values of forwarding table lookup and update hints, as compared to the gateway scenario experiment. This may be caused by external system factors, such as background load at the time of the experiment. Another explanation may be the different code path followed in this case, that includes fetching JSON code in the extract hints function. Thus, when the update hints reaches the JSON encoding part, it executes it faster. This sort of dependencies make it hard to identify cause and effect relationships between variables in the scale of microseconds.

Figure 5.19: In-network scenario: Execution time of extract hints function versus poset of hints (authority) [step 4 of section 5.1.4]

Extract hints depends more than any other function on the number of hints, as presented in figure 5.19. It has to be noted that not every hint has the same size, as some may have multiple parents and some none. Thus, the decoding time is not constant per hint. The fixed cost consists of initiating the forwarding process, checking the supported features, logging (two times), but this was already included in the previous measurement of 17 µs. The variable cost includes JSON decoding of the ext field, but it seems this is a costly process in itself, contributing to the high fixed cost of the first poset A (two hints). Then, each additional hint adds 3-5 µs to the execution time. All the hint fields (locator, priority, parents, flags) are validated before returning the extracted routing hints to the forwarding process. The slight "jump" from C to D may be because the high-priority hint of poset D has two instead of one parents.

NRS lookup is not constant any more, due to the hints merging and sorting process, as shown in figure 5.20. The NRS cache contains already
Figure 5.20: In-network scenario: Execution time of NRS lookup with merging of hints function versus poset of hints (authority) [step 5 of section 5.1.4]

JSON-parsed posets of hints for each authority. After the NRS cache lookup, the hints from NRS and the hints from the request are merged and sorted by priority. It must be noted that this is a simple merging process of all the hints, as opposed to the loop-free variation discussed in section 3.2.2. Because the NRS cache is the same for all nodes in this experiment, there is only one new hint added from the NRS lookup (the lowest-priority hint). Therefore, the variable cost is little, as the new list does not really need to be sorted. The fixed cost consists of the table lookup, logging (4 times), and the merging and sorting functions. The variable part of these functions is 1-2 µs per hint.

5.4 Conclusion

The measurements on common computers on the microsecond scale can be affected by numerous factors, however the effort was to make the experiments as controlled as possible. In addition, it is challenging to isolate the effects of the design and the implementation of NetInf forwarding within these results.

The choice of an associative array (Python dictionary) for the hint forwarding table proves adequately fast, amortizing the cost for up to 10K entries. This suffices for demonstration and small networks, but not the DFZ, which would require dedicated implementation and analysis. In addition, the concurrent table access problem makes forwarding table lookup the potential bottleneck of the forwarding process functions under high work-
Among the rest of the forwarding process functions, extracting hints from the GET request is the most expensive one, mostly due to JSON decoding, followed by updating hints, which includes JSON encoding. The fairly complex structure of routing hint posets contributes to that cost, thus a custom JSON encoder/decoder could be investigated as an improvement. NRS lookup at the cache (with merging or not) is efficient as it comprises only one table lookup using the same data structure as the hint forwarding table (Python dictionary). Besides, policies about the size of the NRS cache can be set, forcing a DNS lookup for the unknown authorities. The DNS lookup is an expensive function, due to the propagation delay to the DNS server, but the NRS topology and configuration is a vast topic that is not discussed here. It has to be pointed out that the multiple DNS resource records for each authority to construct the poset of routing hints may raise performance issues due to the increasing DNS table size.

The HTTP Convergence Layer functions show interesting behaviour, as the hop-by-hop forwarding of the requests has a high cost of TCP connection establishment. It sets up a separate connection for each forwarded request, leading to a potential bottleneck under heavy load. The remaining functions do not scale well for large NDO sizes, as the triple cost of retrieval, parsing, and transmission of the NDO leads to potential bottlenecks for large objects. Caching objects in the NDO cache is an expensive process especially for small objects. In addition, these functions face challenges with concurrent requests, but this is again a matter of TCP and also of the physical layer, the number and bandwidth of network interfaces and links.

The HTTP server does not always cope with handling a high rate of concurrent requests. The cache check also adds a high variable cost. Therefore, it is examined to use a faster lookup mechanism, like a Bloom filter, before actually looking up the NDO in memory.

In summary, the forwarding performance of routing with posets of hints is substantially limited by the JSON representation. JSON is a text-based data exchange format that was chosen for attaching information to a NetInf message in combination with the HTTP CL. However, it was specified as a CL-independent solution in the IETF draft [26], therefore it was used in this implementation. The more complex the encoded information is, the more severe the effect of JSON parsing is. The newly defined routing hints have a parents field, thus keeping multiple locators in one routing hint structure. This exhibits the trade-off between the complexity in order to manage the routing loop problem and performance. A binary format could be examined for handling the generic case instead, as it could prove more efficient, for instance a Type-Length-Value (TLV) encoding.

The experiments have shown that the current implementation performs well in small-scale test and demonstration deployments. HTTP may not be the optimal CL for a backbone NetInf router, but it provides the necessary
capabilities which otherwise would need to be implemented.

5.4.1 Comparison with NNRP

Similar performance evaluation work was conducted by Baskaravel in his master thesis [16]. The results can only be compared relatively, as the C++ implementation of the NEC NetInf Routing Platform (NNRP) is expected to be much faster than the one in Python.

In his work, where the HTTP CL is also used, the most costly functions are the pre- and post-forwarding process functions. In particular, the step of connecting to next hop (part of this work’s forward request) is the most expensive one, as expected because of the propagation delay and TCP. After forwarding the frame, the HTTP-specific functions are not measured. However, the following most expensive function is the GET request parsing, which classifies the request and spawns a new processing thread. It is observed that the router follows a different process from the point of receiving a request until it starts the forwarding process.

Regarding the forwarding process, it only includes (in this work’s terms) NRS lookup and forwarding table lookup. The hint extraction happens already before parsing the GET request. It is, however, the most expensive out of these three functions, in agreement with the Python results (but much faster, below 20 µs). The NRS lookup includes looking up a hash table (NRS cache) for hints and then merging, resulting in the second most expensive function (again below 20 µs). The forwarding table lookup is the least expensive function, even for multiple lookups (below 7 µs). It returns the highest-priority exact match found, instead of exhaustively searching through the poset for multiple next hops. In the current work, these two functions have almost the same cost. However, more expensive than these is the update hints function, which does not exist as such in the NNRP work. Encoding the new routing hints is part of the frame forward function, which also includes the transmission delay, so these two are not comparable.

For a more qualitative comparison, the transition from a set of routing to a partially ordered set of routing hints is presented in listings 5.1 and 5.2 respectively. As the poset structure is more complex, it decreases the forwarding performance in various functions. For example, a simple JSON encoding of that poset of hints in Python needs almost double the time of the respective set of hints. Similar behaviour is expected in the JSON decoding and validation of hints in extracting hints, and JSON encoding in updating hints. Computationally intensive algorithms for keeping a loop-free poset in merging hints and updating hints, which were not present in the previous design, also add to the total cost.
["10.10.20.1,3",
"10.10.10.1,3",
"10.10.0.1,2",
"10.0.0.1,1"
]

Listing 5.1: Set of routing hints format

[["10.10.20.1", 3, ["10.10.0.1"]],
 ["10.10.10.1", 3, ["10.10.0.1"]],
 ["10.10.0.1", 2, ["10.0.0.1"]],
 ["10.0.0.1", 1, []]]

Listing 5.2: Poset of routing hints format
Chapter 6

Conclusion and future work

A routing scheme for NetInf based on routing hints was developed during the course of this thesis, aiming at scalability. The introduction of partial ordering of the routing hints led towards the elimination of routing loops. An interface to the Quagga open source routing daemon was implemented, to allow the use of existing Internet routing protocols in NetInf. The forwarding performance of the router was evaluated through controlled experiments. In this chapter, the main conclusions are discussed and future work is suggested.

6.1 Conclusion

Scalable and efficient routing in a global scale ICN is a challenging problem, with regard to the great amount of objects that may be requested, and the finite maximum size of routing tables. The proposed solution is based on name resolution, by mapping the authority (domain) name of a requested object to a set of routing hints, which suggest where to forward the request to, in order to locate a copy of the object. Topological aggregation on the authority names (based on DNS) and on the location-independent routing hints allows compression of the routing tables in the default-free zone of the Internet. The scalability requirement is theoretically claimed with regard to table sizes, but the developed routing scheme is evaluated with regard to routing loop avoidance and forwarding performance.

The redesign of routing hints to incorporate partial ordering tackles the routing loop problem. The solely priority-based routing hints may cause a forwarding loop for an object request, mainly due to misconfiguration of the routing hints and the routing protocol. The partial ordering of routing hints explicitly specifies paths towards a content source, and these paths can be systematically verified to be loop-free, thus avoiding loops in the forwarding process. It is also assumed that the routing protocol is loop-free and that the NRS validates the routing hints and enforces a common priority policy.
However, this more complex design also affects the forwarding performance, by increasing the execution time of the forwarding functions.

Therefore, the experimental performance evaluation provides key insights regarding the costs of the forwarding process functions, by measuring their execution times. Among the forwarding process functions, which follow the object cache check, and precede the request forwarding to the next hop, forwarding table lookup is identified as the potential bottleneck under heavy load. The main reason is the hint forwarding table, a shared resource among concurrent requests and the routing protocol that populates it.

Nevertheless, that is not the most expensive function under normal conditions. Extracting the routing hints from the request and then updating them are the two most costly functions, mainly due to JSON decoding and encoding respectively. The routing hint locators are encoded as ASCII strings, and the new routing hint format that includes the parents field, implies that each routing hint carries from one up to three or four locators (in a realistic scenario). The routing hint structure complexity makes the JSON effect more visible. NRS lookup performs efficiently, and DNS serves well as the choice for providing NRS, but scalability issues may arise with DNS serving as the global NRS, due to the multiple routing hint records for each authority name.

HTTP as the convergence layer proves a convenient but not optimal choice, providing all the underlying TCP functionalities at the cost of increased latency. This latency is tightly bound to the hop-by-hop request forwarding of NetInf, that includes establishing a TCP connection for each request. In addition to increased latency, this could also become a bottleneck in case of multiple concurrent requests. The transmission delays of receiving and sending the object cannot be avoided. However, the intermediate step of decoding, parsing and validating the GET response (including MIME and JSON) to allow object caching, and encoding it again to forward the GET response, is the third function depending on the file size. This is another potential bottleneck for large object sizes, even though it is necessary for ubiquitous caching in ICN.

After receiving a request, checking the cache for a copy of the object imposes a high variable delay. Storing the object in cache is another expensive process, especially for smaller objects. The concurrent access problem lies also with the object cache, making it another potential bottleneck under high request rates.

The interface to Quagga enables dynamic routing in NetInf. The use of existing routing protocols such as BGP makes this an incrementally deployable solution, though inflexible due to the necessary source code modifications. The configuration complexity is expected to prevent widespread adoption. Nevertheless, it is considered a sufficient solution for experimenting at larger scale and driving growth from the edge.

In summary, the proposed scheme could be a feasible solution for loop-
free NetInf routing targeting at global scale. The forwarding performance analysis indicated challenges to improve upon. Considering this is a research prototype in Python, a production quality implementation in a different programming language that tackles the issues identified here, could prove much faster.

6.2 Future work

The identified challenges and results of this project suggest a few extensions or improvements in NetInf design and implementation.

Regarding routing loop avoidance, a more thorough analysis of the interdependencies between the NRS and the routing protocol would allow to formulate a complete set of requirements for consistent NRS and routing configuration. The goal is to verify the loop-free scheme and ensure a robust routing solution that minimizes the percentage of failed requests.

Regarding performance, a more efficient encoding of the poset of routing hints in the NetInf messages could improve the execution times of extracting and updating hints. Therefore, binary format representations like Type-Length-Value encoding could be examined.

A different convergence layer that resolves the HTTP/TCP scaling problems could be examined as the NetInf underlay. This has also been proposed in a previous master thesis [16]. There has been work with Bluetooth as the NetInf convergence layer, but with focus on ad hoc scenarios and opportunistic content sharing [27, 46]. An investigation on a suitable convergence layer for the global ICN infrastructure should tackle the hop-by-hop issue by keeping a pool of open connections and allowing connection reuse, while providing the minimum capabilities as defined in the NetInf specification [26].

The potential bottleneck of the forwarding table lookup function could be alleviated by using a readers-writer lock, which allows multiple concurrent readers but only one writer. The routing tables are not expected to be modified very frequently by the routing protocol, but allowing concurrent access to multiple requests could prove vital. The more sophisticated solution of a read-copy-update mechanism, also used in the Linux kernel, could be employed to provide wait-free reads.

Concerning caching, an investigation on a more efficient cache lookup mechanism is proposed. A possible candidate is a Bloom filter. Bloom filters have gained popularity in the networking literature, providing space-efficiency and fast membership queries [47]. They allow false positives, but no false negatives, meaning that if according to the Bloom filter a node does not hold a copy of an object, it can decide with certainty to immediately forward the request. In addition, a more efficient implementation of the NDO caching process is considered necessary.
A NetInf routing extension to incorporate multipath routing is supported by the forwarding table lookup function. Currently, multiple next hops allow redundancy in case the highest-priority next hop fails. Thus, an investigation on multipath routing and load balancing is recommended to take advantage of the multiple paths. Therefore, a metric for measuring the quality of the next hops and a structure for keeping the relevant statistics should be selected. Related research has already been conducted for ICN and in particular for NetInf [48]. In combination with the posets of routing hints defining explicit paths to the source, this mechanism could be extended for traffic engineering purposes.

Another area of ICN that could take advantage of this routing scheme is the ad hoc mode. By using route discovery mechanisms, routing hint posets could be constructed to form a distribution tree from the source (e.g. router) to the consumers (e.g. mobile devices). Peer caching, where user devices also function as caches, could be enabled, so that some users retrieve the content from other users, offloading the router. This investigation could be based on the previous work with Bluetooth as the convergence layer.

Concerning the dynamic routing solution through the interface to Quagga routing daemon, automation could simplify the deployment process. The selection of a routing protocol could also be supported at the software configuration step. After the overlay network is built and the routing protocol converges, the routing information could be used for forming the NRS posets of routing hints.

Furthermore, the next hop table configuration could be avoided through a service discovery protocol. The neighbours should still be configured at Quagga as BGP peers, so that the routing tables converge. Next, a service discovery could be initiated for each next hop, to discover which convergence layers it supports and the CL-specific addresses. Thus, NetInf neighbours could support a different set of CLs and handle matching and selection themselves.
Bibliography


Appendices
Appendix A

System specifications

The system specifications of all three computers used in the experiment are presented at table A.1. The specifications of the switch are presented at table A.2.

Even though *peng* is the measured node, the functions that include a transfer part also depend on the communicating node. Thus, the *forward request* and *receive NDO* functions depend on server’s speed (*sproink*), while the *GET response* function depends on the client’s speed (*ccts*). Nevertheless, the switch is the main limiting factor, as all communication passes through it.

<table>
<thead>
<tr>
<th>Computer</th>
<th>ccts</th>
<th>peng</th>
<th>sproink</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>Intel(R) Core(TM) i5-3210M CPU @2.50GHz</td>
<td>Intel(R) Core(TM) i5-3570 CPU @3.40GHz</td>
<td>Intel(R) Xeon(R) CPU E31225 @3.10GHz</td>
</tr>
<tr>
<td>Architecture</td>
<td>x86_64</td>
<td>x86_64</td>
<td>x86_64</td>
</tr>
<tr>
<td>Memory (GiB)</td>
<td>4</td>
<td>4</td>
<td>8</td>
</tr>
<tr>
<td>Ethernet interface</td>
<td>Realtek RTL8101E/RTL8102E PCI Express Fast Ethernet controller (100Mbps)</td>
<td>Intel Corporation 82579LM Gigabit Network Connection (1Gbps)</td>
<td>Intel Corporation 82579LM Gigabit Network Connection (1Gbps)</td>
</tr>
<tr>
<td>Operating System</td>
<td>Ubuntu 12.04.4 LTS</td>
<td>Ubuntu 12.04.4 LTS</td>
<td>Ubuntu 14.04 LTS</td>
</tr>
</tbody>
</table>

Table A.1: Systems specifications
Appendix A. System specifications

<table>
<thead>
<tr>
<th>Name</th>
<th>Netgear FS108 Fast Ethernet Switch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ports</td>
<td>8</td>
</tr>
<tr>
<td>Data rate</td>
<td>100 Mbps</td>
</tr>
<tr>
<td>Network latency</td>
<td>20 μs max (using 64-Byte packets)</td>
</tr>
<tr>
<td>Queue Buffer</td>
<td>96 KBytes</td>
</tr>
</tbody>
</table>

Table A.2: Switch specifications
Appendix B

Experiment topology

The physical layer experiment topology is shown in figure B.1. The network interfaces of \textit{peng} and \textit{sproink} support 1Gbps, but all communication is limited to 100Mbps, since they are connected through the switch.

Figure B.1: Physical layer experiment topology