NETINF – NETWORK OF INFORMATION

An Information-Centric Networking Architecture for the Future Internet

Dissertation

CHRISTIAN DANNEWITZ

zur Erlangung des akademischen Grades

Doktor der Naturwissenschaften (Dr. rer. nat.)

im Fach Informatik

Fakultät für Elektrotechnik, Informatik und Mathematik

Universität Paderborn

May 2013

Christian Dannewitz: *NetInf – Network of Information,* An Information-Centric Networking Architecture for the Future Internet, © May 2013

GUTACHTER:
Prof. Dr. Holger Karl (Universität Paderborn, Germany)
Prof. Dr. Jörg Ott (Helsinki University of Technology, Finland)

ABSTRACT

The usage of the Internet has changed significantly in recent years. The original Internet architecture has been designed to facilitate the communication between a defined set of network nodes, i.e., the design is *host-centric*. Today's dominating use case has become information retrieval, i.e., an *information-centric* use case. In this use case, the user is only interested in the information itself; it is irrelevant *which* node delivers the information as the information itself is node-independent in this scenario. At the same time, this shift has resulted in a tremendous traffic increase. The Internet protocol suite handles this traffic inefficiently due to its host-centric design which dictates the data source in advance instead of allowing the network to choose the most suitable data source; i.e., the host-centric design does not fit the information-centric use case. This results in several problems. Especially, the original Internet architecture incorporates neither caching nor the ability to intelligently select the most suitable data source(s), which results in inefficient, redundant transmission of identical content and unnecessarily high network traffic and latency. Other problems include high load at the origin server, limited robustness due to single points of failure, vulnerability to distributed denial of service (DDoS) attacks, the flash-crowd effect[1], and hard-to-handle connection interruptions and mobility (e.g., of clients, servers, and entire networks).

These problems are mainly addressed via application-specific peer-to-peer (P2P) overlays, proprietary, provider-specific content distribution networks (CDNs), and specific application layer solutions such as Hypertext Transfer Protocol (HTTP) load balancing today. However, these solutions cannot leverage the full potential of content distribution as each solution independently addresses only a subset of symptoms instead of addressing the main cause of all these problems. Moreover, today's solutions often reside in *application-specific* P2P overlays and *provider-specific* CDN overlays. This leads to problems such as inefficient data forwarding, a limited number of supported applications, and missing support for small content providers, which suffer most from problems such as the flash-crowd effect and DDoS. Other solutions such as proxy caches require additional client-side or server-side configuration. Moreover, all mentioned solutions only focus on content distribution but do not address the other problems of the original Internet architecture such as mobility and intermittent connectivity. Currently, these problems again require sepa-

1 The effect that certain content, e.g., a web page, is suddenly "overwhelmed" by an unusually large number of concurrent users, typically rendering this content inaccessible for all users.

rate solutions such as mobile IP. This leads to an unnecessarily complex Internet architecture and in some cases (such as transparent web caching) also violate the rules of the Internet protocol as data packets are transparently redirected to a destination other than defined in the packet header.

The information-centric networking (ICN) paradigm aims to address these problems by shifting the focus from a host-centric architecture towards an information-centric architecture that puts information retrieval at the center of the architecture, thereby better fitting today's dominating Internet use case.

This thesis describes a new, information-centric Internet architecture called Network of Information (NetInf) that addresses the afore mentioned problems in an architecturally sound way. It aims to offer efficient information dissemination in a very general form including many different related use cases for a wide variety of applications, for all content providers, and in a variety of different scenarios.

In this thesis, I first evaluate the ICN paradigm and existing approaches in general before describing the design of the NetInf architecture in general. The NetInf architecture combines a unique set of characteristics. This includes its secure naming scheme, flexible object retrieval based on name resolution *and* name-based routing, its object model, and the ability to adapt to a wide variety of different network scenarios. This also includes a flexible caching model that supports on-path (i.e., data and resolution path) as well as off-path caching, including in-network caches and caching on user nodes. I focus specifically on two aspects: (1) The design and evaluation of the secure naming scheme, which builds the foundation of the secure, information-centric architecture. (2) The name resolution service (NRS), which is essential for efficient information retrieval but challenging due to the flat namespace and huge number of expected data objects ($\approx 10^{15}$). In addition, I describe and evaluate our prototype of the overall NetInf architecture called *OpenNetInf*.

The results from the theoretical analysis, simulations, and prototyping indicate that a scalable, world-wide Network of Information with 10^{15} data objects is feasible with an NRS latency of (well) below 100 ms. The flexible NetInf architecture proved to have several benefits and is adaptable to various network and application scenarios. Based on these results, it seems reasonable that NetInf can offer a serious complement to the current Internet architecture.

ZUSAMMENFASSUNG

Die Nutzung des Internets hat sich innerhalb der letzten Jahre signifikant verändert. Während die ursprüngliche Internet Architektur *knotenzentrisch* ist (d.h. Aufbau einer Kommunikation zwischen festgelegten Netzknoten), hat sich die Informationsbeschaffung, also ein *informationszentrischer* Anwendungsfall, in den letzten Jahren zum dominierenden Anwendungsszenario entwickelt. In diesem Szenario ist der Nutzer nur an der Information an sich interessiert und nicht daran, welcher Knoten im Netz die Information liefert, da die Information in diesem Szenario knotenunabhängig ist. Gleichzeitig hat diese Veränderung zu stark gestiegenem Datenverkehr im Internet geführt. Dieser Verkehr kann von den Internetprotokollen jedoch nicht effizient gehandhabt werden, da ihr knotenzentrischer Entwurf im Vorhinein schon die zu verwendende Datenquelle vorschreibt anstatt dem Netz die Freiheit zu lassen, selber die geeignetste Datenquelle auszuwählen. D.h., der knotenzentrische Entwurf passt nicht mit dem dominierenden informationszentrischen Anwendungsszenario zusammen. Insbesondere ist in der ursprünglichen Internetarchitektur das Zwischenspeichern von Daten („Caching“) nicht vorgesehen und das Netz kann nicht selbstständig die geeignetste Datenquelle auswählen, was zu ineffizienten, redundanten Mehrfachübertragungen identischer Daten und hohem Datenverkehr führt. Zusätzlich führt die knotenzentrische Architektur teilweise zu unnötig hoher Latenz, hoher Last am Ursprungsserver, und Schwierigkeiten mit Mobilität und Verbindungsabbrüchen. Weitere Probleme sind Anfälligkeit für distributed denial of service (DDoS) Angriffe, der Flash-Crowd Effekt[2] und begrenzte Robustheit, da der Ursprungsserver zum „Single Point of Failure“ werden kann.

Heutzutage wird diesen Problemen hauptsächlich mittels *Applikations-spezifischer* peer-to-peer (P2P) Overlay Netze, *proprietärer, Anbieter-spezifischer* content distribution networks (CDNs) und speziellen Lösungen auf der Anwendungsschicht wie z.B. Lastbalancierung via Hypertext Transfer Protocol (HTTP) begegnet. Diese Lösungen können jedoch nicht das volle Potenzial einer gesamtheitlichen Architektur zur Informationsverteilung ausschöpfen, da die Lösungen jeweils nur einzelne Symptome adressieren anstatt den Kern der genannten Probleme zu lösen. Darüber hinaus sind die Lösungen größtenteils auf einzelne Applikationen, P2P Overlay Netze und Inhalteanbieter beschränkt. Hieraus resultiert u.a. eine ineffiziente Daten-Weiterleitung, eine nur begrenzte Anzahl unterstützter Applikatio-

2 Flash-Crowd bezeichnet den Effekt, dass Inhalte wie z.B. eine Webseite durch eine ungewöhnlich hohe Anzahl an gleichzeitigen Nutzeranfragen überlastet wird und daher für alle Nutzer nicht mehr verfügbar ist.

nen und mangelnde Unterstützung für kleine Inhalteanbieter, die besonders von Problemen wie dem Flash-Crowd Effekt und DDoS betroffen sind. Andere Lösungen wie Proxy Caches erfordern zusätzliche Konfiguration der Clients oder Server. Darüber hinaus fokussieren die genannten Lösungen nur auf die effiziente Informationsverteilung. Die anderen Internet-Probleme wie z.B. Handhabung von Mobilität und Verbindungsabbrüchen erfordern wiederum separate Lösungen. All dies führt zu einer unnötig komplexen Internetarchitektur und in einigen Fällen wie z.B. transparentem Web Caching zu einer Verletzung der Internetprotokoll-Regeln.

Das information-centric networking (ICN) Paradigma versucht diese Probleme zu lösen, indem sie den Fokus von einer knotenzentrischen Architektur auf eine informationszentrische Architektur verschiebt. Der informationszentrische Architekturansatz stellt die Informationsbeschaffung in den Mittelpunkt der Architektur. Daher passt er besser zum dominierenden informationszentrischen Anwendungsszenario im heutigen Internet.

In dieser Arbeit beschreibe ich die Network of Information (NetInf) Architektur, eine neue, informationszentrische Internet Architektur, die die zuvor genannten Probleme auf eine architektonisch einheitliche und saubere Art löst. Die Architektur zielt darauf ab, eine große Anzahl von Applikationen und alle Inhalteanbieter zu unterstützen, und dabei in vielen verschiedenen Anwendungsszenarien nutzbar zu sein.

Zuerst diskutiere und analysiere ich das ICN-Paradigma im allgemeinen. Danach beschreibe ich die NetInf-Architektur im Ganzen. Die NetInf Architektur zeichnet sich durch eine besondere Kombination von Eigenschaften aus, wie unter anderem sein Namensschema, das die Überprüfung der Datenintegrität ermöglicht, ein flexibles Verfahren zur Informationsbeschaffung, dass Namensauflösung mit namensbasiertem Routing verbindet, sein Objektmodell, die Fähigkeit, sich an eine große Anzahl verschiedener Netzszenarien anzupassen, und ein flexibles Modell zur Datenzwischenspeicherung, das sowohl Zwischenspeicherung auf dem Daten- und Namensauflösungspfad unterstützt als auch Zwischenspeicherung außerhalb dieser Pfade. Zwischenspeicher können dabei sowohl in der Anbieter-Infrastruktur als auch auf Nutzerseite sein. Mein besonderer Fokus in dieser Arbeit liegt einerseits auf dem sicheren Namensschema, das die Grundlage für eine sichere, informationszentrische Architektur bildet, und andererseits auf dem Namensauflösungssystem, welches eine zentrale Rolle bei der Informationsbeschaffung spielt aber durch den flachen Namensraum und die hohe Anzahl von erwarteten Objekten nicht trivial ist. Zusätzlich beschreibe und evaluiere ich unseren *OpenNetInf* Prototypen, der die NetInf-Gesamtarchitektur implementiert.

Die Ergebnisse der theoretischen Analyse, der Simulationen und der Prototypimplementierung implizieren, dass ein skalierbares, welt-

weites „Network of Information“ mit 10^{15} Datenobjekten machbar ist, wobei eine Namensauflöselatenz von (deutlich) unter 100 ms erreichbar ist. Die flexible NetInf Architektur hat sich als sehr vorteilhaft herausgestellt, da sie an viele verschiedenen Netz- und Anwendungsszenarien anpassbar ist. Basierend auf diesen Ergebnissen scheint es realistisch zu sein, dass NetInf eine ernsthafte Ergänzung für die heutige Internetarchitektur anbieten kann.

The best way to predict the future is to invent it.

— Alan Curtis Kay

ACKNOWLEDGMENTS

I thank my thesis supervisor Prof. Dr. Holger Karl for his constant support and many helpful discussions and Prof. Dr. Jörg Ott for serving as second advisor. Likewise, I thank all my coauthors as well as my colleagues from the Architecture and Design for the Future Internet (4WARD) project and Scalable and Adaptive Internet SoLutions (SAIL) project for the fruitful and constructive collaboration. Doing research without all of you would have been a lonely and definitely less joyful experience. Thanks also to my colleagues Matthias Herlich and Thorsten Biermann for the good collaboration in supervising our students project groups, which has let to a successful OpenNetInf prototype implementation. The successful prototype implementation would have obviously also not been possible without the many dedicated students of the AugNet I, AugNet II, and NetInf project group. Thanks for your dedication and great implementation work. Finally, I thank my colleagues from the computer networks group at the University of Paderborn for many good discussions, proofreading of this thesis, and their mental support.

CONTENTS

1 INTRODUCTION 1
1.1 Shortcomings and Pitfalls of Today's Internet 1
1.2 The Information-Centric Networking Paradigm 3
1.3 Main Components of ICN Designs 5
1.4 Problem Statement and Requirements 8
1.5 Overview of Information-Centric Network Designs . . 11
1.5.1 Data-Oriented Network Architecture 12
1.5.2 Content-Centric Networking 12
1.5.3 Publish–Subscribe Internet Routing Paradigm . 14
1.5.4 Network of Information 14
1.6 Thesis Contributions 15
2 NETWORK OF INFORMATION ARCHITECTURE 23
2.1 Overview 23
2.1.1 Design Principles 23
2.1.2 Architecture Overview and Sample Setup . . . 28
2.2 Elements of the Network of Information 30
2.2.1 Named Data Objects 30
2.2.2 Basic Naming and Security 32
2.2.3 Advanced Naming and Security 33
2.2.4 NetInf Communication 34
2.2.5 Caching 37
2.2.6 Name Resolution 39
2.2.7 Inter-Domain Communication 40
2.2.8 Search 41
2.3 Related Work 42
2.3.1 General Overview of ICN-Related Work 43
2.3.2 ICN Architectures: Design Choices and Trade-Offs 47
2.4 Summary 53
3 SECURE NAMING 55
3.1 Introduction 55
3.2 Requirements 56
3.3 Naming Scheme 57
3.3.1 Basic Concepts 57
3.3.2 ID Structure 58
3.3.3 Security Metadata Structure 59
3.4 Analysis of Security Properties 60
3.4.1 Name–Data Integrity 60
3.4.2 Name Persistence 61
3.4.3 Owner Pseudonymity and Identification 63
3.5 Evaluation 65
3.6 ni URI Scheme 65

3.7 Related Work . . . 66
3.8 Summary . . . 68
4 NEIGHBORHOOD EFFECT – LOCALITY IN DNS REQUESTS 71
4.1 Introduction . . . 71
4.2 Measurement Setup . . . 73
4.3 Data Evaluation . . . 74
4.3.1 Data Preprocessing . . . 74
4.3.2 University DNS Zone . . . 75
4.3.3 Computer Science Department DNS Zone . . . 82
4.4 Related Work . . . 87
4.5 Summary . . . 89
5 HIERARCHICAL NAME RESOLUTION 93
5.1 Introduction . . . 93
5.2 Requirements . . . 94
5.3 Hierarchical NRS Architecture . . . 96
5.3.1 General NRS Framework . . . 97
5.3.2 MDHT . . . 103
5.3.3 HSkip . . . 104
5.3.4 System Comparison: MDHT vs. HSkip . . . 109
5.4 Global Name Resolution . . . 110
5.5 Load Balancing . . . 111
5.6 Scalability and Node Performance Analysis . . . 112
5.7 System Analysis . . . 114
5.7.1 General Analysis Approach . . . 114
5.7.2 Independent MDHT . . . 117
5.7.3 HSkip and Entangled MDHT . . . 117
5.7.4 Analysis Results . . . 118
5.8 Simulation . . . 122
5.8.1 Simulation Setup and Assumptions . . . 123
5.8.2 Results: Latency . . . 124
5.8.3 Results: Work Load Distribution . . . 127
5.9 Related Work . . . 127
5.10 Summary . . . 130
6 PROTOTYPING 133
6.1 Introduction . . . 133
6.2 OpenNetInf Prototype . . . 134
6.2.1 Overview . . . 134
6.2.2 Interfaces . . . 134
6.2.3 Named Data Objects and Security . . . 136
6.2.4 Name Resolution and Metadata Storage . . . 137
6.2.5 Caching . . . 138
6.2.6 Data Transfer . . . 138
6.2.7 Additional Services . . . 139
6.3 Application Development . . . 140
6.3.1 Media Distribution . . . 140
6.3.2 InFox . . . 141

6.3.3 InBird 143
6.3.4 Context-Aware Applications 144
6.4 Evaluation 148
6.4.1 Measurement Setup 149
6.4.2 Results 150
6.5 Related Work 154
6.6 Summary 155
7 CONCLUSION AND FUTURE WORK 157
7.1 Deployment 157
7.2 Summary and Implications 159
7.3 Future Work 160

BIBLIOGRAPHY 163

LIST OF FIGURES

Figure 1 ICN communication model – client side . . . 4
Figure 2 DONA overview when caching on all resolution handlers (RHs) . . . 12
Figure 3 CCN overview . . . 13
Figure 4 PSIRP overview . . . 14
Figure 5 NetInf overview . . . 15
Figure 6 NetInf protocol stack, assuming a node with two convergence layers over two different underlays . . . 26
Figure 7 NetInf example message flow . . . 29
Figure 8 NetInf sample network setup . . . 31
Figure 9 Conceptual Object Model . . . 32
Figure 10 Example NetInf convergence layers . . . 35
Figure 11 NetInf inter-domain scenario (triangles = client nodes; hexagons = NetInf routers; green rectangle = destination node; gray ovals = networks; yellow notes = requests; gray note = NRS bindings) . . . 41
Figure 12 Basic ID structure . . . 58
Figure 13 Certificate chain with two certificates, containing three owners in total . . . 63
Figure 14 DNS log example: computer science DNS zone (full hash values elided in figure for space reasons) . . . 74
Figure 15 Uni: All requests, *filtered* . . . 76
Figure 16 Uni: DNS requests by user devices and servers 76
Figure 17 Uni: DNS requests by servers for external host names . . . 77
Figure 18 Uni: DNS requests by servers for internal host names . . . 78
Figure 19 Uni: DNS requests by user devices . . . 79
Figure 20 Uni: All requests, *unfiltered* . . . 80
Figure 21 Uni: Only reverse lookup requests . . . 81
Figure 22 Uni: Reverse lookups by requester type . . . 81
Figure 23 Uni: DNS requests by university-external requesters . . . 82
Figure 24 IRB: All DNS requests, *filtered* . . . 83
Figure 25 IRB: All DNS requests, *unfiltered* . . . 84
Figure 26 IRB: Reverse lookups . . . 84
Figure 27 IRB: DNS requests by user devices . . . 85
Figure 28 IRB: DNS requests by IRB servers . . . 86

Figure 29 IRB: DNS requests by IRB LDAP servers 86
Figure 30 Binding entries and resolution of object X . . . 99
Figure 31 HSkip hierarchy with GET request for object A 107
Figure 32 Analysis: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 1–9 levels (L); LP=0; dashed line = MDHT, solid = HSkip 119
Figure 33 Analysis: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 3–9 levels (L); LP=0.3; dashed = MDHT, solid = HSkip 120
Figure 34 Analysis: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 1–9 levels (L); 12 million nodes; dashed = MDHT, solid = HSkip 122
Figure 35 Analysis: Latency of REX+MDHT-O ($\mathcal{O}(1)$) and HSkip; 1–9 levels (L); 12 million nodes; dashed = MDHT, solid = HSkip 123
Figure 36 Simulation: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 3–9 levels (L); LP=0.3; dashed = MDHT, solid = HSkip 125
Figure 37 Simulation: Latency MDHT ($\mathcal{O}(\log n)$), HSkip: 1500 nodes; 3–9 levels (L); dashed = MDHT, solid = HSkip 126
Figure 38 Simulation: Relative number of used levels; 1500 nodes; 3–9 levels (L); dashed = MDHT, solid = HSkip 126
Figure 39 No. of requests per node: MDHT, LP=0 127
Figure 40 No. of requests per node: MDHT, LP=0.3 . . . 128
Figure 41 No. of requests per node: HSkip, LP=0 128
Figure 42 No. of requests per node: HSkip, LP=0.3 . . . 129
Figure 43 OpenNetInf node with *NetInf Core* and *AddOn* components, connected to other NetInf nodes via the N2N interface 135
Figure 44 Video streaming in Firefox via OpenNetInf . . 142
Figure 45 Shopping application use case scenario 145
Figure 46 Mockup of AugNet application showing an information overlay for the Eiffel Tower 147
Figure 47 AugNet browser application showing the users current position (red dot) and two retrieved virtual entities (blue dots) in the user's search area (orange circle area) 148
Figure 48 Measurement scenario and hierarchical Multi-Level Distributed Hash Table (MDHT) setup (Cl = client) 150
Figure 49 Cumulative inter-domain traffic between levels 1 & 2 and levels 2 & 3 151
Figure 50 Inter-domain traffic between levels 1 & 2 . . . 153
Figure 51 Inter-domain traffic between levels 2 & 3 . . . 153

Figure 52 Evaluation overview. The numbers indicate the sections of this thesis where details can be found. Light boxes (number 2.2.8 and 5.5) indicate evaluations that are only summarized in this thesis. 160

LIST OF TABLES

Table 1 Summary of characteristics of ICN approaches. 54
Table 2 Content matrix for top 2000 most popular host names. Each line provides the percentage of all requests that originate from a given continent. Columns indicate the continent from where content is served (see Table 1 in reference [1]). 89

LISTINGS

Listing 1 Creating name–data integrity for dynamic content 60
Listing 2 Verifying name–data integrity of dynamic content 61
Listing 3 Compatible NetInf link (NetInf ID is curtailed) 142

ACRONYMS

4WARD Architecture and Design for the Future Internet 11
ADSL Asymmetric Digital Subscriber Line 152
ALTO Application-Layer Traffic Optimization 38
AS autonomous system 98
AN access node 98
API application programming interface 4
AugNet Augmented Internet 32

BBT balanced binary tree . . . 113
BGP Border Gateway Protocol . . . 36
CBN Content-Based Networking . . . 45
CCN Content-Centric Networking . . . 11
CDN content distribution network . . . 2
CL convergence layer . . . 25
CLB constrained load balancing . . . 105
CMS Cryptographic Message Syntax . . . 31
CNF Cache-and-Forward . . . 45
CONNECT Content-Oriented Networking: a New Experience for Content Transfer . . . 44
CoAP Constrained Application Protocol . . . 19
COAST Content Aware Searching retrieval and sTreaming . . . 44
COMET COntent Mediator architecture for content-aware nETworks . . . 44
CoRE Constrained RESTful Environments . . . 19
CRC cyclic redundancy check . . . 26
DAG directed acyclic graph . . . 98
DDoS distributed denial of service . . . 3
DECADE Decoupled Application Data Enroute . . . 19
DFZ default-free zone . . . 40
DHCP Dynamic Host Configuration Protocol . . . 30
DHT distributed hash table . . . 45
DNS Domain Name System . . . 9
DOI digital object identifier . . . 56
DONA Data-Oriented Network Architecture . . . 11
DoS denial of service . . . 6
DPSP DTN Pub/Sub Protocol . . . 46
DSL Digital Subscriber Line . . . 38
DTN delay-tolerant networking . . . 10
EU European Union . . . 17
FIT Future Internet Toolbox . . . 21
FTP File Transfer Protocol . . . 139
GIN Global Information Network . . . 53
GPS Global Positioning System . . . 147
HDHT hierarchical DHT . . . 130

HIP Host Identity Protocol . . . 67
HSkip Hierarchical SkipNet . . . 20
HTTP Hypertext Transfer Protocol . . . 10
I3 Internet Indirection Infrastructure . . . 67
ICN information-centric networking . . . 1
ID identifier . . . 20
IP Internet protocol . . . 1
ISP Internet service provider . . . 89
LDAP Lightweight Directory Access Protocol . . . 77
LLC Late Locator Construction . . . 28
LP level probability . . . 116
M2M machine-to-machine . . . 2
MAAN Multi-Attribute Addressable Network . . . 42
MDHT Multi-Level Distributed Hash Table . . . 20
MIME Multipurpose Internet Mail Extensions . . . 31
N2N node-to-node . . . 134
NBR name-based routing . . . 14
NDN Named-Data Networking . . . 11
NDO named data object . . . 5
NetInf Network of Information . . . 1
ni named information . . . 18
NNRP NEC NetInf Router Platform . . . 154
NNTP Network News Transfer Protocol . . . 87
NPO named person object . . . 143
NRS name resolution service . . . 1
OSPF Open Shortest Path First . . . 36
OWL Web Ontology Language . . . 31
P2P peer-to-peer . . . 1
PIT Pending Interest Table . . . 13
PPSP Peer-to-Peer Streaming Protocol . . . 19
P4P provider portal for P2P . . . 38
PDU protocol data unit . . . 25
PK public key . . . 48
PKI public key infrastructure . . . 6
PoP point of presence . . . 45
PSIRP Publish-Subscribe Internet Routing Paradigm . . . 11

PURSUIT Publish-Subscribe Internet Technology 11
QoS quality of service 44
RDF Resource Description Framework 31
REST Representational State Transfer 136
REX Resolution Exchange 93
RFID radio-frequency identification 147
RH resolution handler 12
RIPE NCC Réseaux IP Européens Network Coordination Centre . 87
SAIL Scalable and Adaptive Internet SoLutions 11
SDSI Simple Distributed Security Infrastructure 66
SHA Secure Hash Algorithm 10
SMTP Simple Mail Transfer Protocol 78
SPKI Simple Public Key Infrastructure 67
SIENA Scalable Internet Event Notification Architecture 46
SIONA Service and Information Oriented Network Architecture . 44
SK secret key 58
SPARQL SPARQL Protocol and RDF Query Language 139
SSD solid state disk 113
TCP Transmission Control Protocol 27
TRIAD Translating Relaying Internet Architecture integrating Active Directories 43
TLS transport layer security 10
URL uniform resource locator 6
URI uniform resource identifier 9
VoD video on demand 1
VoIP voice over IP 161
XIA eXpressive Internet Architecture 44
XML Extensible Markup Language 31

1 INTRODUCTION

This chapter is based on work published in references [2, 3, 4].

This thesis describes the Network of Information (NetInf) architecture, an information-centric networking (ICN) architecture for the future Internet. Chapter 1 gives an introduction to information-centric networking in general and describes the contribution and content of this thesis in more detail in Section 1.6. Chapter 2 describes the overall NetInf architecture. Secure naming and a scalable name resolution service (NRS) are critical components of this architecture. Hence, I subsequently focus on these two aspects. In Chapter 3, I describe our secure naming scheme that is based on a flat namespace. Chapter 4 evaluates request patterns for data objects that are important input to the design of the NRS framework and two specific incarnations of this framework in Chapter 5. To evaluate the overall NetInf architecture, we have developed a prototype of the NetInf architecture (called *OpenNetInf*) and several ICN applications, described in Chapter 6.

This chapter gives an overview of the background and motivation for this thesis and the ICN paradigm in general. In the following sections, I give a brief introduction to the main problems and shortcomings of today's Internet (Section 1.1), followed by an overview of the ICN paradigm (Section 1.2) and the main components of ICN designs (Section 1.3). Based on this overview, I discuss the problem statement and requirements for information-centric network architectures in general (Section 1.4), followed by an overview of the main recent ICN designs (Section 1.5). In Section 1.6, I state the contributions of this thesis and list the publications where this work has first been published.

This thesis mostly uses the first-person plural (i.e. "we") instead of the first-person singular as parts of the work have been performed together with the co-authors stated in Section 1.6.

1.1 SHORTCOMINGS AND PITFALLS OF TODAY'S INTERNET

According to recent predictions [5], global Internet protocol (IP) traffic will approach 966 exabytes per year in 2015. Much of this traffic stems from various forms of video, including TV, video on demand (VoD), Internet video, and distribution via peer-to-peer (P2P), that will continue to account for approximately 90% of global con-

sumer traffic by 2015. Global mobile data traffic [6] is expected to increase by a factor of 26 from 2010 to 2015. Likewise, machine-to-machine (M2M) traffic is expected to grow 22-fold from 2011 to 2016, with a compound annual growth rate of 86%, and the number of mobile-connected M2M modules is expected to grow 5.8-fold between 2011 and 2016, reaching 1,9 billion.

The best current practice to manage this growth in terms of data volume and devices is to employ overlays (i.e., networks built on top of IP) such as content distribution networks (CDNs), P2P networks, and M2M application platforms, which cache content, provide location-independent access to data, and optimize its delivery. In principle, such platforms provide a service model of *accessing data objects* (e.g., videos, web pages, documents, M2M data) instead of a host-to-host packet delivery service model.

However, P2P networks (e.g. BitTorrent) and M2M application platforms are typically limited to specific applications and each application has its own underlying P2P/M2M solution. Likewise, CDN deployment is provider-specific and multiple proprietary CDN solutions exist with currently no or only limited cooperation between the different CDNs.

Therefore, since this functionality resides in application-specific or provider-specific overlays only, the full potential of content distribution and M2M application platforms cannot be leveraged:

- The network is not aware of the semantics of data requests/transmissions and which data is requested/transmitted. Even deep packet inspection would not provide all required information as a consistent, persistent, and location-independent naming of data objects is missing. Hence, the network is unable to efficiently manage data access and transmission and such attempts typically lead to layer violations.
- Data has to travel suboptimal routes imposed by the overlay topology rather than by the IP-layer topology.
- Multicast and broadcast features of wireless networks cannot be leveraged, i.e., request and delivery for the same object have to be made multiple times.
- The overlay functionality is only available for selected providers and selected applications.
- Overlays typically require a significant amount of infrastructure support, e.g., authentication portals, content storage, and applications servers, making it often impossible to establish local, direct communication.
- Many applications provide their own approach to caching, replication, transport, authenticity validation (if at all), although

they all share the same model of accessing data objects in the network.

1.2 THE INFORMATION-CENTRIC NETWORKING PARADIGM

The ICN paradigm builds on the following two basic changes in recent years:

CHANGE OF HARDWARE ASSUMPTIONS In any modern communication network, there are three main aspects: processing power in network nodes, storage capacity of network nodes, and capacity of transmission links. It is well known that the capacity of processing power (including the number of transistors and their speed) evolves according to Moore's law, doubling every 18 months[1]. Optical transmission evolves twice as fast and the capacity of hard disk storage at an even faster rate. At the time the Internet was designed, memory and storage was expensive and the use of it had to be minimized in network nodes. This is no longer true. It is reasonable to assume that this should be reflected in future network architectures.

CHANGE OF USAGE MODEL Today, users' prime interest is to retrieve a specific data object such as a song, a movie, or a web page. The users do not really care which network node is delivering the requested object. However, the Internet was initially designed for interconnecting terminals and servers, not to be used for mass distribution of information.

The ICN paradigm is about shifting the focus from interconnecting nodes to node-independent information retrieval. The ICN paradigm puts information at the center of the architecture and provides access to data objects as a first-class networking primitive, i.e., it is *information-centric*. In contrast, current networks are *host-centric* where communication is based on addressing and connecting to specific, named hosts, for example web servers, PCs, and mobile handsets.

The aim of ICN approaches is to develop a network architecture that is better suited for efficiently accessing and distributing content and that better copes with disconnections, disruptions, mobility, flash-crowd effects, distributed denial of service (DDoS) attacks, and inadequate security in the communication service.

This is achieved by embedding many of the required functions deeper into the network fabric, including secure object naming, in-network caching, multi-party communication, and interaction models that decouple senders and receivers. The ICN approach makes these

1 To be more precise, the 18 months period is based on Intel executive David House, who has included both the number of transistors and their speed in his prediction, while Moore only referred to the number of transistors, hence, predicting 24 months.

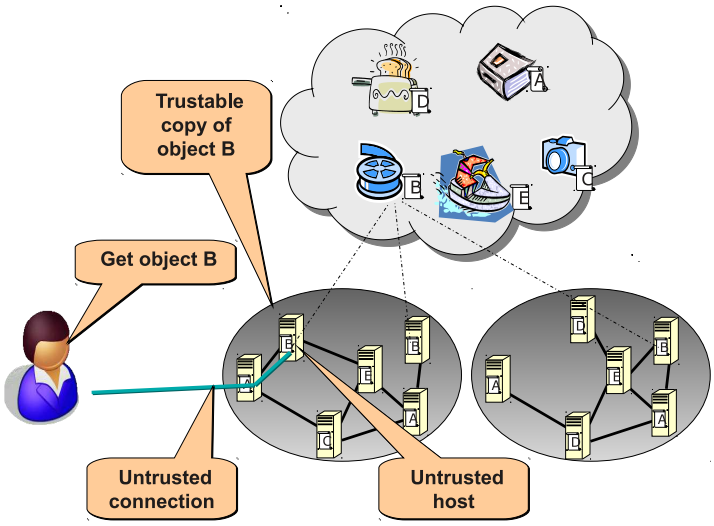

Figure 1: ICN communication model – client side

functions accessible for developers via an application-agnostic application programming interface (API) that inherently includes functionality such as caching, best-copy selection, and information-centric security. This is complemented by functions and interfaces for network operators to reduce network (in particular, inter-domain) traffic and to potentially simplify traffic engineering. Communication is driven by receivers *requesting* data objects. Senders make data objects available to receivers by *publishing* the objects.

As illustrated in Figure 1, the network can satisfy client requests with data from any source holding a copy of the object, enabling efficient and application-independent caching as part of the network service. The integrity of the delivered data is established independently of the delivering host. Hence, the delivering host can be untrusted without jeopardizing security.

Media/information distribution has become the dominating use case in today's Internet. However, efficient information distribution services are today only available in overlay systems such as application-specific P2P overlays and proprietary CDNs. ICN can be seen as a generalization of P2P and CDN technologies that integrates functionality for efficient and reliable media/information distribution into the network architecture, thereby solving the problem of proprietary CDN solutions and application-specific P2P solutions in an architecturally sound way.

ICN is not limited to large-scale media distribution scenarios but is useful in a wide variety of scenarios. Such scenarios include, e.g., any kind of ad hoc setup that can benefit from efficient direct data communication without requiring extra infrastructure nodes or complicated network setup. This also includes large-crowd scenarios like

a soccer game where the infrastructure is challenged by the temporarily large number of users.

Also, scenarios where accessing and modifying local caches gives advantages (e.g., reduced latency and traffic) over remote server accesses can benefit from the ICN approach. In scenarios with intermittent Internet connectivity, remote server access might not be feasible at all, hence, an ICN-based solution has strong advantages over a remote-server-based solution. To illustrate, consider a group of collocated users (with good local connectivity (e.g. WiFi) and weak Internet connectivity) collaboratively editing documents stored in a version control repository (e.g., Subversion[2]) or a Wiki. Traditionally, all updates would go via the slow Internet link only to be redistributed locally again. With an ICN infrastructure, the relevant objects are cached and modified locally, with local copies being created opportunistically at nearby places, without the application having to worry. Obviously, such an ICN-based solution has to handle related aspects like dealing with consistency of the different copies at some point.

1.3 MAIN COMPONENTS OF ICN DESIGNS

Most ICN architectures, albeit different in details, share general design ideas and main components. We describe these ideas and components in this section, introduce the relevant ICN terminology, and bring up some of the major design choices.

NAMED DATA OBJECTS The main abstraction of ICN is the *named data object (NDO)*. Examples are web pages, documents, movies, photos, songs, as well as streaming and interactive media, in other words, all type of objects that are stored and accessed via computers. In addition to their main content, there can be *metadata* associated with NDOs, i.e., data describing/complementing the main content, e.g., author, creation date, or other data about the represented content.

The NDO is independent from location, application program, and transportation method. This means that an NDO keeps its name regardless of its location and regardless of how it is copied, stored, and communicated. It also means that any two copies of an NDO are for all purposes equivalent; for instance, any node holding a copy can supply it to a requester. The *NDO granularity* varies between approaches from packet size to full files.

From a networking perspective, these objects can be viewed as named data chunks without semantics, but some ICN designs have an *object model* that allows to represent abstract information. For example, such an object model allows to create an NDO that represents a video independent of its specific encoding (i.e., unique bit pattern), or different recordings of a piece of music.

2 http://subversion.apache.org/

NAMING AND SECURITY Naming data objects is as important for ICN as naming hosts is for today's Internet. Fundamentally, ICN requires *unique names* for individual NDOs, since names are used for identifying objects independently of its location or container. It is important to establish a verifiable binding between the object and its name (*name–data integrity*), so that a receiver can be sure that received bits actually represent the named object (*object authenticity*). In this case, trust is directly based on the data and its name and not on the network nodes/connections that deliver the data; hence, we call this *information-centric security* here.

Information about an object's *owner*, i.e., who generated or published it, is also useful to associate to the name. If the owner is identified via the owner's real-world identify (i.e., the real, uniquely identified name) in a trustworthy manner (e.g., via trusted certificates), it is called *owner identification* here. If the owner is only identified via a pseudonym, it is called *owner pseudonymity*. Note that there are other roles besides the owner, such as the *producer* (creating/generating the object) or the *publisher* (publishing the object in the network) that can but do not have to be identical with the owner role.

Name–data integrity is fundamentally required for an information-centric network to work reliably – otherwise neither network elements nor receivers can trust objects' authenticity, which would enable several attacks including denial of service (DoS) attacks, e.g., by injecting a large number of spoofed objects into the network. There are different ways to use names and cryptography to achieve the desired functions [7], and there are different ways to manage namespaces correspondingly.

Two naming concepts have largely been proposed: one with a *hierarchical* and one with a *flat namespace*. The hierarchical scheme has a structure similar to current uniform resource locators (URLs), where the hierarchy is rooted in a publisher prefix, i.e., each name has a publisher prefix which is unique for each publisher. The hierarchy enables *aggregation* of routing information, hence, improving scalability of the routing system by reducing the size and update rate of the routing tables. In some cases, the names are *human-friendly*, which makes it possible for users to manually type in names, and, to some extent, assess the relation between a name and what the user wants.

The flat naming scheme is *self-certifying*, meaning that the object's name–data integrity can be verified without needing a public key infrastructure (PKI) or other third party to first establish trust in the key. Self-certification is achieved by binding a hash of the content to the object's name. This can be done by directly embedding the hash of the content in the name. Another option is an indirect binding, which embeds the public key of the publisher in the name and signs the content with the corresponding secret key. The resulting

names are typically non-hierarchical, or *flat*, although the publisher field provides structure that can be used for *routing aggregation*.

There are delicate design trade-offs for ICN naming, affecting routing and security. Self-certifying names are typically neither human-friendly nor hierarchical. They can, however, provide some structure for aggregation, for instance, a name part corresponding to a publisher as mentioned above. Without self-certification, as already mentioned above, the infrastructure depends on a PKI for its operation, which many consider to be a major disadvantage.

APPLICATION PROGRAMMING INTERFACE The ICN *API* is defined in terms of requesting and delivering NDOs. The publisher makes an NDO available to others by publishing it to the network (called `PUBLISH` or `REGISTER` in different approaches). A client/consumer asks for an NDO by name (called `GET`, `INTEREST`, `REQUEST`, `FIND`, or `SUBSCRIBE`)[3]. Asking for an NDO is in most ICN designs a synchronous, one-time operation. However, some approaches build on a more *publish/subscribe*-like approach, where the client registers a subscription and gets notified when something is available.

Both types of operations (subsequently referred to as `PUBLISH` and `GET`, unless a specific approach is indicated) use the object's name as main parameter. In addition, some approaches support supplemental parameters. For example, the CURLING [8] approach supports location preferences for scoping and filtering publications and requests.

ROUTING, FORWARDING, AND NAME RESOLUTION Routing/forwarding can be divided into two phases: (1) *routing/forwarding of NDO requests* and (2) *routing/forwarding of NDOs* back to the requester. There are two general approaches in ICN to handle routing/forwarding during those phases; both strongly depend on the properties of the object namespace, in particular if the names are aggregatable or not.

The first approach is commonly called *name resolution* and creates bindings of a name from one namespace into another namespace. More specifically, it uses an *NRS* that stores bindings from object names to (typically) topology-based *locators* pointing to corresponding storage locations in the network. Other types of bindings are also possible as will be discussed later.

The name resolution approach has three conceptual phases: (1a) forwarding the request message to the responsible NRS node where the object name is translated into one or several object copy locators (typically locators of lower network layers), (1b) forwarding the request message to the object copy(s), and (2) forwarding the data from the source(s) to the requester. All phases can be based on differ-

3 Obviously, this characterization ignored some differences in details of the approaches' different API primitives for simplification purposes.

ent routing algorithms. There are multiple alternatives to loosely or tightly integrate these phases in an ICN architecture.

The second general approach forwards the request from the requester directly to one or multiple object copy(s) in the network based on the requested object name during phase 1a, without first resolving the object name into some lower-layer locators (i.e., skipping the name resolution step 1b). The request is forwarded based on *routing tables* in the network that contain bindings between names of the same namespace (i.e., the object namespace). This approach is often referred to as *name-based routing*. The routing algorithm used for this approach heavily depends on the properties of the namespace. After the source has received the request message, the data is routed back to the requester, equaling phase (2) in the NRS-based approach.

CACHING Caching named data objects is an integral part of the ICN service. All nodes can have caches (depending on the specific ICN approach), including nodes in operator-run infrastructure networks (*in-network cache*) as well as user-run home networks and mobile terminals (*peer-side cache*). Requests for NDOs can be satisfied by any node holding a copy in its cache. ICN thus combines caching at the network edge, like in P2P and other overlay networks, with in-network caching, for instance transparent web caches. Caching is generic; i.e., it is application-independent and applies to all providers of content, including user-generated content.

ADDITIONAL SERVICES Depending on the actual ICN approach, additional components that are external to today's Internet architecture might become part of an ICN network architecture. For example, persistent data *storage* might be closer integrated with the network service, including a closer integration with caching and name resolution. Likewise, Google-like information *search* might be closer integrated, especially when metadata about NDOs is stored in the ICN network.

1.4 PROBLEM STATEMENT AND REQUIREMENTS

This section discusses the major problems of today's Internet architecture. Some of these problems have been mentioned earlier for motivation purposes but are summarized and discussed in more detail here again to include a consistent overview. The section focuses on problems that can be addressed by an information-centric network architecture. In conjunction, we explain the main requirements that an information-centric network architecture has to fulfill and how the ICN paradigm addresses these problems and requirements in general.

SCALABLE AND EFFICIENT CONTENT DISTRIBUTION The original Internet architecture does not incorporate caching nor the ability to automatically select the most appropriate data copy. This leads to unnecessarily high traffic and latency, and inefficient, redundant transmission of identical content. The tremendous traffic increase described in Section 1.1 necessitates more efficient content dissemination mechanisms that scale in traffic, latency, cost, and energy usage. There are two main developments: CDNs and P2P networking, both related to an information-centric model. Yet there are a number of issues. P2P networks suffer from suboptimal peer selection that leads to expensive inter-provider traffic and heavy loads on weak access links (e.g., mobile and cable networks), currently do not effectively support in-network caching, and suffer from incompatibility of the many existing systems as P2P networks are generally an application-specific solution. CDNs extend the underlying network infrastructure by interpreting uniform resource identifiers (URIs) and Domain Name System (DNS) names to access cached copies of content. However, CDNs are limited to a set of explicitly created copies and cannot benefit from all available copies, e.g., on user devices or other servers. Moreover, CDNs are only an add-on to today's Internet; they solve a limited subset of the general problem and only for a restricted group of customers: Since a CDN requires dedicated setup, configuration, and customer relationships, typically only large players (e.g., large content providers) use it. Small players most in need of support (e.g., to protect their small infrastructure against the flash crowd effect and DDoS) are least likely to benefit from this approach. Hence, a solution that supports all kinds of players and, e.g., eases experiments with new services for new players would be much more favorable.

Looking at the need for scalable and efficient content distribution, the question is: if users are more interested in accessing named content, regardless of endpoint locators, is there a more architecturally sound way of addressing these requirements that does not require individual amendments for specific domains and architectures? ICN is the attempt to answer this question with "yes".

PERSISTENT, LOCATION-INDEPENDENT NAMING Today's Internet architecture lacks a persistent, location-independent scheme to name content. Most content URIs in today's network directly point to a specific location; they are *locators*. As a result, the name–object binding can easily break, for example, when an object is moved, the site changes domain, or the original site is unreachable for some reason. Moreover, independent replicas of an object at different web servers might be accessible only under different URIs. This essentially makes them appear as different objects. This complicates efficient network utilization and caching.

The ICN approach overcomes these problems with persistently and uniquely named data objects and with its service model that decouples producers from consumers.

DATA AVAILABILITY In today's Internet, data access in challenging network conditions with spotty connectivity, network disruptions, or large network delays is difficult or impossible. This is often caused by inaccessibility of global infrastructure like DNS, inability to use locally available copies, and a required persistent connection between hosts.

However, not all applications actually do require seamless end-to-end communication. If the primary objective is access to data objects, ICN with its in-network caching can offer store-and-forward approaches similar to the delay-tolerant networking (DTN) architecture [9] with its convergence layer concept for hop-by-hop transport. This can provide better reliability and better performance by leveraging optimized hop-by-hop transport and in-network caching. In addition, caching can also reduce the load at the origin server and the problem of the origin server being a single point of failure, hence, increasing overall robustness and data availability. This is (to a lesser extent) still true when mechanisms like Hypertext Transfer Protocol (HTTP)-based load balancing are used.

SECURITY Today's security is host-centric, i.e., based on securing channels between hosts via encryption like transport layer security (TLS) and trusting servers via authentication. As a consequence, a client cannot trust a copy that is received from an untrusted location although the copy might be authentic. To address this problem, workarounds like separately published fingerprints of the data (e.g., Secure Hash Algorithm (SHA)-2) are sometimes used.

A future security model should provide an architecturally sound approach that works for any data copy independent of its location. The model should enable ubiquitous caching while retaining data integrity and authenticity, something that the current model does not provide. In addition, it should not have to rely on trusted third parties for data integrity checking. In contrast, in today's security model, trust is practically transfered to the software vendor that compiles the set of trusted authorities.

As a result of the inherent split between object location and object naming, the ICN paradigm is well suited to offer an architecturally sound solution addressing these requirements.

MOBILITY AND MULTIHOMING Due to the host-based nature of today's Internet, mobility and multihoming of nodes and networks is a challenging problem that requires managing end-to-end data flows (e.g., with handovers) and choosing which path/interface to

use. Several solutions like mobile IP [10] have been proposed and implemented; however, they suffer from disadvantages like increased routing stretch and complexity due to the inherent problems of the host-centric approach.

The ICN approach does not enforce end-to-end connections that require this kind of connection management. The problem can, thus, become much simpler. A moving client just continues to issue requests for named data objects on a new access. Requests on the new access is potentially served from a different source, instead of needing to maintain a connection to the previous source. A multi-homed client can similarly choose to send a request on any one, several, or all accesses.

1.5 OVERVIEW OF INFORMATION-CENTRIC NETWORK DESIGNS

To provide a general understanding of existing ICN approaches, this section gives a high level overview of the main existing ICN approaches. This also includes a first glimpse at the NetInf architecture to provide a first understanding and context in relation to the other approaches. NetInf is subsequently described in detail in Chapter 2. A more specific, topic-related discussion of related work can be found in each individual chapter, including a broader discussion of work related to the overall NetInf architecture in Section 2.3.

Although it can be argued that the ICN approach was pioneered in TRIAD [11], this overview is based on the following, more recent projects representing four approaches being actively developed:

- Data-Oriented Network Architecture (DONA) [12], currently continued in related research [7, 13],
- Content-Centric Networking (CCN) [14], currently developed in the Named-Data Networking (NDN) project [15],
- Publish-Subscribe Internet Routing Paradigm (PSIRP) [16], continued in the Publish-Subscribe Internet Technology (PURSUIT) project [17], and
- Network of Information (NetInf) [3], started in the Architecture and Design for the Future Internet (4WARD) project [18] and continued in the Scalable and Adaptive Internet SoLutions (SAIL) project [19].

In CCN, the term *content-centric* is used instead of *information-centric*, and DONA uses the term *data-oriented*. Henceforth, this thesis uses *information*, *content*, and *data* interchangeably.

The following sections are only intended to provide a rough overview comparing the basics of the different approaches to get a general overview. It contains simplifications and may in some cases not take the latest developments of respective approaches into account as the approaches are still under development.

1.5.1 *Data-Oriented Network Architecture*

In DONA, NDOs are published into the network by the sources. Nodes that are authorized to serve data register with the resolution infrastructure, consisting of multiple resolution nodes called resolution handlers (RHs). Requests (called *FIND* packets) are routed by name towards the appropriate RH, as illustrated in Figure 2, steps 1–4. Data is sent back in response, either through the reverse RH path (steps 5–8) enabling caching, or over a more direct route (step 9).

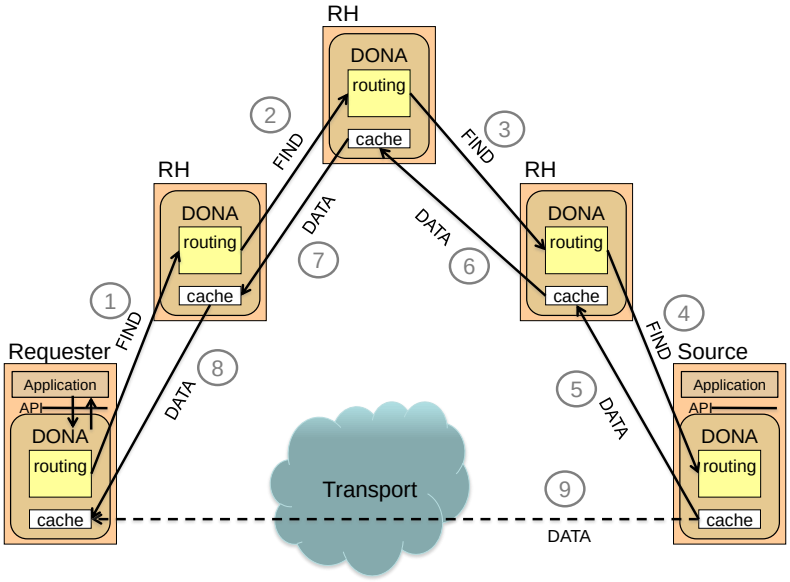

Figure 2: DONA overview when caching on all resolution handlers (RHs)

Objects from the same content provider have a common prefix. Hence, content providers can perform a wildcard registration of their name prefix in the RH, so that queries can be directed to them without needing to register specific objects. It is also possible to register NDO names before the NDO content is created and made available.

Register commands have expiry times. When the expiry time is reached, the registration needs to be renewed. The RH resolution infrastructure routes requests by name in a hierarchical fashion and tries to find a copy of the content closest to the client. DONA's anycast name resolution process allows clean support for network-imposed middleboxes (e.g., firewalls, proxies).

1.5.2 *Content-Centric Networking*

In CCN, NDOs are published at nodes and routing protocols are employed to distribute information about NDO locations. Routing in

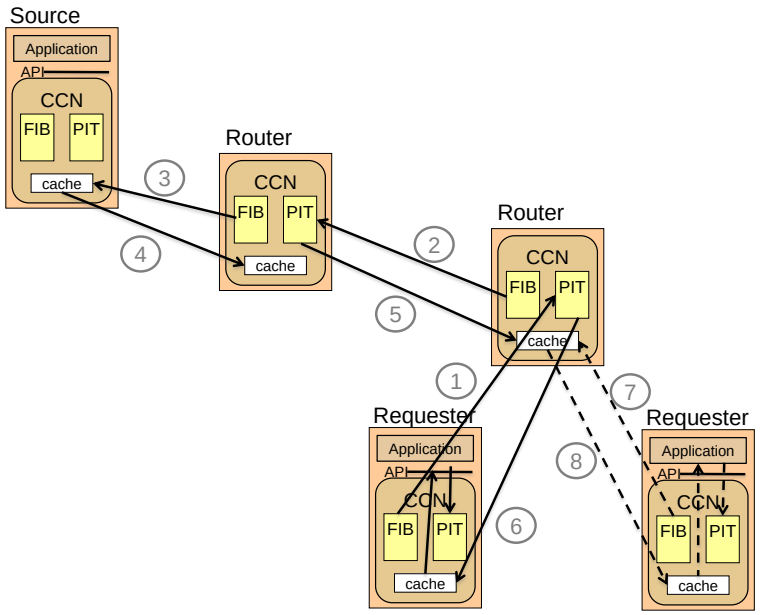

Figure 3: CCN overview

CCN can leverage aggregation through a hierarchical naming scheme. NDO authenticity is achieved through public key cryptography. Trust in keys can be established via different means which provide a different level of trust, e.g., via a PKI-like certificate chain based on the naming hierarchy or via a trusted friend that trusts these keys. Requests (*INTEREST* packets) for an NDO are forwarded towards a publisher location, as illustrated in Figure 3, steps 1–3.

A CCN router maintains a *Pending Interest Table (PIT)* for outstanding forwarded requests, which enables request aggregation, i.e., a CCN router would normally not forward a second request for a specific NDO when it has recently sent a request for that particular NDO. The PIT maintains state for all interests and maps them to a network interface where corresponding requests have been received from. Data is then routed back on the reverse request path using this state (steps 4–6). CCN can cache NDOs that a CCN router receives (in response to requests). Hence, subsequent received requests for the same object can be answered from that cache (as depicted in steps 7–8 in Figure 3). From a CCN node's perspective, there is balance of requests and responses, i.e., every single sent request is typically answered by one response. This enables flow control. CCN nodes can employ different *strategies* for request (re-)transmission pacing and interface selection, depending on local configuration, observed network performance and other factors. The NDN project advances the CCN approach. It provides a topology-independent naming scheme and is exploring greedy routing for better routing scalability.

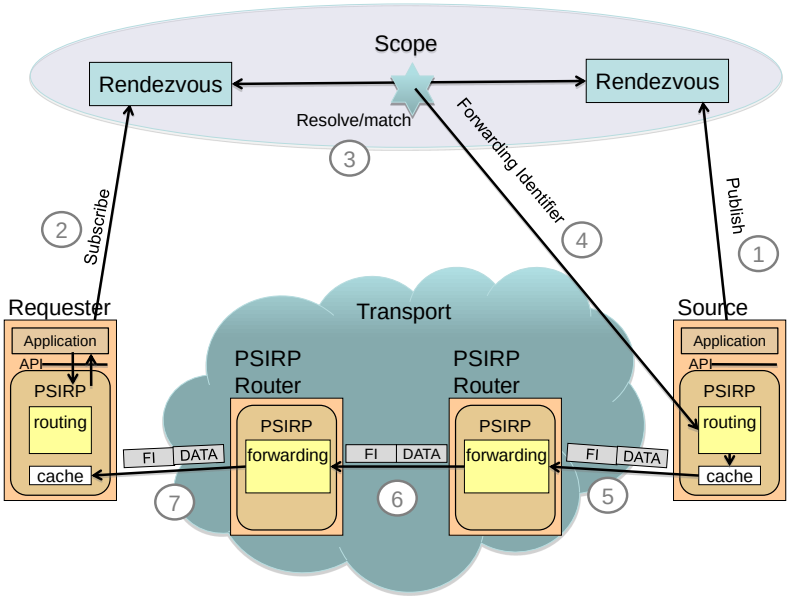

Figure 4: PSIRP overview

1.5.3 *Publish–Subscribe Internet Routing Paradigm*

In PSIRP, NDOs are also published into the network by the NDO sources. The publication belongs to a particular *named scope*. Receivers can *subscribe* to NDOs, as illustrated in Figure 4. The publications and subscriptions are matched by a *rendezvous system*. The subscription request specifies the Scope Identifier (SI) and the Rendezvous Identifier (RI) that together name the desired NDO. The identifiers are input to a matching procedure resulting in a *Forwarding Identifier* (FI) that is sent to the NDO source (publisher) so that it can start forwarding data. The FI consists of a Bloom filter which routers use for selecting the interfaces to forward an NDO on. This means that routers do not need to keep forwarding state. The use of Bloom filters results in a certain number of false positives, in this case this means forwarding on some interfaces where there are no receivers.

1.5.4 *Network of Information*

NetInf offers two models for retrieving NDOs: via *name resolution* and via *name-based routing (NBR)*, thereby allowing adaptation to different network environments. In NetInf, depending on the model used in the local network, sources publish NDOs by registering a name/locator binding with an NRS, or by announcing routing information in a routing protocol. A NetInf node holding a copy of an NDO (in-

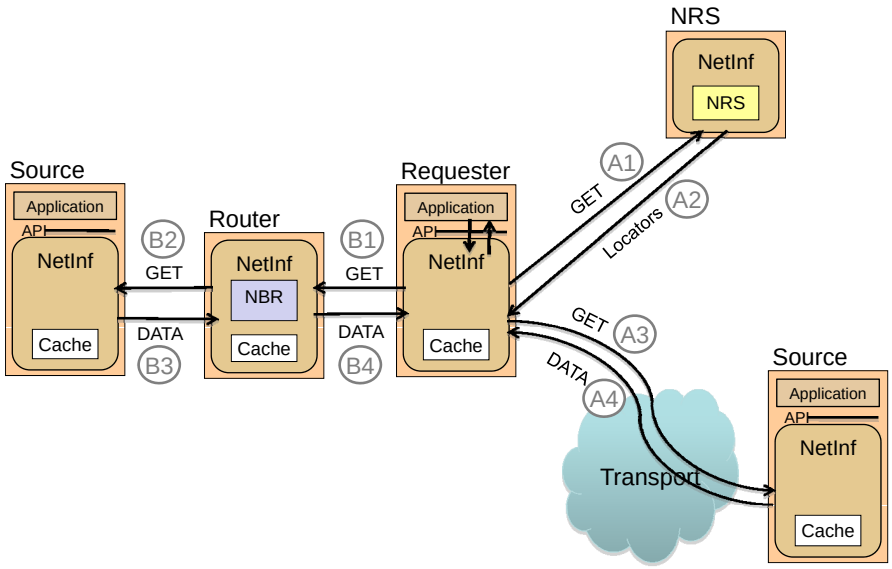

Figure 5: NetInf overview

cluding in-network caches and user terminals) can optionally register its copy with an NRS, thereby adding a new name/locator binding. If an NRS is available, a requester can first *resolve* an NDO name into a set of available locators and can subsequently retrieve a copy of the data from the "best" available source(s), as illustrated in steps A1–A4 of Figure 5. Alternatively, the requester can directly send out a *GET* request with the NDO name that will be forwarded towards an available NDO copy using name-based routing (steps B1–B4 of the figure). As soon as a copy is reached, the data will be returned to the requester. NetInf can also merge the two models in a hybrid resolution/routing approach where a resolution system provides mappings from NetInf object names to hints that support a name-based routing process in forwarding the request to available object copies.

1.6 THESIS CONTRIBUTIONS

This thesis presents the Network of Information (NetInf) architecture, an ICN-based architecture for the future Internet. NetInf has been developed as part of the European project 4WARD[4] and the development is continued in the European project SAIL[5]. I have been part of and have contributed to the overall NetInf development in both European projects. In the following, I briefly highlight the overall key NetInf features that I have contributed to and discuss the specific contributions of this thesis in more detail subsequently.

4 http://www.4ward-project.eu
5 http://www.sail-project.eu

Compared to other ICN proposals, NetInf offers the following combination of key features:

NAMING AND SECURITY MODEL The basic NetInf naming and security model offers a combination of *name–data integrity*, *name persistence*, and *owner pseudonymity*, all without requiring naming authorities or a PKI. Optionally, NetInf can also offer *owner identification*, which requires some kind of trusted third party.

OBJECT MODEL NetInf has a flexible, concrete object model which enables applications to construct powerful information models at the application level. This allows applications to offer efficient information retrieval also at higher semantic levels, e.g., by offering information retrieval while abstracting from specific information encodings. NetInf also offers a search primitive that provides a link from search terms to object names[6].

FLEXIBLE OBJECT RETRIEVAL NetInf offers a flexible object retrieval approach combining name resolution and name-based routing. NetInf can use either one separately in different parts of the network; it can even mix them into a hybrid scheme by switching between them on a hop-by-hop basis. By adjusting the object retrieval method, NetInf can adapt to environments with very different requirements like global connectivity, network mobility [20], and DTN. This creates interesting scaling properties – NetInf can be adapted to an ad hoc network of two directly connected user nodes as well as to large infrastructure networks with dedicated infrastructure nodes, e.g., for global name resolution.

CACHING The name resolution approach allows NetInf to not only rely on caching on the data path but also enables off-path caching and allows NetInf to benefit from any available data copy, including copies at user nodes.

DEPLOYMENT AND MIGRATION NetInf simplifies deployment and migration as it can run as an overlay on top of the existing network infrastructure during the migration phase. First, the most beneficial applications like media distribution could be migrated to NetInf, with other applications following later.

I now summarize the contributions of this thesis in more detail and list the publications in which these contributions were first published:

SURVEY OF INFORMATION-CENTRIC NETWORKING
Chapter 1 – Chapter 1 motivates ICN research in general, gives

6 The term *name* and *identifier* are used interchangeable in the rest of this document.

an overview of the main problems of the current Internet, introduces the ICN paradigm, and gives an overview of the main recent ICN approaches. Major parts of this content have first been published as surveys of information-centric networking [2, 4]. I am a main author of these surveys.

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine – Special Issue on Information-Centric Networking*, July 2012.

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking (draft)," in *Information-Centric Networking*, ser. Dagstuhl Seminar Proceedings, B. Ahlgren, H. Karl, D. Kutscher, B. Ohlman, S. Oueslati, and I. Solis, Eds., no. 10492. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany, 2011.

NETWORK OF INFORMATION ARCHITECTURE

Chapter 2 – In this chapter, I present the overall NetInf architecture. I have been a member of the NetInf architecture design team in the 4WARD and SAIL European Union (EU) projects and have especially been involved in the development of the naming scheme, the name resolution mechanisms, the object model, caching mechanisms, and the overall NetInf architecture.

The chapter starts with an overview of the main requirements that have motivated the NetInf design. I present an architecture overview and discuss the different architecture elements, including the object model, naming, message forwarding, caching, name resolution, and inter-domain communication. This chapter is mainly based on work published in reference [3], of which I am the first author. Related aspects have been published in references [21, 22, 23].

In related research, I have co-designed and led the development of an efficient search mechanism to support NetInf's search capabilities for complex queries (e.g., range queries and multi-attribute queries) in local, P2P-based scenarios. To keep this thesis focused on the main NetInf aspects, only a brief summary of these results is included here. Details about this work can be found in references [24, 25, 26].

[3] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (NetInf) – An information-centric networking architecture," *Computer Communications*, vol. 36, no. 7, pp. 721–735, April 2013.

[21] C. Dannewitz, "NetInf: An information-centric design for the future Internet," in *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, Munich, Germany, May 2009.

[22] B. Ahlgren, M. D'Ambrosio, C. Dannewitz, M. Marchisio, I. Marsh, B. Ohlman, K. Pentikousis, R. Rembarz, O. Strandberg, and V. Vercellone, "Design considerations for a network of information," in *Proc. Workshop on Re-Architecting the Internet (ReArch)*, Spain, December 2008.

[23] C. Dannewitz, K. Pentikousis, R. Rembarz, E. Renault, O. Strandberg, and J. Ubillos, "Scenarios and research issues for a network of information," in *Proc. 4th Int. Mobile Multimedia Communications Conference*, Oulu, Finland, July 2008.

[24] C. Dannewitz, T. Biermann, M. Dräxler, and H. Karl, "Complex queries in P2P networks with resource-constrained devices," *Journal of Advances in Information Technology (JAIT) – Special Issue on Advances in P2P Technology*, vol. 02, no. 01, pp. 02–14, January 2011.

[25] T. Biermann, C. Dannewitz, and H. Karl, "An adaptive resource/performance trade-off for resolving complex queries in P2P networks," in *Proc. IEEE International Conference on Communications (ICC)*, Dresden, Germany, June 2009.

[26] T. Biermann, C. Dannewitz, and H. Karl, "Extended results on an adaptive resource/performance trade-off for resolving complex queries in P2P networks," University of Paderborn, Paderborn, Germany, Tech. Rep. TR-RI-08-294, October 2008.

SECURE NAMING

Chapter 3 – The ICN paradigm makes use of any available copy, including copies from untrusted hosts. Hence, today's security model based on host authentication and secured channels is not applicable. Therefore, we have developed the NetInf naming scheme, which attaches the basic security functionality directly to the data and its naming scheme. I have acted as main designer and developer of the NetInf naming scheme.

In this chapter, I discuss the NetInf naming scheme and its security properties in more detail. The naming scheme enables name–data integrity verification, owner pseudonymity, owner identification, name persistence, and extensibility. The naming scheme builds the foundation for a secure NetInf architecture. This chapter is based on content first published by Dannewitz et al. [27].

Building on this general naming scheme, we have developed and published an RFC [28] that standardizes the details of the NetInf naming scheme, called the *named information (ni) URI scheme*. The ni URI scheme allows various forms of hash-based bindings between the name and the named object. It also supports a human-friendly format and a binary format.

Our naming scheme has generated significant interest and is currently under evaluation or in use by other IETF/IRTF research groups, e.g., the Decoupled Application Data Enroute (DECADE) group [29, 30], the Peer-to-Peer Streaming Protocol (PPSP) group [31, 32], and for the Constrained Application Protocol (CoAP) protocol [33] of the Constrained RESTful Environments (CoRE) group.

[27] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *Proc. 13th IEEE Global Internet Symposium 2010 (in conjunction with IEEE INFOCOM)*, San Diego, USA, March 2010.

[28] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker, "Naming Things with Hashes," RFC 6920 (Proposed Standard), Internet Engineering Task Force, Apr. 2013.

[32] C. Dannewitz, T. Rautio, O. Strandberg, and B. Ohlman, "Secure naming structure and p2p application interaction," IETF Internet-Draft draft-dannewitz-ppsp-secure-naming-02, March 2011.

[30] B. Ohlman, O. Strandberg, C. Dannewitz, A. Lindgren, R. Maglione, and B. Ahlgren, "Requirements for accessing data in network storage," IETF Internet Draft draft-ohlman-decade-add-use-cases-reqs-02, October 2010.

EVALUATION OF LOCALITY IN DNS REQUESTS

Chapter 4 – This chapter presents an evaluation of request patterns for data objects as a prerequisite for designing and evaluating the name resolution services presented in Chapter 5. I have led and performed this DNS evaluation and I am the main author of the associated publication [34] on which this chapter is based.

The evaluation focuses on the locality of requests for data objects, i.e., on the portion of requests that are posed for local content (network-wise), which we call *neighborhood effect*. To evaluate this effect, we have performed DNS measurements in two different DNS zones at the University of Paderborn for almost four months in total, comprising over 2.5 billion DNS requests. In this chapter, I evaluate the magnitude and characteristics of the neighborhood effect, the influence of requests originating from user devices and servers, and the subzone relationship between the two observed DNS zones. The results show a strong neighborhood effect with 71% (university-wide) and 40% (computer science department) requests for local hosts, respectively. As a consequence, I argue that this effect can have a significant impact on future Internet architectures in general and on information-centric networking in particular, especially for name-based routing, caching, and name resolution. The impact on NetInf's NRS framework is discussed in the subsequent Chapter 5.

[34] C. Dannewitz, H. Karl, and A. Yadav, "Report on locality in DNS requests – Evaluation and impact on future Internet architectures," University of Paderborn, Paderborn, Germany, Tech. Rep. TR-RI-12-323, July 2012.

HIERARCHICAL NAME RESOLUTION FRAMEWORK

Chapter 5 – Name resolution is very important for NetInf's object retrieval. It can be used in combination with name-based routing or on its own. Name resolution has the advantage that it can easily support on-path *and* off-path caching, can simplify mobility support, and does not require changes to the underlying routing and forwarding system. The last aspect is especially important to facilitate migration. During the migration phase, NetInf can run on top of today's IP network, which provides a fully connected underlying network graph; in this scenario, name resolution is sufficient to find and retrieve named data objects. Hence, only an NRS has to be deployed without requiring modifications to the underlying routing system, which simplifies deployment.

Building a world-wide NRS for NetInf's flat namespace with 10^{15} expected identifiers (IDs) is challenging because of requirements such as scalability, low latency, efficient network utilization, and anycast routing that selects the most suitable copies. In this chapter, I present a general *hierarchical* NRS framework for *flat* ID namespaces that meets these requirements.

The general NRS framework is flexible and supports different instantiations. These instantiations offer an important trade-off between subsystem autonomy (which simplifies deployment) and reduced latency, maintenance overhead, and memory requirements. To evaluate this trade-off and explore the design space, we have designed two specific instantiations of the general NRS framework: Multi-Level Distributed Hash Table (MDHT) and Hierarchical SkipNet (HSkip). I have been the main designer of the general naming framework and a co-designer of the MDHT and HSkip architecture. I have led and performed the theoretical analysis and the simulation-based evaluation of both systems. I am the main author of the associated publication that this chapter is based on [35] and a co-author of our related conference publication [36]. In addition, I have led the development of our MDHT prototype implementation that we have published as open source; this prototype is evaluated in Chapter 6. Results indicate that an average request latency of (well) below 100 ms is achievable in both systems for a global system with 12 million NRS nodes while meeting the other specified requirements. These results imply that a flat namespace can be adopted on a global scale, opening up several design alternatives for information-centric network architectures.

Load balancing between NRS nodes is another relevant aspect to address. Both MDHT and HSkip are based on structured P2P sys-

tems. As a result, their load balancing properties depend on the load balancing properties of the underlying P2P system. Today, most load balancing algorithms for structured P2P networks focus on range skew and data skew as sources of imbalances. However, the main potential load imbalance of a NetInf NRS system is *execution skew*, which refers to non-uniform data access across the peers' partitions. Therefore, as part of related research, I have co-developed a load balancing mechanism that addresses execution skew. To keep this thesis focused, Chapter 5 only contains a brief summary of these research results. Details can be found in references [37, 38].

[35] C. Dannewitz, M. D'Ambrosio, and V. Vercellone, "Hierarchical DHT-based name resolution for information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 736–749, April 2013.

[36] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A hierarchical name resolution service for information-centric networks," in *Proc. ACM SIGCOMM Workshop on Information-centric Networking*. New York, NY, USA: ACM, August 2011, pp. 7–12.

[37] D. Warneke and C. Dannewitz, "Load balancing in P2P networks: Using statistics to fight data and execution skew," *Journal of Advances in Information Technology (JAIT) – Special Issue on Advances in P2P Technology*, vol. 02, no. 01, pp. 40–49, 2011.

[38] D. Warneke and C. Dannewitz, "Statistics-based ID management for load balancing in structured P2P networks," in *Proc. 34th IEEE Conference on Local Computer Networks (LCN)*. Zürich, Switzerland: IEEE, October 2009.

PROTOTYPING – OPENNETINF

Chapter 6 – Prototyping the NetInf architecture is important to gain valuable insights about its feasibility. Based on a prototyping toolbox called Future Internet Toolbox (FIT), which I have co-developed earlier [39, 40, 41], we have developed a prototype of the NetInf architecture called *OpenNetInf* and have tested the architecture with a wide variety of ICN applications [42, 43]. I have been the main architect and team leader of the prototype development and am the main author of the publications that this chapter is mainly based on [44, 45]. As part of this work, we have also performed traffic measurements to evaluate the influence of caching and the MDHT NRS on inter-domain traffic. The measurements show a decrease in inter-domain traffic by a factor of up to 4 in the test scenario. The prototyping experience has validated the general feasibility of the NetInf architecture. The gained insights have had significant impact on future NetInf ar-

chitecture iterations. OpenNetInf is also published as open source; details are available at `http://www.netinf.org`.

[39] C. Dannewitz, T. Biermann, M. Dräxler, F. Beister, and H. Karl, "Prototyping with the Future Internet Toolbox," in *Proc. 6th Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, May 2010.

[40] T. Biermann, C. Dannewitz, and H. Karl, "FIT: Future Internet Toolbox," in *Proc. 6th Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, May 2010.

[41] T. Biermann, C. Dannewitz, and H. Karl, "FIT: Future Internet Toolbox — extended report," University of Paderborn, Paderborn, Germany, Tech. Rep. TR-RI-10-311, Feb. 2010.

[42] C. Dannewitz, "Augmented Internet: An information-centric approach for real-world/Internet integration," in *Proc. Int. Workshop on the Network of the Future (in conjunction with IEEE ICC)*, June 2009.

[43] C. Dannewitz, H. Karl, and D. Warneke, "Service platform for real-world/Internet integration in mobile applications," in *Proc. 13. Mobilfunktagung*, Osnabrück, Germany, May 2008.

[44] C. Dannewitz, M. Herlich, and H. Karl, "OpenNetInf – Prototyping an information-centric network architecture," in *Proc. IEEE LCN – Workshop on Architectures, Services and Applications for the Next Generation Internet (WASA-NGI)*, October 2012.

[45] C. Dannewitz and T. Biermann, "Prototyping a network of information," in *Demonstrations – IEEE Local Computer Networks (LCN)*, Zurich, Switzerland, 2009.

[143] M. Soellner, P. Schefczik, P. Bertin, G. Wei, X. Zhang, T.-M.-T. Nguyen, J. Mäkelä, T. Rautio, O. Mämmelä, S. Pérez, A. Eriksson, A.-M. Biraghi, C. Foley, M. P. de Leon, C. Dannewitz, T. Biermann, and M. Marchisio, "Mobility in the future Internet: the 4WARD innovations," 2nd Future Internet Cluster Workshop, June 2010.

CONCLUSION AND FUTURE WORK Chapter 7 – In this chapter, I discuss some conclusions of my work. Specifically, I discuss deployment considerations for ICN architectures in general. What has to happen to make ICN in general a success? Subsequently, I give a brief summary and consider implications of the NetInf architecture. This thesis concludes by discussing some future work items and interesting open questions. This chapter partly contains a combination of text previously published in references [2, 3, 4] listed above.

2 NETWORK OF INFORMATION ARCHITECTURE

This chapter is based on work published in references [3, 21, 22, 23, 24, 25, 26].

This chapter gives an overview of the overall Network of Information (NetInf) architecture. To keep this thesis focused, not all aspects can be discussed in detail. Rather, the aim of this chapter is to get a detailed understanding of the overall architecture and the interaction of the different elements. Details of selected aspects (secure naming and name resolution) will be discussed in more detail in the following chapters.

Section 2.1 provides an overview of NetInf's design principles and the NetInf architecture. The design principles are chosen based on the requirements discussed in Section 1.4. Section 2.2 discusses details about the architecture elements, protocol stack, node architectures, and the network architecture. Section 2.3 compares the NetInf architecture to related work and Section 2.4 summarizes this chapter.

2.1 OVERVIEW

NetInf is a *networking approach* that provides *access to NDOs* as a first-order networking primitive, i.e., the primary service of nodes in the Network of Information is the forwarding of NDO requests and the transfer of the corresponding objects (or object location information). In the following, we first describe the main NetInf design principles and then provide an architecture overview.

2.1.1 *Design Principles*

ACCESSING NAMED DATA OBJECTS Accessing named data as a first-order principle implies that the nodes in a NetInf network do generally not communicate on the basis of network or host addresses, but instead use NDO names to identify objects independently of network location. Unique, location-independent naming enables ubiquitous replication and caching of NDOs in the network. An NDO consists of its name in a common format and the actual object in a common data structure (see below). NetInf's main service is to forward NDO requests to appropriate copies and transfer objects back.

MINIMAL COMMON NODE REQUIREMENTS To broadly apply to different types of networks and deployments, NetInf only makes minimal node requirements:

- *Naming format*: There is one common NDO *naming* format that all nodes *must* understand.
- *Object model*: There is one format for *representing* NDOs and optional metadata. All nodes can process this format. The format allows, e.g., application-specific extensions; not all nodes need to understand all extensions or metadata formats.
- *NetInf protocol*: There is one simple protocol (called the *NetInf protocol*) that all nodes implement. The NetInf protocol is message-based; it provides requests and responses for `PUBLISH`ing, `GET`ting, and `SEARCH`ing for NDOs. These requests employ the common naming format and the common object model.

GENERIC NETINF NODES We call the nodes actively participating in a Network of Information *NetInf nodes*. NetInf nodes can provide different functions such as forwarding requests and responses, caching, and name resolution. Conceptually, all NetInf nodes are equivalent and can offer all these functions. In practice, NetInf consists of specialized *infrastructure nodes* and *user nodes*. Infrastructure nodes typically offer only a small subset of the NetInf services, yet on a large/global scale, e.g., services like caching inside the network infrastructure (*NetInf cache*), large-scale name resolution (*NetInf NRS*), or inter-domain request and data forwarding (*NetInf router*). User nodes, in contrast, typically offer some of the corresponding services with a local focus, e.g., local caching, name resolution for locally known objects, and local request forwarding. The generic NetInf node approach makes the NetInf benefits available in many different network contexts such as an *infrastructure-less* ad hoc network and an *infrastructure-based* global Internet.

FLAT NAMESPACE NetInf uses a flat namespace for NDO names, i.e., there is neither topology- nor organization-related hierarchy in names. On the one hand, this limits the ability to aggregate names based on such a hierarchy. Aggregation is a means to reduce the size of routing tables and name resolution tables, which can be important to improve the scalability of the architecture. However, methods like *explicit aggregation* [7] can be used instead of a hierarchical namespace to enable aggregation. In explicit aggregation, aggregation occurs on the naming side (and not on the router side) by concatenating flat names, e.g., A.B.C.D, where each part is a flat name. Forwarding works by looking for an exact match of each name part in the routing table (going from more specific name parts to less specific name parts) until a match is found. Among others, explicit aggregation is more

flexible than strict hierarchical aggregation and promises to be easier to implement. Details can be found in reference [7].

On the other hand, for many hierarchical namespaces, the hierarchy is based on aspects like organizational structures, folder structures, etc. Compared to such hierarchical namespaces, a flat namespace provides better name persistence as it eliminates such interdependencies. Otherwise, e.g., organizational changes would result in name changes. In addition, a flat namespace also has the advantage of separating tussle over trademarks from unique data naming [46].

With properly designed distributed algorithms for name construction, name management is significantly simplified even without a naming authority to assign names. Naming can rely on statistical uniqueness; rare name collisions can be handled as error that will be handled by an error handling mechanism. For example, name collisions can be detected by the NRS during object registration. Details are out of scope here.

NAME–DATA INTEGRITY Name–data integrity validation is NetInf's fundamental object security service for NDOs. This includes both *static* and *dynamic* objects, i.e., objects with changing content. The common naming format and object model enable data-integrity validation by requesters (or any other node). Name–data integrity validation can be performed without infrastructure support like a PKI (employing message digests and/or public key digests as part of NDO names). Not all nodes are required to perform the validation.

NetInf provides additional object security services such as *owner pseudonymity* and *owner identification*, employing public-key cryptography.

LOWER-LAYER INDEPENDENCE The NetInf protocol specifies messages for node-to-node communication, their semantics, and corresponding node requirements. Different deployments will use different link layers and underlays – with a variation of services and properties. NetInf accommodates this by *convergence layers (CLs)* (using the almost identical term from the DTN architecture [47]; see reference [19] for differences between DTN and NetInf convergence layers) that map the conceptual protocol to specific messages, transactions, or packet exchanges in an existing, concrete protocol. A CL provides framing and message integrity for NetInf requests and responses for communication between two nodes as its main service, but specific CLs can provide additional services. In general, CLs can be connection-oriented or not, uni- or multicast, may encapsulate, fragment and reassemble, or even reformat higher-layer protocol data units (PDUs), etc.

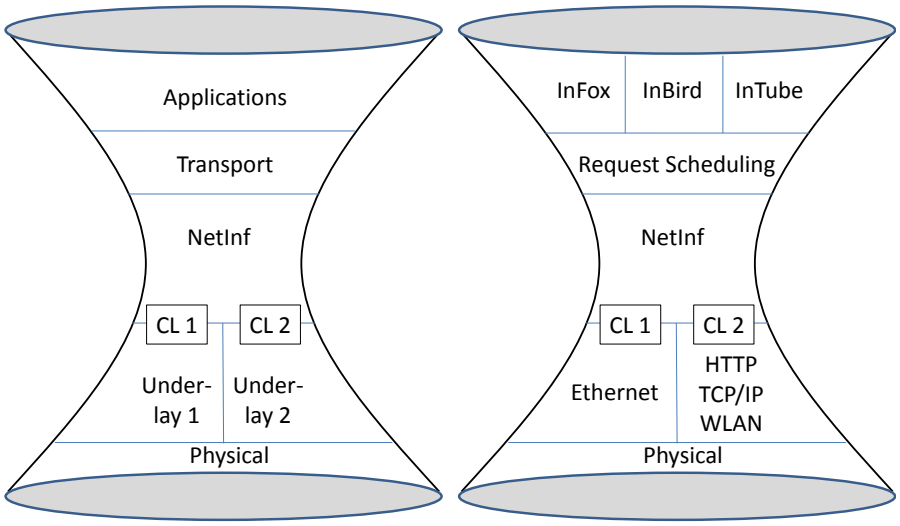

Figure 6: NetInf protocol stack, assuming a node with two convergence layers over two different underlays

Figure 6 depicts a CL example. NetInf-over-IP would require a CL that encapsulates, and potentially fragments and reassembles, NetInf messages for transfer in IP packets and validates message integrity, e.g., via cyclic redundancy check (CRC). NetInf-over-Ethernet, on the other hand, could be done with a slim CL, but that restricts choices within NetInf (see below). Figure 6 depicts a schematic and sample NetInf stack for different CLs/underlays; it also shows that between applications and NetInf, additional functions like request scheduling (for implementing flow control, congestion control and other transport layer functions) can be inserted. With CLs, NetInf runs over quite different types of network links, including uni-directional or heavily delay-challenged links.

While specific CLs *can* provide transport protocol functions (reliability, flow control, congestion control) on a hop-by-hop basis between NetInf nodes, there is also a need for transport layer functions across CL links. In NetInf, such functions are implemented on-top of the NetInf layer, e.g., by request scheduling/retransmission decisions.

ROUTING, FORWARDING, AND NAME RESOLUTION The NetInf protocol's request and response forwarding service requires routing information to decide to which next hop, over which interface a request should be forwarded. For `GET` requests, these decisions are generally based on the name of the requested NDO.

NetInf supports both name-based routing and name resolution. Name resolution plays a major role in NetInf, e.g., for NetInf's scal-

ability and deployability. In responding to a NetInf `GET` request, any node may either return the requested object or *hints* that can help the requester to access the object. We call these hints *routing hints*.

Different types of routing hints are supported. Routing hints can be:

1. another NetInf name that enables indirection,
2. lower-layer host locators that can be directly used to retrieve the object via the underlying network protocol (e.g., Transmission Control Protocol (TCP)/IP),
3. protocol-specific routing hints that can subsequently be used to support name-based routing (e.g., by pointing to the next network that can provide the object).

Alternative (3) supports NetInf's *hybrid* combination of name resolution and name-based routing.

Not all combinations of underlays, convergence layers, and forwarding mechanisms make sense, however. In particular, an underlay that does not provide a (virtually) fully connected graph combined with a functionally slim CL cannot rely solely on name resolution, i.e., alternative (2). Alternative (2) does not work here as the end-point locator that the NRS would return might have no meaning in the local network. Hence, some form of name-based routing or hybrid mode is required here.

ON-PATH AND OFF-PATH CACHING NetInf supports both *on-path* caching as well as *off-path* caching. On-path caching refers to caching data directly on the network path that the data request and/or the data itself take between the requester and a data source, e.g., caching directly in the involved network routers. In contrast, off-path caching refers to caching data off the path, i.e., aloof the path that the data request and the data take.

NetInf can make use of any available copy when retrieving data: the original server, copies from redundant servers, as well as replicas stored on user devices (if so permitted). Thereby, NetInf can access the best available copy.

HETEROGENEOUS NETWORKS NetInf's flexibility in convergence layers and routing/forwarding schemes allows custom-tailored configurations for different technologies and different administrative domains. As long as NetInf's minimal node requirements are maintained, the NetInf protocol can be mapped to quite different types of network links – from bi-directional message/packet-based point-to-point links to unidirectional broadcast links and intermittently connected links. NetInf also does not assume anything about specific network system architectures – e.g., terminals, access networks, core

networks are flexible notions and there is no assumption on *where a network ends*: NetInf user devices can appear as terminals, but they can also provide caching and can forward requests and data, thus, extending the network or connecting NetInf networks.

INHERENT MOBILITY AND MULTI-HOMING SUPPORT NetInf inherently makes mobility and multi-homing simpler. Regarding multi-homing, the NetInf architecture puts no limits on the number of interfaces used in parallel to send requests and to receive responses. As all object copies are equal, the order of requests and responses is not critical. To use network resources in an optimized way, several request strategies can be used in NetInf. In some network scenarios, e.g., in a local setup, broadcasting requests on all interfaces might be optimal. In some other network scenarios, an NRS-based solution using only a single interface might be preferred. A strategy component inside a NetInf node can collect network information and related information, e.g., about previous requests, to choose the best strategy.

There are different types of mobility. For client mobility (i.e., moving requester), there is no need to continue using a specific object copy. Instead, alternative copies close to the client's new location can be used. For server mobility (i.e., moving content), updates of the routing information and/or name resolution information in the network are required to ensure that the server's copy stays accessible.

For traditional point-to-point services (e.g., a voice call) the need for mobility support will be very different in different parts of the network. As NetInf can support different name resolution mechanisms in different parts of the network, these can be selected taking the dynamics of respective network parts into account. While traditional mobility anchor point mechanisms (like in mobile IP [10]) can be used for some parts, very dynamic environments where whole networks are moving (e.g. trains) can rely on mechanisms like Late Locator Construction (LLC) [20] that constructs global locators on demand to support highly dynamic network topologies.

2.1.2 *Architecture Overview and Sample Setup*

SAMPLE MESSAGE FLOW Figure 7 shows an example of name resolution, name-based routing, and the hybrid approach in NetInf. The name-based routing (steps A1–A4) forwards a `GET` request hop-by-hop between NetInf nodes until a cached copy of the NDO is found or the original server is reached. If the router does not have enough routing information to perform name-based routing (NBR) in step A2, it can perform a name resolution step (steps A1.1–A1.2) before forwarding the request (step A2) based on the retrieved routing hints,

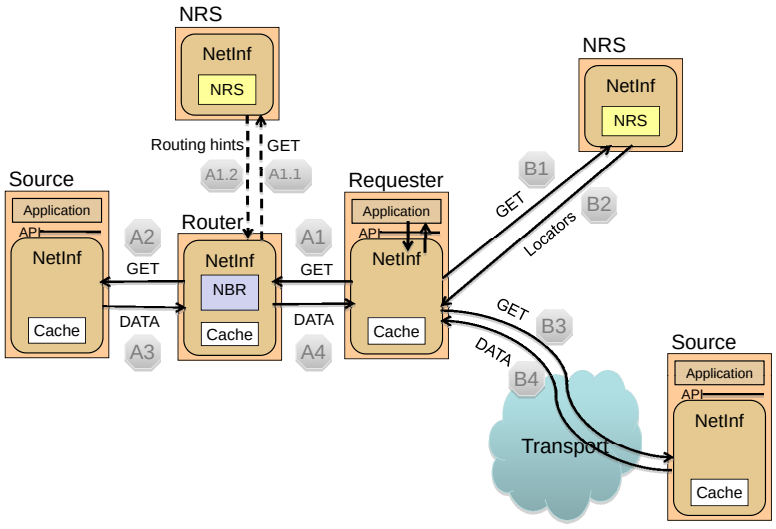

Figure 7: NetInf example message flow

which illustrates the hybrid mode. On the return path, the object can be cached in intermediate nodes for subsequent requests.

Alternatively to the name-based routing approach, the initial requester can first perform name resolution by querying an NRS (steps B1–B2) via a `GET` message to resolve the object name into a set of *routing hints*, in the example simply lower-layer host locators. Subsequently, the routing hints are used to retrieve the object via the underlying transport network (steps B3–B4), e.g., a legacy IPv4 network.

SAMPLE NETWORK SETUP Figure 8 shows an example of a larger Network of Information, which typically consists of multiple interconnected domains called *NetInf domains*. A NetInf domain is an administrative domain like in today's Internet that is running its own internal Network of Information. Due to NetInf's flexible architecture, a domain has some freedom how to configure the internal NetInf setup, e.g., choosing an appropriate NRS or name-based routing approach, on-path or off-path caching, etc.

As a simple case, NetInf nodes can be interconnected in an infrastructure-less ad hoc way, i.e., NetInf services like name resolution and request/data forwarding are provided by the user nodes (Figure 8, left side). Likewise, NetInf nodes can be interconnected in a (local) infrastructure-based network (Figure 8, right side) with dedicated NetInf infrastructure nodes such as a dedicated NRS and NetInf routers, possibly specialized for the idiosyncrasies of the local network (e.g., DTN, network mobility). Such local networks can easily be connected to the global Network of Information (Figure 8, middle).

This requires updating the local NetInf router with routing information or connecting the local NRS node to a global NRS (publishing any locally registered information at the global NRS if desired).

Such setups are based on interconnecting multiple coexisting NRSes. We expect that network providers typically run their own NRS as this allows them to better control the network-internal traffic flow and reduce inter-domain traffic. Each provider can set up a local hierarchy of NRS nodes that matches its network topology. Providers will typically also add NetInf-enabled (on-path and off-path) in-network caches to their networks. Off-path caches are typically connected to NRS nodes to retrieve information about local object popularity and to register cached objects. In addition to in-network caches, NetInf can continue to access today's content servers (e.g., web servers) as shown in Figure 8.

Figure 8 shows a global, hierarchical, topologically-embedded NRS which interconnects smaller, local NRSes. This global NRS consists of all/most other NRS nodes. We estimate (Section 5.6) that such a global NRS would require about 10^6 NRS nodes to handle 10^{15} objects globally.

In an infrastructure-based Network of Information, user nodes that request objects have two alternative options. They can either send the object `GET` request to a known NRS or use name-based routing to find the next hop NetInf router. Both the NRS and the adjacent routers are typically preconfigured at the user node like in today's networks, e.g., via the Dynamic Host Configuration Protocol (DHCP) or manually. NetInf routers are placed inside provider networks to forward requests and data internally as well as between domains to perform inter-domain routing/forwarding. As a result of NetInf's hybrid object retrieval approach, NetInf routers can also contact NRS nodes to resolve NetInf names into a set of routing hints.

2.2 ELEMENTS OF THE NETWORK OF INFORMATION

The following sections provide a more in-depth discussion of the main elements of the NetInf architecture, including NDOs, naming, security, NetInf communication including convergence layers, the NetInf protocol, routing and forwarding, caching, name resolution, name-based routing, and inter-domain communication.

2.2.1 *Named Data Objects*

Applying NetInf on a large scale will require agreement on how to structure NDOs. For example, some name–data integrity validation approaches require additional cryptographic material (signatures, keys, certificates) to be attached to the actual object that is dis-

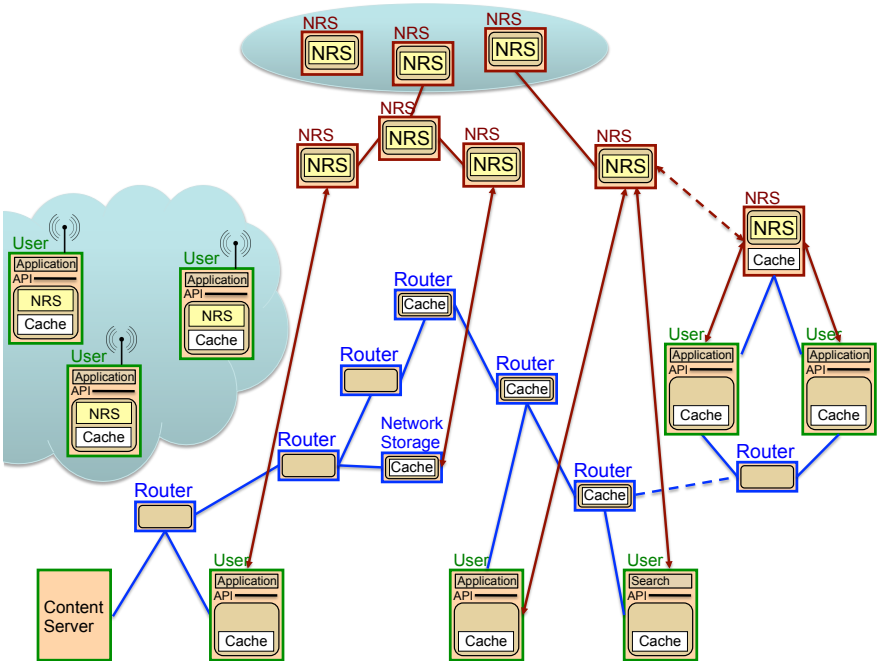

Figure 8: NetInf sample network setup

tributed and stored in a network. Moreover, objects may have some sub-structure, including object fragments, application-specific metadata, etc.

A well defined object structure as depicted in Figure 9 and a common naming format (see below) are cornerstones of NetInf – independent of how NDOs are transmitted and received, the NDO (i.e., its name and data structure) are sufficient for NetInf nodes to forward or otherwise process the NDO. The existence of application-specific extensions (specific data types, metadata, etc.) is explicitly signaled, and the common structure enables access to such extensions.

As a concrete object model format, we currently favor Multipurpose Internet Mail Extensions (MIME), leveraging multi-part messages, message (content) type identification, and security services, i.e., Cryptographic Message Syntax (CMS). Based on our previous experience, we favor MIME mainly due to its wide-spread use and acceptance. Alternatives that we have experimented with include the more powerful but also more complex Resource Description Framework (RDF) and Web Ontology Language (OWL) (which is used in the OpenNetInf prototype described in Chapter 6), or some self-defined Extensible Markup Language (XML)-based scheme, which would, however, require additional standardization.

Based on the simple and flexible model shown in Figure 9, applications can construct complex information models, for example, representing relations between NDOs. NDOs can link to each other by including references to other NDOs in the *application-specific metadata*,

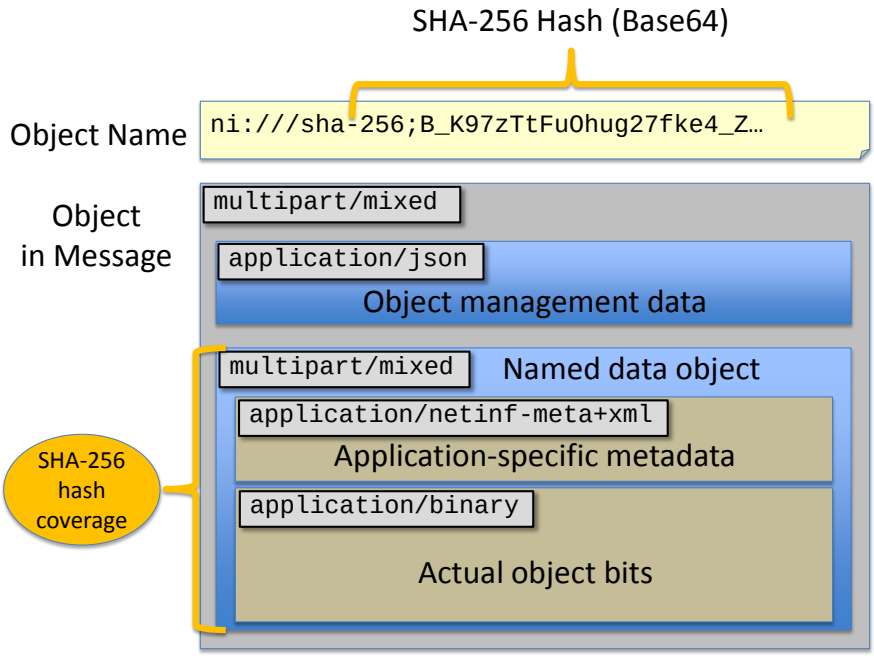

Figure 9: Conceptual Object Model

thereby creating NDOs that, e.g., represent some content independent of its specific encoding. A generic NDO representing a specific movie (independent of its encoding) could link to other, more specific NDOs that each represent a video file that contains the represented movie in different encodings. Based on such an information model, users could request an encoding-independent video NDO by name, and the application would retrieve required metadata from this encoding-independent NDO and could subsequently retrieve the best available copy of the movie among the different encodings based on supported encodings, available bandwidth, etc. A concrete demo implementation of this example is shown in Section 6.3.1. Likewise, an application could create NDOs that represent real-world objects (e.g., a person or a building). We describe this in our Augmented Internet (AugNet) use case in Section 6.3.4.2. Note that these complex information models are application-specific for now, i.e., the NetInf infrastructure does not understand these semantic concepts. However, in the future, more intelligent decisions could also be made *within* NetInf based on this semantic information, e.g., by an NRS.

2.2.2 *Basic Naming and Security*

The following two sections give a brief overview of NetInf's basic and advanced security mechanisms. All mechanisms are described in detail in Chapter 3.

NetInf names serve three different purposes. First, names uniquely identify NDOs. Second, NetInf names contain security-related infor-

mation to enable name–data integrity and other advanced security features. Third, names act as keys (in the database sense) for name resolution and routing mechanisms. This section gives an overview of the basic NetInf naming and security concept, while the following section gives an overview of additional advanced security features of this naming approach. The following overview is based on the general URI scheme for named information (the *ni URI scheme* [28]), which we have developed based on the security features as described in more detail by the general *NetInf naming scheme* in Chapter 3.

In the ni scheme, a basic NetInf name defines the utilized hash algorithm and contains the corresponding hash value in the form `ni:///digest-algorithm/digest-value`[1]. A concrete example would be: `ni:///sha-256;f4Ox...JtkGk`. The ni URI format enables NetInf nodes to optionally verify *name–data integrity*, which is based on the idea of verifying that the received data corresponds to the requested name. In its simplest form, this is achieved if the hash value in an ni URI is calculated over the NDO (i.e., the NDO data and any specified metadata) that will be delivered by the NetInf protocol. This scheme can be applied to any *static* data, i.e., data that does not change during its life time. *Dynamic* data that might change during its life time would destroy name persistence. Changing the data would result in a changed hash value and, hence, the name itself would change. Therefore, we use a separate mechanism for dynamic data as explained in Section 2.2.3.

The major benefit of this simple scheme for static data is that it requires no key management since there are no cryptographic public/private key pairs needed! Hence, no PKI is required for name–data integrity of static data, eliminating requiring trust in a third party, a potential extra round-trip time to contact the PKI, any hassle with revocation lists, etc. Yet, basic name–data integrity provides enough security to enable the main ICN benefits (location independence).

2.2.3 *Advanced Naming and Security*

As described so far, the ni URI scheme only provides name–data integrity for *static data*. To do so for *dynamically changing data*, we include a hash of a public key in the ni URI rather than a hash of the NDO. Thus, anything signed by the holder of the corresponding private key can be verified by NetInf nodes. In addition, this use of signatures provides a concept of "ownership" and a means to manage owner pseudonyms.

Owner pseudonymity can express that multiple NDOs belong to the same owner/creator. An owner can build up trust in this pseudonym,

1 The three slashes are due to the fact that this simplified example skips the optional *authority* element, which is out of scope here. Details can be found in our ni naming RFC [28].

allowing users to trust in subsequent content published under it. At the same time, owner pseudonymity allows NDO owners to remain completely anonymous by supporting an unlimited number of pseudonyms per user.[2]

Owner identification can bind the pseudonym to the owner's real-world identity. This optional feature can be done via the use of standard PKI mechanisms, e.g., a Trusted Third Party or a Web of Trust. They are both, however, external systems and not part of the core NetInf architecture.

2.2.4 *NetInf Communication*

NetInf nodes use the NetInf protocol, forwarding messages over some lower-layer technology. This may be TCP/IP-based. In the future, it may also be based on non-IP networking technologies, e.g., being realized directly on top of point-to-point links without end-to-end networking capabilities. Semantic gaps to lower layers are bridged by convergence layers.

2.2.4.1 *Convergence Layer Approach*

NetInf nodes communicate using convergence layers (CLs) (Figure 10). By abstracting the NetInf protocol from lower layers, CLs ensure that NetInf implementations can be deployed across technologies.

A convergence layer communication is always hop-by-hop between NetInf nodes, i.e., there is no communication over multiple NetInf hops *inside* any convergence layer. A convergence layer does not make use of any NetInf layer node identifiers.

Depending on deployment details, one or more CLs may be used in the overall network. There may be one common CL (that most nodes are expected to support) and a set of less common ones – for instance, for specific access network types.

A NetInf CL does not treat a message PDU as opaque data. It understands structure and nature of the message fields to improve efficiency but it never alters the content. The CL architecture provides marshalling higher "layer" PDUs, exactly reproducing that PDU on the other side of the CL hop; it does so efficiently given the used CL protocol, maintaining message integrity, name–data binding, and other security properties.

2 In a deployed system, incentives (e.g., associated costs) might be required to prevent users from massively "waisting" pseudonyms to remain a high likelihood of statistical name uniqueness.

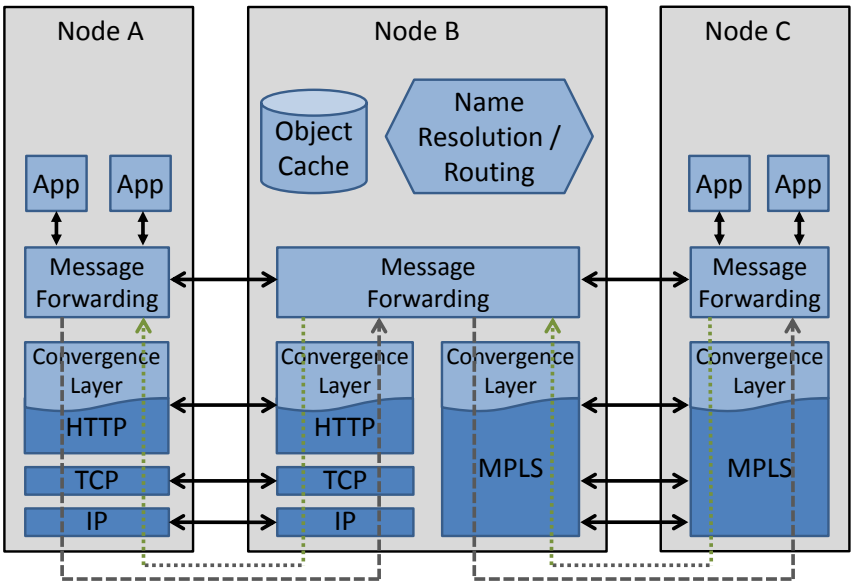

Figure 10: Example NetInf convergence layers

2.2.4.2 *Conceptual Protocol*

A specific CL implements the conceptual NetInf protocol that provides the following fundamental messages and responses:

GET/GET-RESP The `GET` message requests an NDO from the NetInf network. Each `GET` message has a unique message ID for identification purposes. A node responds to the `GET` message if it has a copy of the requested NDO; it sends a `GET-RESP` that uses the `GET` message's ID as its own identifier to link those two messages with each other.

PUBLISH/PUBLISH-RESP The `PUBLISH` message makes a new name known (e.g., to an NRS or NetInf router) and, optionally, it allows to push a copy of the object data and/or object metadata (e.g., to a cache). The `PUBLISH-RESP` message is used to acknowledge the `PUBLISH` message and inform the sender about the status of the request, e.g., `successful` or `declined` (including some additional error information).

SEARCH/SEARCH-RESP The `SEARCH` message contains search keywords. The response is either a status code or a multipart MIME object containing a set of metadata body parts, each of which *must* include a name for an NDO that is considered to match the query keywords.

The details of the conceptual protocol are out of scope here. They are described in reference [19].

2.2.4.3 *Routing, Forwarding, Transport*

NetInf forwards and resolves requests for objects as well as forwards the response messages. Different parts of the network can have different routing requirements and, thus, will need different routing protocols, just as we have multiple routing protocols for IP today. Hence, NetInf supports different request/response routing/forwarding mechanisms, such as Open Shortest Path First (OSPF) for local domains.

REQUEST FORWARDING NetInf uses a hybrid request routing/forwarding scheme, integrating pure name-based routing (in the sense of "routing on flat names") with name-resolution-based aspects. The name-based routing could, e.g., simply match names with default routes via prefix or pattern matching. For example, intra-domain request forwarding can rely on name-based routing mechanisms. At the inter-domain edges, name resolution can be used to support the request forwarding process, i.e., the inter-domain forwarding can be supported by or can rely solely on name resolution. The integration of name resolution and name-based routing also supports *late binding*: In heterogeneous domains or in a NetInf DTN island, it is often not possible to resolve the object name *at the requester side* into a locator that *has meaning* at the requester side. Instead, the name might be resolved into some routing hint that leads only *towards* the final destination. In this case, the resolution into the final destination locator can be performed by an intermediate NetInf node along the path to the destination. This combination with name resolution enhances scalability and performance of name-based routing.

Routers performing name-based routing (or default route forwarding) can be configured to do request aggregation (similar to CCN's pending interest management).

For global connectivity, we suggest a routing infrastructure similar to Border Gateway Protocol (BGP), combined with a global NRS infrastructure, i.e., a hybrid approach as described above. Such a scheme adapted to the convergence layer protocol approach is described in Section 2.2.7.

RESPONSE FORWARDING Responses need to get back to requesters. As with request routing, NetInf requires flexible response routing and forwarding and an implementation should make this pluggable like the request routing.

For response forwarding, the main issue is how to handle the routing state required to make sure the response gets back to the right place when the request has spanned more than one CL "hop." The maintenance of in-network state for a single CL "hop" is something that is handled by the specific CL, e.g., if a TCP CL is used, the response must be generally returned on the same socket from which the

request was received. In the general case where a request has passed over many CL "hops," the issue boils down to how to associate a response coming from "downstream" (towards the requester/"source") with a request previously received from "upstream" (from the requester).

The NetInf protocol allows different approaches depending on the deployment scenario and scalability constraints. Options are to maintain state in the routers or to annotate the request with some form of token or label. Labels can be locally significant identifiers and label stacks can be used to record a request route for returning responses along a path of NetInf nodes. Note that name-based routing is not the focus of this thesis. Hence, details are out of scope here and are in many aspects still subject to further studies. However, Section 2.2.7 provides an example of a system using a label stack approach.

TRANSPORT SERVICES Since not all underlays have their own flow control/congestion control or reliability, there are two options: (1) A NetInf CL implements this functionality. (2) If the CL only provides a basic, integrity-protected message delivery service, transport functions are needed above the CL. For case (2), NetInf employs a receiver-driven transport above the NetInf protocol layer as outlined in reference [19].

2.2.5 *Caching*

Caching is critical for NetInf as it lays the basis for efficient content dissemination, load balancing, and operation in challenging network conditions. NetInf supports three different kinds of caching that address different scenarios. First, a NetInf router with built-in caching functionality can provide *on-path caching*. While forwarding `GET` responses, the router can cache the contained objects. Subsequently, it can answer requests for the same object directly from its cache. Second, NetInf also supports *off-path caching*, i.e., a cache that is not directly on the request/data path. An off-path cache can be a dedicated cache that is typically placed in the network by the provider to reduce traffic (especially costly inter-domain traffic) and to reduce latency. Off-path caching can also be performed by a router. As the off-path cache is not directly on the request/data forwarding path, additional steps have to be taken to know which objects to cache and to advertise cached copies. Typically, an off-path cache is closely connected to one or more NetInf NRS nodes. The NRS tells the off-path cache which objects to cache based on object popularity and the cache advertises cached copies in the NRS in return. When the NRS receives a `GET` request for an object cached at a connected off-path cache, the NRS can return the cache's locator as routing hint to the requester, which can subsequently retrieve the object from the cache. In addition, the NRS

can optimize locator selection by either filtering or sorting available locators based on additional network knowledge and server/cache load information. Thereby, NetInf can offer functionality similar to approaches that utilize network knowledge to improve peer selection in P2P networks (e.g., ALTO [48], P4P [49]). However, this functionality is tightly integrated in the NetInf architecture and does not require supplemental infrastructure or protocols like other approaches such as Application-Layer Traffic Optimization (ALTO) and provider portal for P2P (P4P) do. Similarly, multiple NetInf routers can be closely connected to an off-path cache. Thereby, multiple routers can share a joined cache and only have to store an index of the cached objects' names instead of the objects themselves.

Finally, NetInf also supports *peer-side caching* in NetInf nodes on user devices, which is architecturally very similar to off-path/on-path caching (as NetInf user nodes can also serve as routers) but has some additional practical implications. Peer-side caches serve two purposes: First, it serves as application-independent cache for all local NetInf applications. Unlike today's commonly application-specific caches, the application-independent NetInf peer-side cache can serve all NetInf applications, thereby increasing efficiency. This effect becomes even more useful when multiple terminals share a common *initial NetInf node*, i.e., the first node running a NetInf process that an application connects to. While each user node typically runs a local NetInf node, some efficiency considerations or performance restrictions might result in multiple user nodes sharing a single initial NetInf node, which might be installed, e.g., on the local Digital Subscriber Line (DSL) modem or a dedicated NetInf server. In this case, all user nodes and their applications share the peer-side cache of the initial NetInf node.

Second, the peer-side cache can serve as cache for other NetInf nodes. Each user NetInf node can choose if it wants to make its copies available to other NetInf nodes based on considerations such as shared resources (e.g., bandwidth and memory). NetInf can provide users with incentives (e.g., download limits depending on shared resources) to share their resources as done in P2P networks today.

Peer caches can function as on- or off-path caches or a combination, i.e., a peer cache can advertise its cached copies in a local NRS and can also respond to `GET` requests by peer NetInf nodes (received, e.g., via local broadcast) with a cached copy. When registering a cached copy with an NRS, the NetInf node can choose up to which network level the cached copy should be available/visible. For example, a NetInf node could decide to make its copies available to all other NetInf nodes in the same local network, but not beyond.

Peer caching is especially beneficial in challenging network conditions where outside connectivity is limited. In addition, it can help to reduce the inter-domain traffic and overall network traffic

as also demonstrated in Section 6.4. It reduces traffic for the original source as requests can potentially already be answered locally, supports rapid load distribution for popular content, reduces the flash-crowd effect, and can reduce the latency as content can be served from close-by caches.

2.2.6 *Name Resolution*

Name resolution and name-based routing are essential complements and alternatives in NetInf. Especially during migration, name resolution has several advantages. It does not require global adoption right from the start but can grow from the edges by first providing NRS capabilities in some edge networks that can make objects available inside the edge networks. NetInf name resolution supports today's URLs, hence, any already existing object can be used in NetInf right away. URLs pointing to already existing copies can be registered as object locators in edge NRSes and objects can be retrieved via existing protocols like TCP/IP. NetInf routers are not required for a pure name resolution approach.

NDOs are advertized to an NRS using the `PUBLISH` message that mainly contains a unique message ID, the object name, a set of routing hints (e.g., the locator of the NDO copy), and optionally some metadata. By supporting different kinds of routing hints like lower-layer locators (e.g., IP, URLs), indirections to other NetInf names, and other protocol-specific routing hints, this approach offers a flexible mechanism to adapt the data retrieval to heterogeneous underlying networks. To retrieve the advertized information from an NRS, the `GET` message is used.

NetInf does not dictate a single global NRS but can support multiple NRSes that all support the same NetInf interface. Different types of NRSes are possible, e.g., a local broadcast NRS for local ad hoc networks, a global NRS, an NRS that focuses on handling highly dynamic network topologies such as the Late Locator Construction (LLC) [50] approach, and local network NRSes.

Operating an NRS in their local network allows network providers to improve traffic engineering. Network providers can select appropriate object copies during the resolution process to reduce and control network traffic and, potentially, load at the caches and data servers. This gives network operators a strong incentive to invest in NetInf NRSes, thereby supporting the NetInf migration process.

In general, there are three operational modes for routing hint selection. First, the NRS can return all its known routing hints and the requester selects one or more routing hints to retrieve the data. Second, the NRS can return a preselected set of routing hints based on its network knowledge, resulting inter-domain traffic, server load, internal policies, etc., giving the NRS full control of the selection process.

In a hybrid approach, the NRS can return a prioritized list of routing hints. This leaves the final selection to the user while still making use of the internal knowledge of the NRS. Choosing between these three operational modes is up to the NetInf network provider and depends on operational considerations.

For a fully functional, world-wide Network of Information, at least one global NRS is required. We have developed a generic, hierarchical name resolution framework that interconnects separate local NRSes into a global NRS infrastructure. Two implementations of the framework have been developed: the MDHT system (Section 5.3.2) and the HSkip system (Section 5.3.3). Both systems form a global, hierarchical NRS that is topologically embedded in the underlying network to improve scalability, latency, and locator selection for efficient information dissemination. The hierarchical, topological embedding of both systems combined with their registration and retrieval scheme ensures that close-by locators are automatically preferred over farther away locators during name resolution. To reduce load at the global level, caching of NRS entries can be used at lower-level NRS nodes. Details about the generic name resolution framework and both implementations are described in Chapter 5.

The NRS can also collect statistics about object popularity. The NRS is well suited for that as it aggregates resolution requests by many users. In a topologically embedded NRS, dedicated NRS subsystems are responsible for separate subnetworks. Thereby, the statistics collected by a local NRS automatically represent local popularity, making caching even more efficient when combined with *local* caches.

2.2.7 *Inter-Domain Communication*

Figure 11 shows an interconnection of different NetInf domains, connected via a central default-free zone (DFZ). Like in today's Internet, edge domains can decide internally on NetInf routing/forwarding, adapted to a domain's needs. In the DFZ, the NetInf BGP routing system is used, a variant of today's BGP combined with a global resolution system for aggregation. Routing in the DFZ is not the focus of this thesis, hence, only a small example is given below. The DFZ is described in some more detail in reference [19].

The example in Figure 11 shows NetInf's inter-domain routing that relies on NetInf's hybrid object retrieval using name resolution and name-based routing. The hexagons are NetInf routers. The messages exhibit type and additional parameters (e.g., requested NDO name, label stack, routing hints). The example assumes that no cached copy is available at first. In step 1, a client in *J1*'s access network sends a *GET* request for the NDO with the name `ni://example.com/foo;YY`. Note that this name contains an (optional) *authority* (`example.com`) as

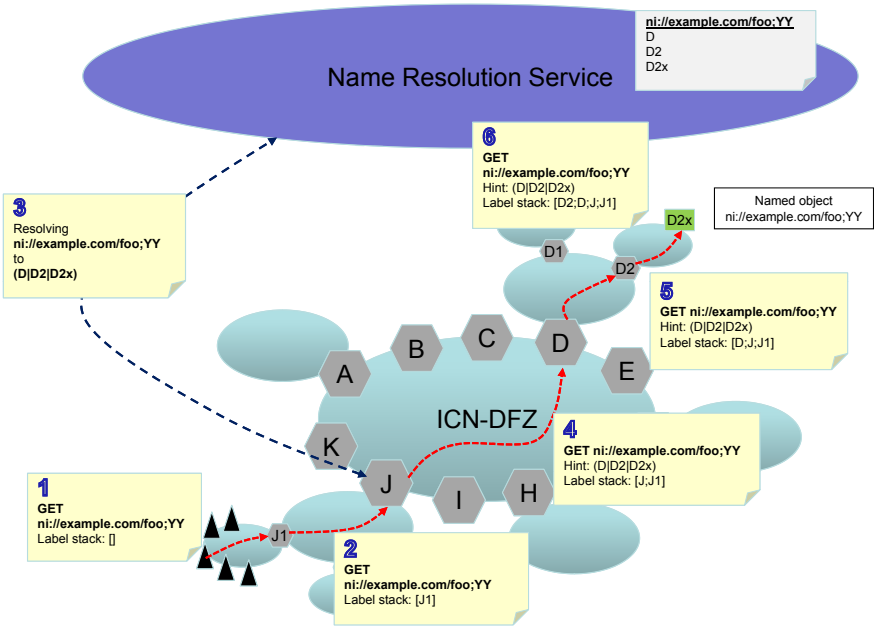

Figure 11: NetInf inter-domain scenario (triangles = client nodes; hexagons = NetInf routers; green rectangle = destination node; gray ovals = networks; yellow notes = requests; gray note = NRS bindings)

defined in our RFC "Naming Things with Hashes" [28]. The authority field can be used to assist the object retrieval process but is not used in this example. The name also defines the utilized digest algorithm (`foo`) and digest value of the data object (YY).

In step 2, the request is forwarded to router *J* via a default route (step 2). While forwarding the request, the names of involved NetInf routers are added to the label stack in the request (here router *J1*) to construct a source route. Router *J* lacks routing information. Therefore, it consults an NRS, which resolves the NDO name into a set of routing hints (D|D2|D2x) (step 3), which are added to the request. The routing hints typically do not name the end-point destination node of the request but are rather used to forward to the next hop; node *D* in our example (step 4). Node *D* belongs to a different provider with its own routing/forwarding scheme, e.g., based on the provided routing hints (step 5). Finally, the request reaches node *D2x* that holds a copy of the requested NDO (step 6). Then, the information collected in the label stack is used to forward the NDO back to the requester (not shown in Figure 11).

2.2.8 *Search*

The NetInf architecture supports both *global search services* similar to a Google search today as well as *local search services*. Both are addressed using the `SEARCH` request as described in Section 2.2.4.2.

Local search services are of special interest for NetInf to support local, infrastructure-less network setups like ad hoc scenarios. Such a service can be offered by a single NetInf node as well as jointly by multiple nodes in a distributed fashion. For such distributed, scalable search services, structured P2P systems have gained increased importance as enabling technology. At the same time, the importance of mobile, resource-constrained devices like smartphones and netbooks for accessing and storing data as well as searching and offering data in such local scenarios has rapidly increased during recent years. Therefore, we have investigated methods how to build distributed, P2P-based search services using resource-constrained NetInf devices that can offer advanced search services like geographic search. To keep this thesis focused on the main NetInf aspects, only a brief summary of these results is following in this section. More details, including the full system architecture description and simulation results, can be found in our related publications [24, 25, 26].

To support geographic search and similar search services, complex queries such as multi-attribute and range queries have to be supported by the search infrastructure. Current distributed approaches for resolving complex search queries typically require multiple messages to resolve a single search request. This generates significant messaging overhead and increases the response latency. To reduce the messaging overhead and the search latency, some approaches like the Multi-Attribute Addressable Network (MAAN) [51] use static replication. However, this results in high main memory requirements and large data transfers each time a device joins the P2P network. Those drawbacks can be tolerated for P2P networks that mainly consist of fixed, powerful nodes like PCs but are intolerable for resource-constrained nodes with high churn, like mobile devices. To enable distributed resolution of complex queries on resource-constrained devices, e.g., in local NetInf scenarios, we have developed an improved search mechanism.

Compared to MAAN, our approach significantly reduces the memory footprint and bandwidth requirements (up to a factor of $3 - 5$, depending on the load model and the type of query in our sample scenario). At the same time, the good latency properties and the low messaging overhead of MAAN are retained on average. This is achieved via a dynamic replication scheme, which introduces an adjustable trade-off between memory footprint and search latency. Thereby, our approach makes efficient, distributed resolution of complex queries on resource-constrained devices feasible.

2.3 RELATED WORK

Section 2.3.1 first gives a general overview of the work related to the ICN approach in general and to NetInf specifically. Based on this

overview, Section 2.3.2 provides an in-depth comparison of NetInf with the main recent ICN projects which have been introduced in Chapter 1.5.

2.3.1 *General Overview of ICN-Related Work*

The following related work is subdivided into several sections, starting with architectures that implement the ICN paradigm in general (Section 2.3.1.1), event notification architectures (Section 2.3.1.2), P2P technologies (Section 2.3.1.3), DTN architectures (Section 2.3.1.4), and publications focusing on selected aspects of the ICN paradigm (Section 2.3.1.5).

2.3.1.1 *ICN Architectures*

Several projects have developed an ICN architecture in recent years. The architectures all center around the content itself and around efficient content distribution. However, the projects differ in their focus, design choices, and resulting solutions.

The Translating Relaying Internet Architecture integrating Active Directories (TRIAD) project [11] has created one of the first next-generation architectures based on the information-centric paradigm. TRIAD defines an explicit content layer that provides content routing, caching, connection setup, load balancing, and object naming. In contrast to NetInf, TRIAD uses a hierarchical namespace and performs aggregation based on the hierarchical names for name-based routing. The object namespace does not provide name–data integrity like in NetInf. In addition to content distribution, TRIAD also incorporates content transformation (e.g., adapting web pages for small screen resolutions), which sets it apart from other ICN projects.

The DONA project [12] is described in Section 1.5.1. Like NetInf, it uses a flat namespace that allows for name–data integrity checking. Objects can be replicated to several servers but servers have to be authenticated in order to be allowed to distribute content, which is not the case in NetInf. In contrast to NetInf, DONA only relies on name-based anycast routing to retrieve objects.

The CCN project [52] (followed by the NDN [15] project) is described in Section 1.5.2. It also uses a name-based routing approach to retrieve objects. In contrast to NetInf, CCN's name-based routing approach purely relies on routing state in the network routers during data transport. Name-based routing is based on CCN's hierarchical namespace. Names can be human-friendly but require a PKI for name–data integrity checking or some other external trust source, which is not the case for NetInf. CCN performs caching of small data chunks in each network router with a focus on on-path caching. In addition to on-path caching, NetInf also supports off-path caching and peer-side caching based on its complementary name resolution.

The PSIRP architecture [16] (continued in the PURSUIT project [53]) is described in Section 1.5.3. In contrast to NetInf, PSIRP fundamentally builds on the publish/subscribe paradigm to find and retrieve data in the future Internet. A rendezvous mechanism matches publishers and subscribers. A name-based routing approach is used to forward data based on routing information stored in the packet, i.e., routers do not need to keep forwarding state. A name-resolution-based object retrieval approach like in NetInf is not supported.

Several other projects relate to the ICN ideas or build on ideas from the above projects. The Content-Oriented Networking: a New Experience for Content Transfer (CONNECT) project [54] builds on CCN and aims to complement CCN with proposals in the area of traffic control, naming, routing/forwarding, replication and caching, and deployment strategies.

The CONVERGENCE project [55] focuses on enhancing the Internet with a content-centric, publish/subscribe-based service model. A strong focus lies on the information model using containers for all kinds of digital content like audio data, video data. These containers are accessed via a publish/subscribe mechanism. The underlying information access is performed via a name-based routing mechanism. Unlike CCN and similar to NetInf, CONVERGENCE uses a stateless routing mechanism and source routing by storing information for the return path in the request packet.

The Content Aware Searching retrieval and sTreaming (COAST) project [56] focuses on a content-aware delivery architecture with "on the fly" identification and distributed "on-line" discovery. It makes heavy use of deep packet inspection in the network. In addition, it supports content adaptation and enrichment similar to TRIAD.

The architecture by Xu et al. Service and Information Oriented Network Architecture (SIONA) [57] builds on IP to construct an information-centric network architecture. IP addresses are explicitly made part of the object and service naming structure. Based on these names, SIONA performs name-based routing. The request packets leave "bread crumbs" in network routers, which are used to route responses back along the request path, closely related to CCN's approach.

The COntent Mediator architecture for content-aware nETworks (COMET) [58, 8] builds an overlay network that aims to become a global quality of service (QoS)-aware content access mechanism for all kinds of content. It can be deployed on top of today's Internet architecture as well as on top of future Internet architectures. COMET supports content distribution in a content- and network-aware way based on a name resolution approach.

The eXpressive Internet Architecture (XIA) [59] aims to not limit the architecture to a single communication scheme (e.g., content-, host-, or service-centric) but supports multiple schemes in parallel.

It also supports yet unforeseen communication schemes. This is achieved via flexible addressing and forwarding semantics. The communication scheme defines the desired *intent*, e.g., to retrieve a certain piece of content, and additionally several alternative *means* how to achieve this intent. For example, the means could be host-centric, i.e., via a specific web server running on a host in a defined network, or the means could be information-centric, i.e., via directly addressing the content in some information-centric underlying network mechanism. Hence, XIA can build on and integrate ICN mechanisms as developed, e.g., by NetInf.

The MultiCache architecture [60] proposes a pure overlay solution. It has a strong focus on caching and multicast for an information-centric extension of today's Internet. The architecture is based on the Pastry [61] distributed hash table (DHT) overlay and on the Scribe [62] multicast overlay. In addition, providers deploy proxy overlay access routers and in-network caches at their point of presences (PoPs). Based on this setup, data is delivered in a hybrid approach either based on multicast (e.g., in case of flash crowds) or via unicast, making use of Pastry's inherent locality awareness to locate nearby caches.

The Cache-and-Forward (CNF) architecture [63, 64] uses a name resolution mechanism to resolve content identifiers into IP addresses, which is related to NetInf's NRS-based approach. CNF introduces dedicated routers in the network that can cache content and forward content hop-by-hop, quite similar to the CCN approach. However, forwarding is performed based on the previously resolved IP addresses. The project takes special measures for mobile nodes by introducing so-called post offices at the network edges where participants with intermittent connectivity can deliver and fetch their content.

2.3.1.2 *Event Notification*

Event notification and some approaches based on the event notification principle (e.g., *Content-Based Networking (CBN)* [65]) have certain similarities with ICN. Specifically, both are based on the idea of naming and addressing content. However, ICN typically uses a receiver-initiated communication paradigm whereas event notification typically uses a sender-oriented communication paradigm. Furthermore, event notification focuses on distributing typically small, short-lived information (e.g., a local storm warning) whereas ICN focuses on disseminating typically large, long-lived content (e.g., a video file), although ICN is not restricted to large, long-lived content. As a result of these differences, both types of communication require significantly different network operations. Despite these differences, the existing similarities encourage attempts like CBN to integrate both event notification and ICN [66]. In CBN, object naming and subscription is based on a set of predicates that are used to identify matches between subscriptions and publications. CBN builds on ideas from the

Scalable Internet Event Notification Architecture (SIENA) [67] event notification architecture.

2.3.1.3 *P2P Technologies*

ICN also has similarities with P2P technologies. Both focus on efficient content dissemination and both try to increase content availability. However, P2P is currently only used as an application-specific technology, i.e., most applications currently run their own dedicated P2P overlay. Therefore, synergies between applications running on top of different P2P overlays are limited. Moreover, P2P is an overlay technology working purely on top of the current Internet infrastructure whereas ICN focuses on a strong integration with the network technology and partly supersedes today's network technology. Among others, a strong integration with the network makes ICN approaches network-topology-aware. In contrast, a lack of topological embedding produces significant problems for P2P technologies. Recent projects like P4P [49] and IEEE ALTO [48] try to address these problems.

2.3.1.4 *Delay-Tolerant Networking*

DTN [47] is a research area that is related to ICN in that it provides a hop-by-hop store-and-forward approach for data transfer. NetInf utilizes a Convergence Layer approach related to the one proposed in the DTN context. While DTN is generally building on a traditional host-centric addressing paradigm using end-point identifiers, projects like DTN Pub/Sub Protocol (DPSP) [68] and Haggle [69] bridge the DTN and ICN paradigm by providing a kind of publish/subscribe system for DTN that uses a data-centric addressing approach. The addressing scheme is based on metadata in the form of key-value pairs. This approach is, however, closer related to publish/subscribe architectures like SIENA and CBN than to NetInf.

2.3.1.5 *Selected Aspects of ICN*

In addition to the afore described projects that aim at a full ICN architecture, there are several publications focusing on various specific ICN aspects, including the overall ICN paradigm [13, 70, 71], caching [72, 73, 74, 75, 76], routing and transport [77, 78, 79, 80], and naming and security [81, 7, 82]. Finally, some recent ICN surveys have been published [83, 84, 85] that complement our ICN survey as published in reference [19].

In general, NetInf shares many goals and assumptions with other ICN approaches but differentiates itself from the other projects by its broad architecture that covers a wide range of ICN aspects from upper-layer aspects like data search and object model to lower-layer

aspects like data retrieval. Unlike most other approaches, NetInf combines name resolution and name-based routing into a flexible and synergistic data discovery and retrieval approach. Moreover, design choices of a single aspect, e.g., the namespace, have a strong influence on the resulting architecture. For example, NetInf's flat, self-certifying namespace results in many differences compared to the presented projects in the areas of trust and security, object retrieval via name-based routing and name resolution, scalability, and aggregation.

2.3.2 *ICN Architectures: Design Choices and Trade-Offs*

Based on the NetInf architecture description given in this chapter, we can now provide a more detailed discussion of the main recent ICN architectures, DONA, CCN/NDN, and PSIRP/PURSUIT, and compare them with NetInf. This section complements the brief initial overview given in Section 1.5 and discusses the design choices and trade-offs of different ICN architectures. The discussion focuses on the following architectural aspects: naming and security, API, name resolution and routing/forwarding, caching, transport, and mobility.

2.3.2.1 *Naming and Security for Data Objects*

DONA names NDOs with a flat namespace in the form *P:L*. NetInf has adopted this basic structure for its flat name structure. In the DONA architecture, *P* is the globally unique *principal* field which contains the cryptographic hash of the publisher's public key, and *L* is the unique object label. As *P* identifies the *publisher* (i.e., the entity publishing an object in the network) and not the *owner* (i.e., the entity creating the secure name of the object) as in NetInf, republishing the same content by a different publisher (e.g., by an in-network cache) generally results in a different name for the same content. While this can be circumvented with specific means in DONA (e.g., via wildcard queries or principal delegation), it might complicate benefiting from all available content copies.

The CCN namespace is hierarchical for better routing scalability through name-prefix aggregation. The names are rooted in a prefix, unique for each publisher. The publisher prefix makes it possible for clients to construct valid names for data that does not yet exist, and publishers can respond with dynamically generated data. CCN names are used both for naming information and for routing purposes. The granularity of the names is very fine: single chunks (packets) are named.

PSIRP makes use of two type of names to name NDOs: *rendezvous identifiers* and *scope identifiers*; they both belong to a flat namespace. The NDOs are mapped to rendezvous points, which are used to establish contact between publishers and subscribers. PSIRP also uses

forwarding identifiers, which are used by the forwarding fabric to transport data after contact is established at a rendezvous point. The forwarding identifiers (Bloom filters in LIPSIN [86]) are not names for NDOs; they are transient and identify a path from the publisher to the subscriber.

DONA, NetInf, and PSIRP use flat namespaces. All three can check the *name–data integrity* solely based on the data's name, i.e., without requiring external means like a PKI. To achieve this for static data, the cryptographic hash of the content can be included as object label. For dynamic data, name-data integrity is achieved by providing a signature of the content's hash as metadata with the NDO, signed by the public key corresponding to the hash in the ID's authenticator field. In this way, the object identifier is securely bound to the data and also allows to handle data that does not yet exist. Using self-certifying identifiers is a deliberate trade-off between desirable name–data integrity properties and human readability as the identifiers contain a cryptographic hash[3]. As a result, additional means are potentially required to securely bind more human-friendly application-level names to these identifiers. On the other hand, self-certifying identifiers allow to check whether the received data matches the identifier used in the data request *without* requiring a PKI, which simplifies the security model and makes it more reliable. For example, no trust in the PKI is required and data integrity can be verified off-line.

CCN names typically do not contain the publisher's public key (PK) (nor its cryptographic hash). The hash of static content is typically also not explicitly part of the name used by requesters. While this improves human readability, it complicates self-certification. Data integrity is also achieved by signing the content with the publisher's secret key, but trust in the signing key always needs to be established using external means since there is no direct binding between the key and the NDO name. CCN supports multiple different means to verify trust in the key, e.g., direct experience, information provided by friends, a trusted directory of keys, or a global PKI.

2.3.2.2 *Application Programming Interface*

The main API calls of most presented ICN approaches have many similarities to the `PUBLISH` and `GET` primitive as outlined in Section 1.3. However, the primitives address different underlying network entities in different approaches. In PSIRP and NetInf, publications are addressed to the rendezvous system and NRS respectively to register new names, resulting in corresponding binding entries. In CCN and DONA, publish is used to fill the routing tables of the content routers/resolution handlers. Likewise, in NetInf and PSIRP, the `GET` primitives are addressed to the resolution/rendezvous system respec-

3 One could argue that many URLs in today's Internet are already human-unfriendly, hence, self-certifying identifiers are not introducing a new problem here.

tively, followed by a second step to retrieve the data from the NDO source. This second step is, however, typically hidden from the API as locators and routing hints are typically not exposed to ICN applications. In CCN and DONA, the `GET` primitive is handled directly by the routers/resolution handlers. NetInf is a hybrid approach as it can send a `PUBLISH/GET` to the NRS for name resolution as well as directly to a router for name-based routing. This will be discussed in more detail in the next section.

2.3.2.3 *Name Resolution and Routing*

DONA uses name-based routing to route the query via the resolution handlers (RHs) to a copy of the requested NDO. Nodes that are authorized to serve data use the `REGISTER(P:L)` primitive to register a datum with an RH. Each domain has an RH. To resolve a name, the `FIND(P:L)` primitive is used. Both primitives allow wildcards to be used in the place of P or L. RHs are organized in a hierarchical structure. Every request that an RH cannot handle is forwarded to its parent RH. The RH tries to find a copy of the content closest to the client. Once a copy is found, the data is returned to the client, potentially via the RH request path as shown in Figure 2 (page 12) when the RHs perform caching. Otherwise, the data can also be returned directly to the client. Originally, DONA used longest-prefix matching for name matching; currently, the more scalable deepest-match approach is being proposed [7].

CCN also uses name-based routing. Clients ask for a data object by sending `INTEREST` packets, which are routed towards the publisher of the name prefix using longest-prefix matching in the Forwarding Information Base (FIB) of each node. The FIB can be built using routing protocols similar to those used in today's Internet. The CCN nodes keep state for each outstanding request in the Pending Interest Table (PIT – see Figure 3 page 13). This makes request aggregation possible, i.e., when the same node receives multiple requests for the same NDO, only the first is forwarded towards the source. When a copy of the data object is encountered on the path, a data packet containing the requested object is sent on the reverse path back to the client (all nodes along the path cache a copy of the object). The reverse path is found using the state that the `INTEREST` packet has left in the nodes.

PSIRP uses a resolution model where the resolver is called *rendezvous point*. The data returned to the client can, potentially, take a different path than the name resolution/rendezvous path. The rendezvous point does neither have to be on the path to the publisher nor does it have to hold a copy of the data. Data is forwarded using a source routing approach called *zFilters*: a Bloom filter describing the route is built by the rendezvous point and used to forward packets from the selected source to the destination. The Bloom filter is attached to the packet itself, and it contains all names of the links

that have to be followed. The Bloom filter approach allows to trade off packet length against wasting network resources. A large Bloom filter gives fewer false positives, thus resulting in less packets being forwarded on links without any receiver.

Comparing the alternatives, one can note that an NRS-based approach can simplify migration as the routing and forwarding underlay does not have to be modified. On the other hand, name-based routing eliminates the name resolution step completely, thereby potentially reducing the overall latency and simplifying the overall process. It is also not clear how `INTEREST` aggregation can be done in an efficient way in an NRS-based approach.

2.3.2.4 *Caching*

In DONA, caching is inherent in the architecture. Any RH can also serve as a cache. To populate its cache, the RH modifies the FIND request so that the NDO is returned to the RH before it is returned to the original requester. Any cache can respond to a FIND request by returning a cached copy of the NDO.

CCN can cache both requests (through its request aggregation) and objects. CCN routes a request for data towards the publisher and makes use of any cached data copies along that path. A CCN node can keep received interest packets in a *pending interest table* and, thus, suppress forwarding of subsequently received requests for the same object, if it has already sent a request. Object copies can also be found by local search. As single packets are the atomic objects in CCN, it is possible that only a part of a bigger object is cached.

In PSIRP, caching is limited to the scope of the rendezvous point for the identifier associated with an object. Within that scope, an object can be cached in multiple nodes.

NetInf can cache requests, name-resolution results (i.e., routing hints), and objects. Generally, name-based routing and name resolution is employed to find next-hop options. When a node receives a `GET` request, it can decide to employ a pending interest table-like structure for request aggregation. It can also decide to perform a dedicated NRS lookup for each received `INTEREST`, so that performing request aggregation becomes a policy decision. There are two ways to make use of a cached object copy: first, any copy can be found directly by querying the NRS, provided that the copy is explicitly registered there or discovered by the NRS via other means (e.g., broadcast). Second, the copy can be found by a cache-aware NetInf transport protocol on the path to a location known to hold a copy, for example, a location retrieved from the name resolution system.

2.3.2.5 *Transport*

By *transport* we refer to two concepts: 1) the fundamental request and response transport mechanisms and 2) transport protocol functions such as resource sharing, flow control, and reliability.

The DONA architecture does not put much emphasis on transport and relies on existing transport protocols such as TCP.

CCN defines different packet types – *`INTEREST` packets* and *data packets*, representing basic elements of a protocol. A node sending an interest packet via one of its interfaces (called *faces* in CCN) to a (set of) neighbor nodes has some expectation to receive a corresponding data packet shortly, i.e., CCN nodes operate on the principle that there is a balance of `INTEREST` and data packets. Interest and data packets work on the packet level – the assumption is that larger objects would be represented by individual chunks, and each chunk can be accessed by a unique name.

This fundamental mechanism is CCN's basis for realizing different services that are conventionally considered as "transport layer functions", e.g., reliable transmission, flow control, and multi-path communication. It is essentially up to a node's specific *strategy* to which face `INTEREST` packets should be sent and how to behave as a reaction to the received data packets.

PSIRP's basic forwarding mechanism is based on Bloom filters as described in Section 4.2. PSIRP proposes to use different names for each object to handle flow control. These names are derived algorithmically from the original name, encoding the desired receiving speed. Another option is for the receivers to publish flow-control feedback under some algorithmically derived name for the sender to possibly subscribe to.

NetInf defines a set of messages to request resources and to reply to these requests – for example by returning the requested object or by returning a locator or redirection hint. There could be multiple hops involved in forwarding such request and response messages, and each hop can potentially use a different convergence layer.

The convergence layer used in this hop-by-hop approach implements (or employs) a specific transport protocol that provides the appropriate resource sharing and reliability mechanisms for the corresponding network path (segment). This approach allows for localized transport mechanisms, i.e., for challenged wireless link, without degrading performance on other hops.

2.3.2.6 *Mobility*

In this section, we discuss three types of mobility and how they relate to information-centric networks. The first type is *client mobility*: a client moves during or in between requesting data objects. The second type is *content mobility*: an object or set of objects changes lo-

cation. The third type is *network mobility*: when an entire network moves, e.g., a body area network or a train network.

A key feature with information-centric networks is that all copies are equal. For client mobility this means that when a client moves there is no need to keep an association to a specific copy alive. Instead, new associations can be established to alternative copies close to the new location. All discussed ICN approaches can easily find a new, appropriately located copy when a client moves.

When the content (i.e., the server storing the content) moves, the routing information in the network needs to be updated. For NRS-based approaches like NetInf, this means that a new locator is registered when an NDO is published with a new location in the network. For approaches like CCN that use name-based routing and hierarchical naming to aggregate route announcements, the situation is more problematic. If the full aggregate of objects is moving, the new route announcement needs to be propagated and old routing entries need to be replaced before routing converges, causing similar issues that we see in today's IP networks. In the case that only parts of the objects belonging to an aggregate moves to a new location, e.g., a company employee takes a laptop on a trip, there will also be a need to fragment the routing tables. In agile network scenarios, this could defeat the benefits of having a hierarchical namespace. In PSIRP and DONA, content mobility involves updating the routing state in the rendezvous nodes and RHs, respectively. However, neither suffers from the aggregation problem due to their flat namespaces. The same applies to updates of NetInf's routing state for the name-based routing approach.

Moving an entire network (including all content hosted by this network) can cause a storm of routing/resolution updates, especially if the moving network consists of a heterogeneous set of publishers, like in a train. This can be problematic for both name-based routing schemes as well as NRS-based approaches if they do not allow for relative route announcement or relative locators. I.e., if the publishers can express their location as relative to the location of the moving network, only the location of the network needs to be updated when it moves, not all the locations of objects currently attached to the moving network. An example of such a routing/forwarding system supporting relative locators is the NetInf LLC resolution system [50].

NetInf can support all three types of mobility. Content/content-provider mobility is supported via the NRS. When a data copy moves, this movement results in an update in the NRS to account for the new network location. However, creating new object–locator bindings is the rule in NetInf. Hence, these updates are a standard operation in NetInf, which can be performed fast as the NRS is especially optimized for this kind of operation. In addition, these update operations do not result in inflated lookup tables as each old binding is

exchanged against a corresponding new binding. The handling of client mobility heavily depends on the data transport and forwarding technology used in NetInf. In general, NetInf can support different data transport and forwarding technologies. For example, the integrated NRS and routing/forwarding system Global Information Network (GIN) [87], which has been developed as part of the NetInf project, natively supports client mobility without inflating the routing tables. LLC, on the other hand, provides very good support for network mobility.

In PSIRP, clients can just unsubscribe, switch networks, and resubscribe again. A new path/subtree will be computed by the routing layer. Buffering and sequence numbering allow seamless handovers. Content provider mobility is more complex and involves updating the routing state in the rendezvous nodes.

Client mobility in CCN is inherent. A client can switch to another network and continue to issue `INTEREST` packets. The strategy layer could notice the switch and re-issue all the pending `INTERESTS`, without waiting for them to time-out. Content-provider mobility is more complex: a content provider would have to update the routing tables of all relevant neighboring nodes. Furthermore, moving content providers would pollute the routing tables with specific prefixes, countering the advantage of prefix aggregation.

Client mobility in DONA is achieved in a way similar to PSIRP: clients can de-register from their previous location and re-register at the new location. De-registration is not mandatory, as Resolution Handlers can expunge outdated content entries.

2.3.2.7 *Comparison Table*

Table 1 summarizes the different properties and design choices of the analyzed approaches according to the main ICN characteristics discussed in Section 1.3.

2.4 SUMMARY

The NetInf architecture is targeted at global-scale communication with support for many different types of networks and deployments, including traditional Internet access and core network configurations, data centers, as well as challenged (e.g., DTN) and infrastructure-less networks. NetInf's approach to connecting different technology and administrative domains into a single information-centric network is based on a hybrid name-based routing and name resolution scheme.

The design is based on the idea of not limiting it too much by implicit assumptions how networks are built and what underlying communication services are available, thereby allowing future adaptability and compatibility. A minimal set of node requirements for

	DONA	**CCN**	**PSIRP**	**NetInf**
Namespace	flat with structure	hierarchical	flat with structure	flat with structure
Name–data integrity	signature, PKI independent	signature, external trust source	signature, PKI independent	signature or content hash, PKI indep.
Human-friendly names	no	possible	no	no
Information abstraction/ object model	no	no	no	yes
NDO granularity	objects	packets	objects	objects
Routing aggregation	publisher/ explicit	publisher	scope/ explicit	publisher
Routing of NDO request	name-based (via RHs)	name-based	NRS (rendezvous)	hybrid, NRS & name-based
Routing of NDO	reverse request path or direct IP connection	reverse request path using router state	source routing using Bloom filter	reverse request path or direct IP connection
API	synchronous GET	synchronous GET	PUBLISH/ SUBSCRIBE	synchronous GET
Underlay	IP	many, incl. IP	IP/PSIRP	many, incl. IP

Table 1: Summary of characteristics of ICN approaches.

supporting the NetInf object model, naming format, and protocol messages provides the necessary common denominator. The convergence layer approach enables extensibility with respect to new underlying network technologies. Finally, the hybrid name-based routing and name resolution approach allows us to inter-connect such different domains and flexibly adapt to the specific network context.

This chapter concludes the description and discussion of the general NetInf architecture. The following chapters present more in-depth discussions of two critical components of the NetInf architecture, secure naming and name resolution, and discuss NetInf prototyping.

3

SECURE NAMING

This chapter is based on work published in references [27, 28, 30, 32].

As described in Section 2.2.2, we have developed a general URI scheme for named information called *the ni URI scheme* [28]. In this chapter, we describe details about the *general NetInf naming scheme* that builds the conceptual foundation for the more implementation-oriented ni URI scheme. Note that the ni URI scheme currently only contains a subset of the features defined by the general NetInf naming scheme in this chapter.

As the focus of this thesis is not on network security, this section cannot and does not intend to provide a thorough security analysis of NetInf. Instead, we focus on the details of the NetInf naming scheme. As the naming scheme is closely related to security aspects such as self-certification, we also provide a more detailed discussion of this and related security features with a focus on trust in the retrieved data and the publisher. This discussion can serve as a starting point for a more thorough security analysis in the future.

3.1 INTRODUCTION

Security in an information-centric network needs to be implemented differently than in current, host-centric networks. In the latter, most security mechanisms are based on host authentication and then trusting the data that the host delivers. In an information-centric network, host authentication cannot and should not be relied upon. Otherwise, one of the main advantages of an information-centric network, i.e., benefiting from any available copy, is defeated. Authenticating a random, untrusted host that happens to have a copy does not establish the needed trust. Instead, the security has to be directly attached to the NDOs, which can be done via the object naming scheme.

The NetInf naming scheme builds the foundation for NetInf's information-centric security model that integrates security deeply into the network architecture. In this model, trust is based on the information itself. Each NDO is given a unique ID (i.e. name) with cryptographic properties. Together with additional metadata, the ID can be used to verify data integrity and several other security properties. In comparison with the security model in today's host-centric networks, this

approach significantly reduces the need for trust in the infrastructure, including the hosts providing the data, the channel, and the NRS.

In Section 3.2, the requirements for the naming scheme are discussed. Section 3.3 describes its core functionality and Section 3.4 describes and analyzes its main security properties. The results of prototype evaluation are presented in Section 3.5 and the relation to other work is addressed in Section 3.7. Section 3.8 summarizes this chapter and discusses details on how the features of the general NetInf naming scheme are implemented in the concrete ni URI naming scheme Internet draft.

3.2 REQUIREMENTS

First of all, the naming scheme has to be generic so that it can name many different kinds of entity, including static and dynamic NDOs, services, network nodes, people, and real world entities like places and objects. This requirement results from the flexible NetInf object model that is able to represent many different entity types. For the same reason and to adapt to future needs, the naming scheme should be extensible, i.e., it should be possible to add new information (e.g., a chunk number for BitTorrent-like protocols) to the naming scheme. The need for such extensions is stressed by today's variety of naming schemes (e.g., digital object identifier (DOI) [88] or PermaLink) added on top of the original Internet architecture. These naming schemes fulfill specialized needs which cannot be met by the common Internet naming schemes, i.e., IP addresses and URLs.

To enable efficient, large-scale data dissemination that can make use of any available data copy, IDs have to be location-independent (often referred to as *identifier/locator split*). Thereby, identical data can be identified by the same ID independently of its storage location and improved data dissemination can then benefit from all available copies. This should be possible without compromising trust in data regardless of its network source. Therefore, *name–data integrity* is the main requirement for an information-centric naming scheme. Name–data integrity ensures that any unauthorized change of data with a given ID is detectable. Beforehand, secure retrieval of IDs (e.g., via search, embedded in a web page as link, etc.) is required to ensure that the user has obtained the correct ID in the first place. Secure ID retrieval can be achieved by using recommendations, past experience, and specialized ID authentication services and mechanisms.

Another important requirement of the NetInf naming scheme is *name persistence* with respect to storage location changes as discussed above and also with respect to changes of owner and/or owner's organizational structure. In addition, it would be desirable to achieve name persistence for *dynamic* data, i.e., persistent IDs in spite of content changes producing a new version of the information. However,

name persistence and self-certification are partly contradictory and achieving both simultaneously for *dynamic* content is not trivial.

From a user's perspective (at the application level), persistent IDs ensure that links and bookmarks can remain valid as long as the respective information exists somewhere in the network, reducing today's problem of "404 - file not found" errors triggered by renamed or moved content. From a content provider's perspective, name persistence simplifies data management, as content can, e.g., be moved between folders and different servers as desired without resulting in name changes. Name persistence with respect to content changes makes it possible to identify different versions of the same information by the same consistent ID. If it is important to differentiate between multiple versions, e.g., of Wiki pages, a dedicated versioning mechanism is required, and version numbers may or may not be included as a special part of the ID. Versioning in a distributed, collaborative fashion poses additional requirements, which are out of scope here.

As outlined in Section 2.2.3, we differentiate between two mechanisms to achieve trust and accountability in NetInf. The first is *owner pseudonymity*, where the owner is only identified via a pseudonym and not his/her real-world identity. The second is *owner identification*, where the owner is also identified via his/her real-world identity, such as a personal name. This separation is important to allow anonymous publication of content, e.g., to support free speech, while at the same time building up trust in an unknown owner.

3.3 NAMING SCHEME

Section 3.3.1 introduces the basic NetInf naming scheme concepts, with details about the ID structure and corresponding security-relevant metadata given in Sections 3.3.2 and 3.3.3, respectively.

3.3.1 *Basic Concepts*

In the NetInf naming scheme, any entity/NDO is represented by a globally unique ID. Together with the NDO's data and metadata, an NDO is defined as *NDO = (ID, Data, Metadata)*. *Data* contains the main information content of the NDO. *Metadata* contains information needed for the security functions of the NetInf naming scheme, e.g., public keys, content hashes, certificates, and a data signature authenticating the content. It can also include application-specific metadata, i.e., any attributes associated with the NDO, e.g., the location where a picture was taken.

In an information-centric network, multiple copies of the same NDO typically exist at different locations. In contrast to today's Internet architecture, due to the ID/locator split, those identical NDOs

have the same ID in NetInf. All NDOs under the same ID constitute an *equivalence class*, represented by the common ID.

NDOs are manipulated (e.g., generated, modified, registered, and retrieved) by physical entities such as nodes (clients or hosts), persons, and companies. Physical entities that can create or modify NDOs *in combination with a valid NDO name* are called *owners*.

Several security properties of our naming scheme are based on the fact that the ID contains the hash of a public key that is part of a public key (PK)/secret key (SK) pair. This includes owner pseudonymity, owner identification, and name–data integrity of *dynamic* content. Name–data integrity of both *static* and *dynamic* content is achieved via *self-certification*, i.e., no third party is required to first establish trust in the key pair in order to verify the name–data integrity. The PK/SK pair used for self-certification is conceptually bound to the NDO itself and not directly to the owner as in other systems like DONA. In NetInf, the PK/SK pair is only *indirectly* bound to the owner, i.e., via a certificate chain. This is important to note because it enables owner change while keeping persistent IDs. The key pair bound to an NDO is thus denoted as PK_{NDO}/SK_{NDO}.

3.3.2 *ID Structure*

The NetInf naming scheme uses flat IDs mainly because we want the IDs to be persistent. In addition, flat IDs are advantageous when it comes to mobility and they can be allocated without an administrative authority by relying on statistical uniqueness in a large namespace. Although IDs are not hierarchical, they have a specified basic ID structure, *ID = (Type, A, L)*, illustrated in Figure 12.

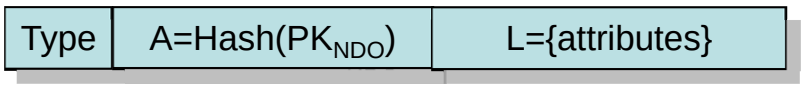

Figure 12: Basic ID structure

The *authenticator* field[1] *A=Hash*(PK_{NDO}) binds the ID to a public key PK_{NDO}. The hash function *Hash* is a cryptographic hash function, which is required to be one-way and collision-resistant [89]. The hash function serves only to reduce the bit length of PK_{NDO}. PK_{NDO} is generated in accordance with a chosen public-key cryptosystem. The corresponding secret key SK_{NDO} should only be known to a legitimate owner. In consequence, an owner of an NDO is defined as any entity who knows SK_{NDO} or any other secret key authorized by SK_{NDO} via a public-key certificate chain (see Section 3.4.2).

1 Note that the authenticator field is not to be confused with the authority field of the ni URI scheme.

In general, the *label* field L can contain a number of *identifier attributes* associated with an NDO. These attributes can be related to the data content and/or the owner or can simply be serial or random numbers. The pair (A, L) has to be globally unique. In particular, the identifier attributes:

- must provide global uniqueness if PK_{NDO} is repeatedly used for different NDOs,
- can authenticate static NDOs or their static parts, by including their hash values (as done in the ni URI scheme).

To build a flexible and extensible naming scheme with many different supported object types (including all entities listed in the requirements section), e.g., to adapt the naming scheme to the named entity type, different types of IDs are supported by the NetInf naming scheme and differentiated via a mandatory *type* field in each ID. The type field defines the variable format and structure of the label L and determines how to interpret this structure. It has to be globally standardized.

3.3.3 *Security Metadata Structure*

The security metadata is extensible and contains all the information required to perform the security functions embedded in the NetInf naming scheme. In particular, the security metadata includes:

- specification of the hash function h and the algorithm *DSAlg* used for the digital signature[2],
- complete PK_{NDO}, not only $Hash(PK_{NDO})$,
- public-key certificate chain authorizing the signing PK/SK pair (see Section 3.4),
- specification of the parts of the *data* and *attribute metadata* (if not all) that are secured via self-certification, i.e., authenticated via the signature,
- for dynamic NDOs, hash of the self-certified data, which, in addition, contains the ID of the NDO, in order to prevent unauthorized change of the type and L fields,
- signature of the self-certified data signed by SK_{NDO} or any other authorized SK from the public-key certificate chain,
- if needed, all data required for owner identification, i.e., mainly a certificate of a trusted third party certifying the identity of the owner.

2 Alternatively, this information can be specified in the ID itself, as done in the ni URI scheme.

3.4 ANALYSIS OF SECURITY PROPERTIES

In the following sections, we describe and analyze the main security-related naming properties: name–data integrity, name persistence, owner pseudonymity, and owner identification.

3.4.1 *Name–Data Integrity*

Name–data integrity of static NDOs, whose content does not change in time, can be achieved by including the hash of the self-certified data in the ID, more precisely in the *L* field of the ID. Verification of the data is then performed by computing the hash value of the retrieved data and comparing it with the hash contained in the ID. The advantage of this approach is that there is no need to resort to the metadata in order to verify the content[3]. Also, the PK/SK pair is not required here.

Name–data integrity of *dynamic* content cannot be achieved this way as it would obviously result in non-persistent IDs. Hence, for dynamic content, we make use of the PK/SK pair. We use the public key PK_{NDO} contained in the ID to securely bind the hash of the content to the ID while storing the content hash separately in the metadata, thereby keeping the ID persistent even if the content changes (Listing 1). To do so, we sign the hashed content C with the secret key SK_{NDO} corresponding to the public key PK_{NDO} contained in the object ID (or any other SK authorized by SK_{NDO}). For hashing and signing, we use the hash function *h* and the signing algorithm *DSAlg* as specified in the security metadata. Replay attacks, i.e., copying valid metadata from one NDO to another NDO with a different ID, are prevented by including the ID in the signed content (line 4). This signed content hash is subsequently stored in the associated security metadata (line 5). Note that the full PK_{NDO} is also stored in the security metadata (line 6) while the ID only contains the hash of PK_{NDO} to keep the ID shorter.

Listing 1: Creating name–data integrity for dynamic content

```
1 void createNameDataIntegrity(string content)
2 {
3   id = createID(type, hash(PK_NDO), label);
4   signedC = sign(hash(content+id), SK_NDO); //Sign with SK_NDO
5   storeInSecurityMetadata(signedC);
6   storeInSecurityMetadata(PK_NDO);
7 }
```

Verification of the content (Listing 2) consists of first verifying if the PK used for signing is equal to PK_{NDO} by comparing the hash values

3 This presumes that the utilized hash and signature functions are defined directly in the ID as done in the ni URI scheme.

(line 5). Alternatively, the PK used for signing can also be authorized by a valid public-key certificate chain originating in PK_{NDO}/SK_{NDO} as described in Section 3.4.2 (this is not shown in the listings for clarity reasons). Second, the signature of the self-certified data has to be verified (line 8). This ensures a secure binding between the self-certified data and the ID. Only the legitimate owners can produce the valid signature and any change to the self-certified data performed by other entities can be detected, assuming that the functions *h* and *DSAlg* are secure. If an unauthorized PK/SK pair is used for signing (to produce a new signature for potentially modified data), the ID will change. Finally, the signed hash stored in the security metadata is compared with the hash of the retrieved content (line 11).

When the content of a dynamic data object should be changed, the owner only has to recalculate the new content hash, sign it again with its secret key SK_{NDO}, and exchange the old signed hash in the metadata with the new signed hash.

Listing 2: Verifying name–data integrity of dynamic content

```
1  boolean checkNameDataIntegrity(string content)
2  {
3    PK_NDO = getPK(securityMetadata);
4    hashOfPK = getHashPK(id);
5    pk_ok = compare(hash(PK_NDO), hashOfPK);
6    if pk_ok {
7      signedC = getSignedContent(securityMetadata);
8      signature_ok = verifySignature(signedC, PK_NDO);
9      if signature_ok {
10       hashOfContent = getHashOfContent(signedContent, PK_NDO);
11       hash_ok = compare(hashOfContent, hash(content));
12       if hash_ok {
13         return true; //All checks ok!
14       } else return false;
15     } else return false;
16   } else return false;
17 }
```

3.4.2 *Name Persistence*

The NetInf naming scheme can ensure persistent IDs in spite of a changing storage *location*, the *content* itself, the *owner* of a data item, as well as owner's *organizational* changes.

Independence of *organizational changes* is an inherent feature of the NetInf naming scheme as IDs, especially the authenticator field, are flat and do not reflect organizational structures as in other approaches, e.g., CCN.

Location independence results from the ID/locator split that the naming scheme builds upon. The NetInf IDs are *dynamically* bound to

one or multiple network locators where copies of the NDO are stored. Hence, when a locator changes, the ID remains persistent and only the binding has to be adapted, which is managed outside the naming scheme by an NRS.

Content independence is achieved via our approach for dynamic data, i.e., by storing the signed content's hash in the associated metadata instead of in the ID.

Owner independence can be achieved in two ways, by the less complex *basic approach* and the *advanced approach*, which offers more flexibility, but is also more complex. The owners can choose the approach more appropriate to their needs.

The *basic approach* is based on the fact that PK_{NDO} contained in the ID is bound to the NDO itself, and not to a specific owner. Therefore, when the owner changes, the corresponding SK_{NDO} can be securely passed on to the new owner. The new owner will subsequently use the same PK_{NDO}/SK_{NDO} pair. Thereby, the PK_{NDO}/SK_{NDO} pair is not changed and the ID remains persistent. This approach is simple, but requires a secure (confidential and authenticated) channel for passing on the SK_{NDO}, hence, it is not robust with respect to disclosure of the secret key. Moreover, this basic approach restricts building up trust in the owner based on PK_{NDO} in the context of owner pseudonymity. Owner pseudonymity is restricted to the first owner of the object, i.e., the creator of the object name. An owner change (which is typically not transparent to the object receiver) might result in falsely trusting the new owner based on the good reputation of the previous owner. This trust might only (partly) be justified if the old owner can influence which new owner is chosen.

In the *advanced approach*, each new owner can use a new key pair PK/SK for name–data integrity and, possibly, also a new hash function h and signing algorithm *DSAlg*. This eliminates the need for securely transferring the secret key and ensures a certain level of robustness with respect to secret key disclosure. To keep the ID persistent, the hash of the original PK_{NDO} in the ID remains unchanged. However, the metadata is signed by the secret key of the latest new owner, SK_{latest}, and verified by the corresponding public key, PK_{latest}. Hence, owner pseudonymity and trust can be achieved for each owner individually.

The PK_{latest}/SK_{latest} pair used for signing needs to be securely bound to the ID. This is achieved by using a public-key certificate chain authorizing the PK_{latest}/SK_{latest} pair by the original PK_{NDO}/SK_{NDO} pair (Figure 13). The public-key certificate chain provides a secure binding between PK_{NDO} and PK_{latest} and, hence, also between PK_{NDO} and SK_{latest}. Each particular public-key certificate includes the PK of the former owner and the new PK of the new owner. It also contains the NDO's ID to bind the authorizations to this ID. The specification of a new function h and algorithm *DSAlg* is

also included in the public-key certificate if those have changed. The certificate is signed by the SK of the former owner.

To ensure validity of the digital signature, the whole certificate chain, stored in the NDO's security metadata, needs to be verified. This is done by verifying each individual certificate along the chain starting with the first certificate signed by SK_{NDO}. If all individual certificates are valid, then the whole chain is valid.

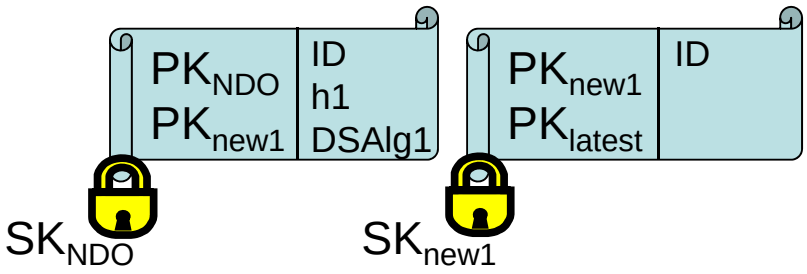

Figure 13: Certificate chain with two certificates, containing three owners in total

Both the basic and advanced approach technically allow all legitimate previous owners (i.e., all owners in the certificate chain in the advanced approach) to make valid changes to the NDO. If this behavior is undesired and former owners should be prevented from making changes to the NDO, then the advanced approach facilitates prohibition based on time stamps. To achieve this, each authorization certificate includes the production and expiry times. In addition, each object change is accompanied by a time stamp, based on a trusted time certification service (e.g., offered by the NRS during registration/unregistration). Invalid changes by previous owners can then be detected and can be prohibited (e.g., on a legal basis). This is simplified as there already exists a relationship between the different parties, i.e., the former owner previously authorized the new owner's PK/SK pair in the certificate chain.

Alternatively, a key revocation mechanism can be used to allow object receivers to validate if a change was performed by a valid owner. The same mechanism can also be used to handle key renovation and revocation of compromised secret keys.

3.4.3 *Owner Pseudonymity and Identification*

A distinctive feature of NetInf's naming framework is that owner pseudonymity is separated from data self-certification. This means that the PK/SK pair (PK_{owner}/SK_{owner}) used for owner pseudonymity is allowed to be different from the one used for data self-certification of an NDO (PK'_{NDO}/SK'_{NDO}), which itself is equal to PK_{NDO}/SK_{NDO} or is authorized by it.

Owner pseudonymity essentially binds the NDO's self-certified content to PK_{owner}. It can be achieved by including the $hash(\mathsf{PK}_{owner})$ in the self-certified data and by signing this data both by $\mathsf{SK}'_{\mathrm{NDO}}$ and by SK_{owner} (if $\mathsf{SK}_{owner} \neq \mathsf{SK}'_{\mathrm{NDO}}$). The signatures are included in the associated security metadata. The validity of the pseudonym can be verified by verifying the signatures. Only an entity that knows $\mathsf{SK}'_{\mathrm{NDO}}$ and SK_{owner} can change the pseudonym, which (by definition) is an *owner*.

Owner pseudonymity allows an owner to remain anonymous while allowing to build up trust in the owner as mentioned previously. This is done by reusing the same PK_{owner} for several NDOs. Thereby, the owner can build up trust in this PK_{owner} and, hence, in the content itself on the basis of the quality and trustworthiness of the previously published content. PK_{owner} becomes a kind of virtual identity of the owner, comparable to, e.g., an eBay user name that has a certain level of trust based on its history of transactions.

Owner identification essentially binds the NDO's self-certified content not only to PK_{owner}, but also to the corresponding real-world identity of the owner, e.g., the name of a person or a company. It can be achieved by including the real-world identity in self-certified data and by signing this data in the same way as for owner pseudonymity. In addition, the real-world identity needs to be verified, and this can be achieved by an additional signature binding PK_{owner} to this identity, i.e., the public-key certificate. This certificate is issued by a trusted third party upon verifying that the physical entity with this identity knows SK_{owner} and is included in the security metadata. Owner identification is then performed by verifying all these signatures.

The NetInf naming scheme allows several additional use cases. For example, the security features inherent to the NetInf naming scheme enable a secured name registration process of NDOs at the NRS. This can be very helpful to prevents replay attacks on the underlying registration servers. To perform such a replay attack, an attacker could record the communication during NDO registration between legitimate publishers and the NRS. Subsequently, the attacker could simply replay the transcripts of previous registrations, thereby overwhelming the NRS with many seemingly legitimate registration requests. To prevent such an attack, the NRS can request any publisher to include the current registration time in the registration request, signed by $\mathsf{SK}_{\mathrm{NDO}}$. I.e., in order to register/unregister an NDO, an owner needs to provide a (fresh) signature of the NDO's ID and the registration time, where the signature is verified by using $\mathsf{PK}_{\mathrm{NDO}}$ from the ID. Signing the registration time prevents the replay DoS attacks. The attacker would not be able to create a valid signature of the correct (i.e. updated) registration time as this requires knowledge of $\mathsf{SK}_{\mathrm{NDO}}$.

3.5 EVALUATION

We have built a Java-based NetInf prototype (Chapter 6) to evaluate and show the feasibility of the NetInf naming scheme. The naming scheme has proven to be easy to implement as it is based on several established security mechanisms like encryption and digital signatures that can be integrated via existing, proven libraries. Likewise, it is also easy to integrate and use the naming scheme in applications. We have built applications from scratch and have extended existing applications like the Thunderbird email client and the Firefox web browser with additional security functionality based on simple plugins. For example, our Firefox plugin enables the browser to interpret web pages that contain NetInf IDs instead of regular URLs as links. Thereby, users and content publisher can benefit from all NetInf naming scheme advantages right away by simply using the plugin on the client side and using NetInf IDs instead of (or in addition to, for backward compatibility) URLs as links in web pages. For example, the plugin gives users an additional icon indicating if the currently received web page is authentic or has been (maliciously) altered. Publishers, in addition, benefit from more flexible content management as a result of persistent IDs. More details about the prototype are described in Chapter 6. In general, the naming scheme has demonstrated to be able to add additional security properties – name-data integrity, owner pseudonymity, and owner identification – to a wide variety of applications ranging from information dissemination and information management to advanced, context-aware mobile applications.

3.6 NI URI SCHEME

To foster application development and simplify migration, we have started to standardize the main aspects of the NetInf naming scheme by developing the ni URI scheme [28] as described in Section 2.2.2. The ni URI scheme is a practical realization of the general NetInf naming scheme. It focuses on simplicity and, hence, currently only contains a subset of the features described in this chapter with the main focus on providing a consistent, generic way to provide name–data integrity for static data. The handling of dynamic data is described in a separate Internet draft [90].

In addition to the basic security features described in this chapter, the ni URI scheme offers some other features that focus more on practical implementation aspects. For example, ni URIs can contain an authority field to assist in accessing the named object for routing requests or assisting NRS scalability. Names may additionally contain a query string that can hold routing hints or other values used

in the NetInf protocol. Moreover, the ni URI scheme defines a binary form of this name format for use in more constrained environments.

Flat namespaces such as the NetInf namespace are generally considered human-unfriendly. In many cases, human-unfriendly names are not problematic as URIs do not have to be manually processed by humans but are often transferred via emails, hyperlinks, search results, QR codes, etc., and simply "clicked on" by users. To support manual processing of NetInf names by humans, the ni URI scheme also defines a human-friendly form in case it is required to "speak" a name, e.g., over the phone. To reduce the risk of communication errors in this error-prone verbal communication, the human-friendly form adds a checksum to allow for validation of name correctness. It also supports a hexadecimal representation of the hashes and hashes can be truncated to reduce the size of the names, which obviously involves a trade-off between security and name length.

In addition, there is a well-defined way to map these ni URIs to and from HTTP URLs, via the "well-known" URL scheme [91].

3.7 RELATED WORK

The basic idea that an ID contains an "object owner"-related part (hash of public key) that can be used for authentication and a "label" part that is under the control of "the owner" has been suggested in previous work, which includes the Simple Distributed Security Infrastructure (SDSI) [92] and DONA [12]. However, SDSI and DONA bind the public key directly to an entity called *principal*. Therefore, when the principal changes, the ID also changes both in SDSI and DONA, breaking the name persistence. In our naming scheme, we can keep the ID persistent even when the owner changes because the public key is bound to the NDO itself and only indirectly to the owner. There is another difference in the way that the owner information *is used* for serving cached copies in DONA and NetInf. In DONA, "... only hosts authorized by the principal P can offer to serve (i.e., provide access to) entities with IDs of the form P:L" [12], which limits the usability of available copies. In contrast, it is an important feature in NetInf to make use of any available data copy to improve efficient data dissemination.

The naming approaches of NetInf and CCN [14] differ essentially because CCN uses hierarchical names typically corresponding to organizational structures. This implies that name persistence with respect to owner or organizational changes is not given. The CCN security concept requires that the NDO's ID and the content be signed by an entity trusted by the users. If this entity is bound to the NDO's ID (e.g., the owner or any part of organizational structure), then the trust changes if the owner or organizational structure change. If not, then the trust becomes difficult to control. If this signing entity is different

from the owner, then signing of dynamic content may be a problem. Since the signing public keys are placed outside the ID, CCN IDs do not inherently support self-authenticated name registration and users typically cannot specify the trusted public keys beforehand when indicating an interest in data. Hence, users could be overwhelmed by fake data packets with the "right" ID, which cannot be filtered simply on the basis of their ID and content. This makes the system vulnerable to DoS attacks.

As mentioned earlier, PSIRP/PURSUIT uses rendezvous IDs to retrieve NDOs and scope IDs to restrict the distribution of objects. However, to our best knowledge, PSIRP/PURSUIT currently only focuses on name–data integrity via the IDs but not on owner pseudonymity, owner identification, and other security properties supported by the NetInf naming scheme.

For the special case of *static* content, NetInf can include the cryptographic hash of the underlying information in the ID itself. A similar idea has been proposed, e.g, in SFS [93]. However, SFS focuses on self-certifying pathnames that are location dependent. Likewise, the idea of certificate chains (although in a different context) has been used in other proposals like the Simple Public Key Infrastructure (SPKI) [94] and KeyNote [95].

The Handle System [96] offers a general mechanism to persistently identify digital objects and builds the basis for other systems like DOI [88]. However, the Handle system does not include security mechanisms like name–data integrity and owner pseudonymity in the naming scheme itself, which are required for an information-centric network architecture. In addition, the NetInf naming scheme differs from many of the above mentioned systems because of its flexibility to support various different ID structures via its ID type tag.

During recent years, it has become apparent that many of the problems that are haunting the Internet stem from the semantic overloading of the IP address. A lot of effort has been put into investigating how an ID/locator split can be instrumental in providing better support for mobility, multihoming, protection against DoS attacks, etc. Important work in this area includes the Host Identity Protocol (HIP) [97], the Internet Indirection Infrastructure (I3) [98], the NodeID architecture [99], and the Layered Naming Architecture [100]. Our naming scheme is also based on an ID/locator split, but has different properties as our focus is on supporting efficient data dissemination while at the same time satisfying the requirements of a secure information-centric network architecture.

In the Layered Naming Architecture, the authors state four principles which stress that names should not impose unnecessary restrictions by how they bind to underlying protocols or provide name persistence. To cater for these requirements and allow maximal flexibility, we use flat IDs which allow arbitrary recursions and indirections as

they are allowed to map onto themselves before they finally resolve to a locator. Flat IDs are instrumental for the NetInf naming scheme to provide several security properties. However, scalable name resolution for large flat namespaces is a challenge. Chapter 5 describes in detail how this problem is addressed in NetInf.

There are several systems that build their own mechanisms to check data integrity based on cryptographic hashes, including open source software package distribution and the BitTorrent protocol. Such systems can benefit from the NetInf naming scheme, eliminating the need to build their own mechanisms while providing additional security properties.

3.8 SUMMARY

Information-centric network architectures have an inherent need for a secure naming scheme. Because requested data can be delivered from any available untrusted network location that happens to have a copy of the data, security has to be based on the data and its ID itself and cannot be based on network nodes. There are some existing proposals for information-centric network architectures, including corresponding naming schemes. However, it seems that they are all missing some properties that we think are important.

The NetInf naming scheme simultaneously fulfills all our requirements based on the combination of a flexible ID structure and a securely attached set of metadata. In addition to security properties like name–data integrity and owner pseudonymity also provided by some other naming schemes, the NetInf naming scheme is characterized by its unique, non-trivial combination of security-related properties. This includes the flexibility to name a wide variety of entities, extensibility, persistent IDs under various changing conditions (especially owner change), secured name registration, and support for anonymous publication of information.

Our prototype evaluation (Chapter 6) shows that the NetInf naming scheme is feasible and provides a powerful foundation for a secure information-centric network architecture.

The ni URI scheme has generated significant interest and is currently under evaluation or in use by other IETF/IRTF research groups. This includes, e.g., the DECADE group [29]. We have published an Internet draft describing the "Requirements for accessing data in network storage" [30], aiming at the use of the ni URI scheme in DECADE. The PPSP group [31] also evaluates the ni URI scheme for its work and we have published an Internet draft about "Secure naming structure and p2p application interaction" [32] to describe the usage of the ni URI scheme for PPSP. The ni URI scheme is currently also evaluated for use in the CoAP protocol [33] of the CoRE group.

Details about the ni URI scheme features, about its URI syntax, the encoding, digest algorithm handling, etc. can be found in our RFC "Naming Things with Hashes" [28] and our Internet draft "The Named Information (ni) URI Scheme: Parameters" [90].

4

NEIGHBORHOOD EFFECT – LOCALITY IN DNS REQUESTS

This chapter is based on work published in reference [34].

Basing a new architecture on realistic assumptions, including request patterns that are validated via (recent) measurements, is important to ensure an architecture's feasibility. Hence, in this chapter, we perform a DNS traffic analysis to evaluate request patterns for data objects that have influence on the NetInf NRS performance. The results of this evaluation are a major input to the design of the general NRS framework described in the following Chapter 5.

4.1 INTRODUCTION

In this chapter, we evaluate the locality pattern of content requests, i.e., the correlation between the (network) location of the requester and the "location" of the requested content. We define "local content" as content that has a close *semantic* relationship to the requester's network, e.g., a company's web page or intranet content has a close semantic relationship to the company's network. We identify this relationship via the content's host name, i.e., a close relationship exists if the content is named using a domain name that is associated to the requester's network location. For example, *www.uni-paderborn.de/someContent* is associated to the campus network of the University of Paderborn. This definition makes no statement about *where* the content is hosted. In today's Internet architecture, "local content" is in fact not necessarily hosted locally. In the course of this chapter, we argue that hosting local content locally has several benefits. This does not preclude content being hosted in multiple networks nor does it imply that this content is then always local in all networks.

We call the interest of users in local content *neighborhood effect*, i.e., users show strong interest in content from their (semantical) neighborhood. Note that the neighborhood effect differs from *request pattern similarity* of homogeneous groups, i.e., the effect that homogeneous user groups tend to share a common interest in *similar* content. Request pattern similarity lacks the locality aspect. We see an important influence of the neighborhood effect on several aspects of network architectures, especially on *name resolution*, *name-based rout-*

ing, and *caching*. A large neighborhood effect has the potential to increase an architecture's scalability and reduce latency, overall network traffic, and costly inter-domain traffic. This positive influence stems from the potential to keep a significant amount of data traffic and/or resolution requests within the local network vicinity.

Obviously, our main focus is on future ICN architectures in general and on NetInf specifically. Here, we expect a strong influence of the neighborhood effect as all three components – name resolution, name-based routing, and caching – play a major role in ICN. However, the results presented in this chapter also have influence in other areas such as the general design of future Internet architectures.

We evaluate the neighborhood effect based on DNS request patterns. Several DNS evaluations exist already. Most evaluations focus either on the DNS top level or on the popularity distribution of the requested hosts, e.g., to evaluate cachability. However, we are not aware of any recent DNS evaluation that focuses on the neighborhood effect. Hence, in this chapter, we evaluate this effect in detail.

This evaluation might appear trivial at first glance and a strong neighborhood effect might be expected based on gut feeling. However, due to the lack of quantitative results, we believe that this kind of evaluation is essential as a basis for future ICN architectures.

Our DNS evaluation is based on two independent measurements that we have performed at the internal DNS servers of the University of Paderborn during 2009, 2010, and 2011, covering almost four months of DNS traffic in sum and containing more than 2.5 billion DNS requests. Each measurement covers a distinct set of DNS servers: the DNS servers responsible for the main university domains and the separate DNS servers of the computer science subdomain.

We focus on the following research questions:

- What are the characteristics and the magnitude of the neighborhood effect?
- What is the influence of requests originating directly from user devices and originating from servers (e.g., mail and web server) on the overall request patterns?
- What is the influence of the sub-zone relationship between the two evaluated DNS zones?

By investigating the influence of the sub-zone relationship, we want to evaluate if the neighborhood effect is visible at different "levels", which can be exploited by our *hierarchical* NRS framework described in Chapter 5.

In Section 4.2, we discuss the measurement setup. In Section 4.3, we evaluate the measurement results for both DNS zones. Section 4.4 discusses related work before we discuss the consequences of our results for future Internet architectures with a focus on ICN architectures in Section 4.5.

4.2 MEASUREMENT SETUP

The first data set, called *university* (*Uni*) DNS zone, includes all authoritative DNS servers for our university's principal DNS zone, including the domains *uni-paderborn.de*, its alias *upb.de*, as well as some department-specific second-level domains. The data set contains the full DNS traffic of an 11 weeks period between December 2009 and February 2010, including more than 2.5 billion DNS requests.

The second data set, the *computer science* (called *IRB*[1]) DNS zone, has been captured at the authoritative DNS servers of the computer science department. This data set mainly contains the subdomain *cs.uni-paderborn.de* and its alias *cs.upb.de*. The data set has been captured during June/July 2011 for a four weeks period, containing about 39 million DNS requests.

All data has been collected using a syslog-ng logging server and some custom python scripts to perform data anonymization prior to logging to protect the users' privacy. Figure 14 illustrates the logging results using an example of the IRB log file. The third and fourth columns describe the requester. All requests that originate from a *computer science department-internal server* are marked `irb_int` and its IP address is shown in plain text in the fourth column. All requests originating from *user devices* (also simply referred to as *users* in the following) *within the university network* are marked `upb_int` and requests from *outside the university network* are marked `extern`. In both cases, the requester's IP address is removed and substituted with a random number for data privacy. To further protect the users' privacy, these client numbers are reset every 24 hours, i.e., a requester gets a new random number every 24 hours. This allows us to identify short-term patterns, e.g., redundant/duplicated requests by the same client, while preventing us from identifying long-term personal request patterns[2].

Columns five and six identify the requested host name. Similar to above, hosts with an IP address internal to the computer science department are marked `irb_int`, requests for university-internal domains `upb_int`, and all other host names `extern`. All host names are hashed and stored in column six. Again, the hashing is performed for privacy reasons, yet it still allows us to extract valuable information concerning request patterns. Finally, columns 7–9 contain more details about the type of the query, e.g., address record lookup (`A`), reverse lookup (`PTR`), or service lookup (`SRV`).

1 IRB is the abbreviation for the network's German name "Informatik Rechner Betreuung".

2 The measures to protect the users' privacy have been performed in coordination with the local data protection officer.

```
---Date---------Time------Req.Type--Req.no./IP----HostType-Hash---Info--
30-May-2011 18:05:41.023; irb_int 131.234.24.147; irb_int 40... IN PTR +
30-May-2011 18:05:41.149; upb_int 74070213;       irb_int de... IN SRV +
30-May-2011 18:05:41.300; upb_int 41683357;       upb_int 2b... IN A   +
30-May-2011 18:05:51.328; irb_int 131.234.24.148; irb_int 40... IN A   +
30-May-2011 18:05:51.872; upb_int 63098679;       extern  86... IN A   +
30-May-2011 18:05:49.630; extern  96311156;       upb_int 40... IN A   -
```

Figure 14: DNS log example: computer science DNS zone (full hash values elided in figure for space reasons)

4.3 DATA EVALUATION

In the following, we discuss the data preprocessing that we have performed for all measurement data, followed by the result evaluation of the university DNS zone and the computer science department DNS zone.

4.3.1 *Data Preprocessing*

To gain a better insight into the request patterns, we have eliminated side effects in the logging data as much as possible as described subsequently.

Our main interest with this evaluation is to analyze the neighborhood effect of DNS requests. Hence, we are interested in the locality properties of requests by requesters *within the university network*. Therefore, we have filtered requests from clients external to the university network. As we are interested in the general request patterns during regular operations, we have also filtered requests that resulted from irregular situations, e.g., generated by a temporary university-internal Planetlab experiment that generated many requests. Likewise, we have filtered requests from a few obviously misconfigured clients posing requests with duplicated domain names like `hostname.upb.de.upb.de`. We have also filtered redundant requests resulting, e.g., from clients first requesting the IPv6 address and subsequently requesting the IPv4 address for the same host name within 1 s as this represents a temporary special situation due to the IPv4–IPv6 transition.

Finally, we have eliminated reverse lookups as our focus is on the lookups of host names to IP addresses. These are the lookups that correspond to, e.g., object name resolution in an ICN.

We evaluate the filtered DNS requests separately in the following sections. All other figures exclude the aforementioned DNS requests unless explicitly stated. As we are not interested in fluctuations over the course of the day in this evaluation, all values are averaged over 24 hours.

In Section 4.3.2, we first evaluate the data of the university-wide DNS servers. Subsequently, Section 4.3.3 evaluates the results of the computer science DNS servers.

4.3.2 *University DNS Zone*

This section is subdivided into the evaluation of the overall DNS requests (Section 4.3.2.1), the analysis of requests by university-internal servers (Section 4.3.2.2) such as web servers, mail servers, etc., the analysis of requests originating from user devices (Section 4.3.2.3), an analysis of the influence of our data preprocessing on these results (Section 4.3.2.4), and a summary of the results (Section 4.3.2.5).

4.3.2.1 *All Requests (Filtered)*

Figure 15 shows all DNS requests (excluding the filtered requests as previously described) received by the authoritative DNS servers for the university DNS zone, separated into requests for *internal domains* (i.e., university-internal hosts) and *external domains* (i.e., university-external hosts). Figure 15 reveals that the university DNS servers receive significantly more requests for internal host names (10312 req./min) than for external host names (4227 req./min). Hence, approximately 71% of the overall requests are for internal host names.

The requests in Figure 15 show a weekly pattern for both internal and external host names. Note that the weekly pattern differs between December 23rd and January 3rd. This is the Christmas holiday season where the request pattern roughly equals the pattern during weekends. This is even more visible in Figure 19, which only shows requests by user devices, i.e., excluding requests by servers.

The first and last measurement values, which show unusually low request values, are due to the fact that both the first and last measurement period does not include a full 24 hours period. The unusually high number of DNS requests between the 25th and 29th of January is due to an unusually high number of DNS requests by the mail server for university-internal host names, as can be seen in Figure 18. The reason for this peak is unknown but is likely due to unusual mail traffic patterns such as a burst of spam or a configuration issue.

Next, we analyse the ratio between *user requests* (i.e., requests originating directly from user devices, including employee's and students) and *server requests*. Figure 16 shows that the overall requests are dominated by server requests for internal hosts (8819 req./min), followed by user requests for external hosts (2410 req./min), server requests for external hosts (1817 req./min), and user requests for internal hosts (1493 req./min). In sum, server requests dominate the overall DNS

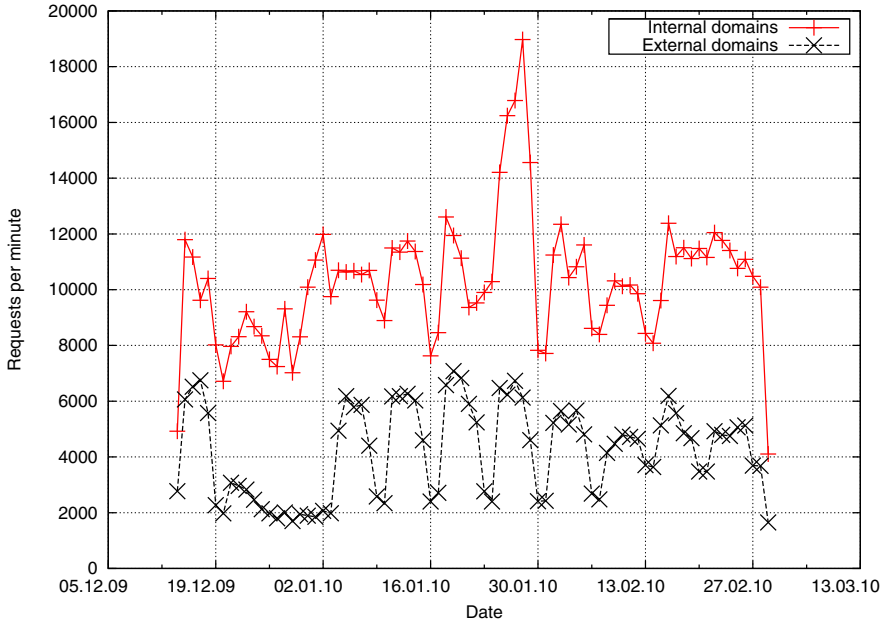

Figure 15: Uni: All requests, *filtered*

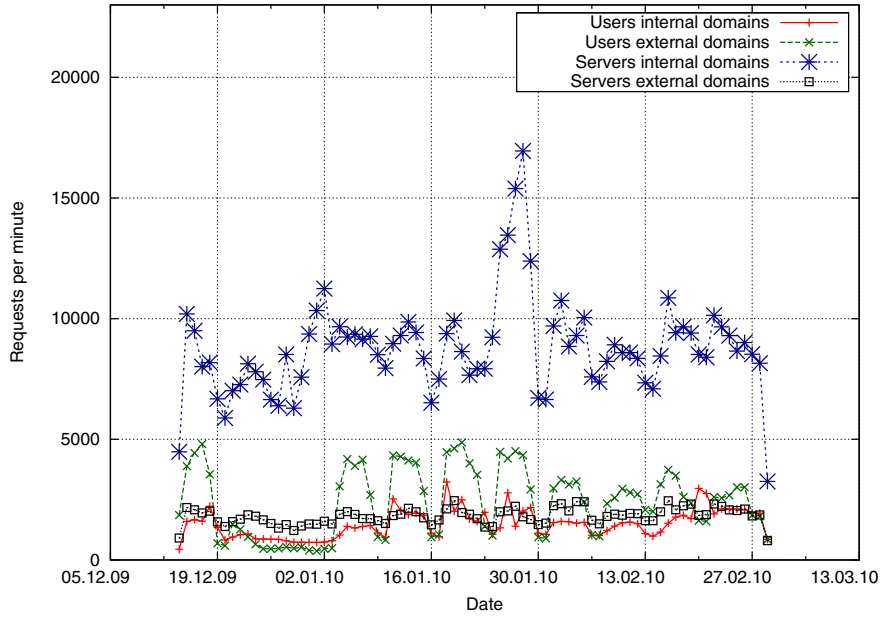

Figure 16: Uni: DNS requests by user devices and servers

requests (61%). In the following two sections, we evaluate server requests and user requests separately in more detail.

4.3.2.2 *Server Requests*

Let us start by evaluating the server requests as servers generate the majority of requests. Figure 16 already illustrated that there are significantly more server requests for internal hosts than for external hosts. Next, we are interested in the distribution of generated requests among different server types. The next two figures show the server DNS requests separated by server type for external (Figure 17) and internal (Figure 18) host names.

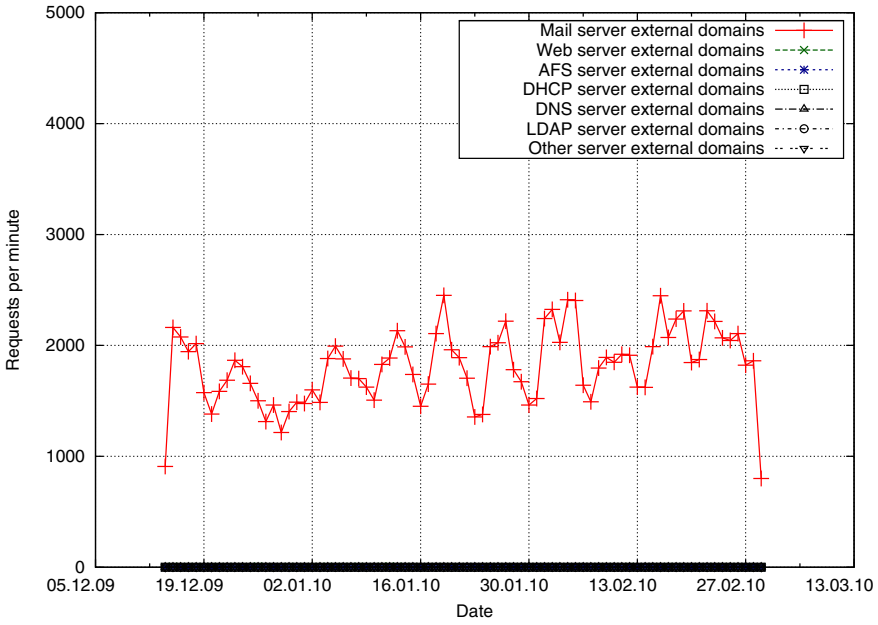

Figure 17: Uni: DNS requests by servers for external host names

Both figures reveal that the university mail servers generate the vast majority of server requests, both for external (1811 req./min) and internal (8069 req./min) host names with approximately 82% for internal hosts. No other servers generate any significant amount of DNS requests.

Our investigation revealed that the major portion of mail server DNS requests is caused by the specific configuration of the mail servers. The mail servers are configured to start a new process for each incoming connection. This process first loads its configuration file that contains several host names that each have to be resolved, causing multiple DNS requests for each connection. These requests are mainly for internal hosts. The configuration file contains information such as the internal Lightweight Directory Access Protocol (LDAP) server, etc. This effect illustrates that DNS traffic can highly depend

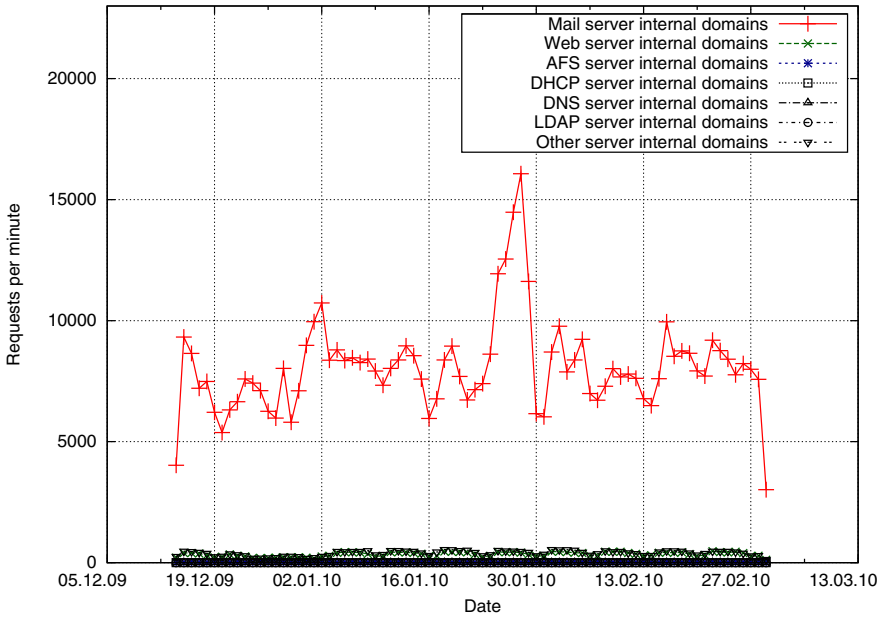

Figure 18: Uni: DNS requests by servers for internal host names

on specific configuration settings and the choice of additional services. DNS traffic analysis by Zdrnja et al. [101] has shown related effects. They analyzed DNS responses with a focus on security aspects like botnets and spam. As a result, they identified that a large amount of their DNS responses was caused by the university-internal anti-spam software. For each received email, the anti-spam software queries several real-time black lists (using address record DNS requests) to determine if an email sender is likely a spammer.

Another part of the mail server requests is generated by the mail transfer agent – more precisely, the client part of the Simple Mail Transfer Protocol (SMTP) – that checks DNS mail exchanger (MX) records to figure out the destination mail server. The emails producing these requests are generated exclusively by university users (as the SMTP servers require authentication). Note that while the majority of university mail users resides within the university network, some users also use the SMTP servers from outside the university network.

4.3.2.3 *User Requests*

Next, we have a look at the requests generated by user devices. Figure 19 shows all user requests, again separated into requests for internal (1493 req./min) and external (2410 req./min) host names. Approximately 38% of the requests are for internal host names.

Figure 19 shows an interesting weekly pattern: The overall requests are significantly higher during weekdays, as would be expected. However, it is interesting to note that the number of requests for external domains is much higher than for internal domains dur-

ing *weekdays*, resulting in a ratio of roughly 1:2 between requests for internal and external host names. During *weekends*, both types of requests are roughly equivalent.

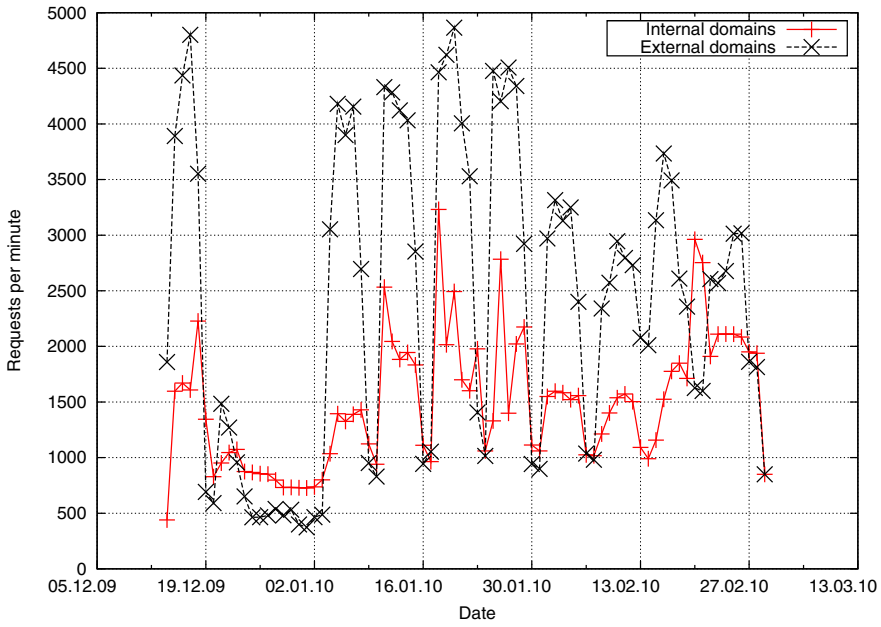

Figure 19: Uni: DNS requests by user devices

4.3.2.4 *Influence of Filtering*

Next, we analyze if the data preprocessing as described previously has some major influence on our results. Figure 20 shows the same graphs as Figure 15; however, Figure 20 includes all requests by university-internal requesters without filtering, i.e., duplicated requests, requests from misconfigured clients, reverse lookups, etc. are included (but not requests by requesters outside the university). Obviously, there are more requests for both internal (13546 req./min) and external (6594 req./min) hosts. However, the overall structure as well as the ratio between internal and external hosts remains roughly the same with 67% requests for internal hosts.

Figure 21 shows only reverse lookup requests, illustrating their impact on the overall internal-to-external ratio. As can be seen, requests for internal hosts (3214 req./min) also dominate the overall requests (4540 req./min) with 71% for internal hosts. Hence, including reverse lookups would not change the overall ratio between requests for internal and external host names. Adding the number of reverse lookups for internal hosts (3214 req./min) to the number of requests for internal hosts in the *filtered* Figure 20 (10312 req./min) adds up to 13526 req./min, which almost equals the number of *unfiltered* requests for internal hosts in Figure 20 (13546 req./min). Hence, almost all re-

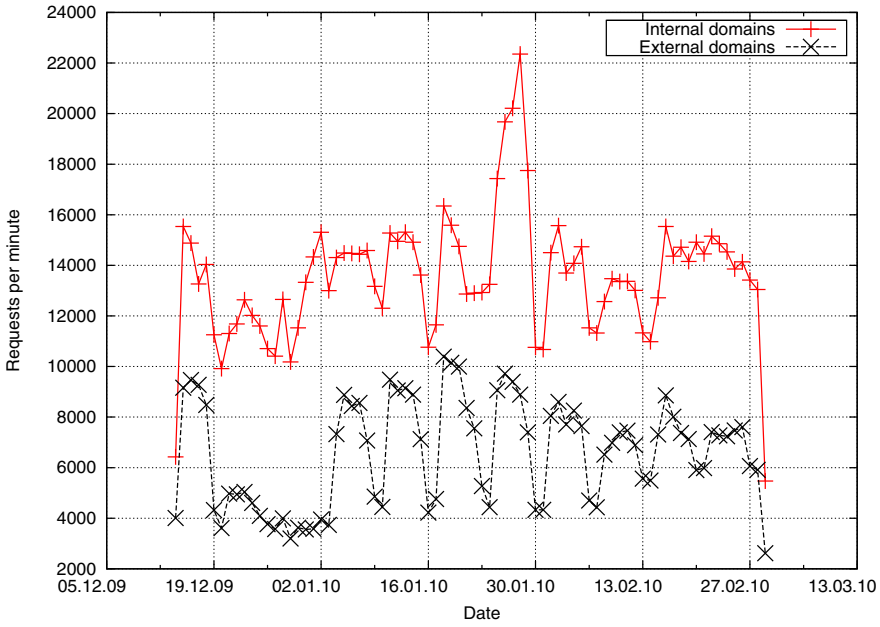

Figure 20: Uni: All requests, *unfiltered*

quests for internal hosts eliminated by filtering are reverse lookups. The requests for external hosts eliminated by filtering are also dominated by reverse lookups but include some additional effects as described in Section 4.3.1.

Figure 22 shows all reverse lookups separated by requester type. Most reverse lookups are generated by user devices, followed by mail server reverse lookups. No other server generates a significant amount of reverse lookups.

Finally, Figure 23 shows all requests (also excluding reverse lookups, erroneous requests, etc.) by requesters outside the university network. They contribute only 11% of the overall requests. Their requests for university-internal hosts constitute 44% compared to requests for external hosts (56%).

4.3.2.5 *Summary of University-wide Results*

In summary, we can conclude that about 70% of the overall DNS requests at the university level are for internal host names. The number of requests by servers outweighs the number of client requests, with the mail servers generating most of the server requests. For both the user and server requests, the number of requests for internal host names is significant with approximately 40% for user devices and even 80% for servers.

Reverse lookups dominate the overall requests that we filtered out. We filtered reverse lookups as they are not relevant for the NetInf

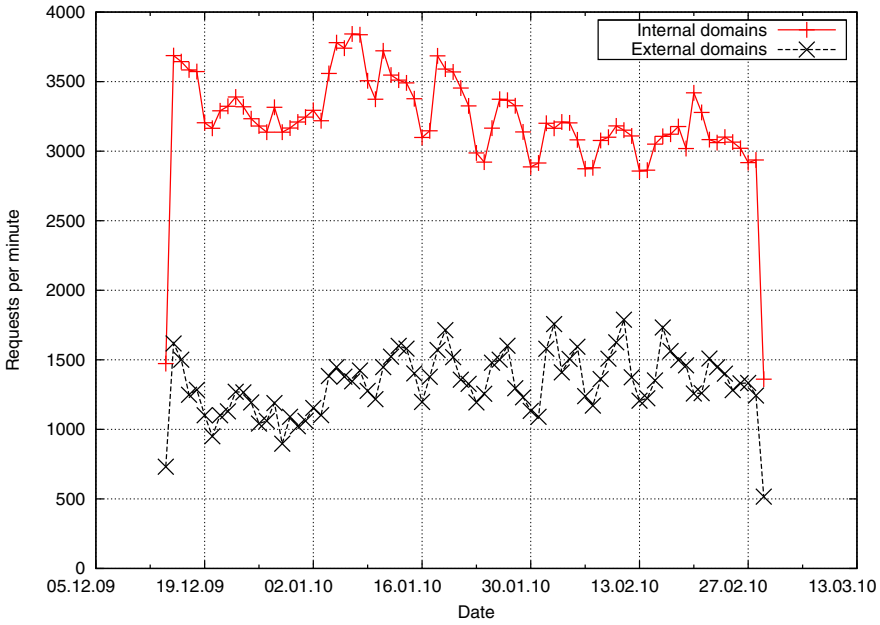

Figure 21: Uni: Only reverse lookup requests

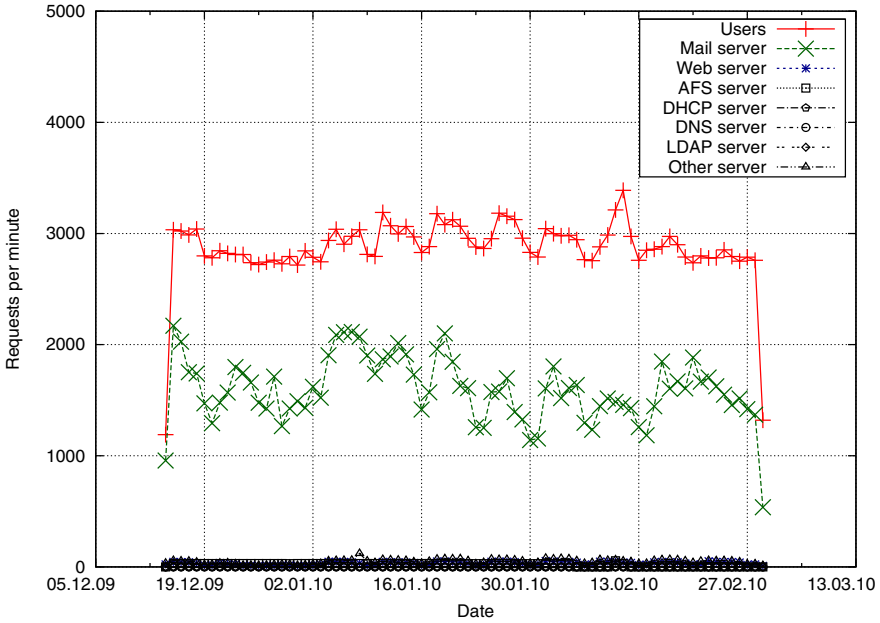

Figure 22: Uni: Reverse lookups by requester type

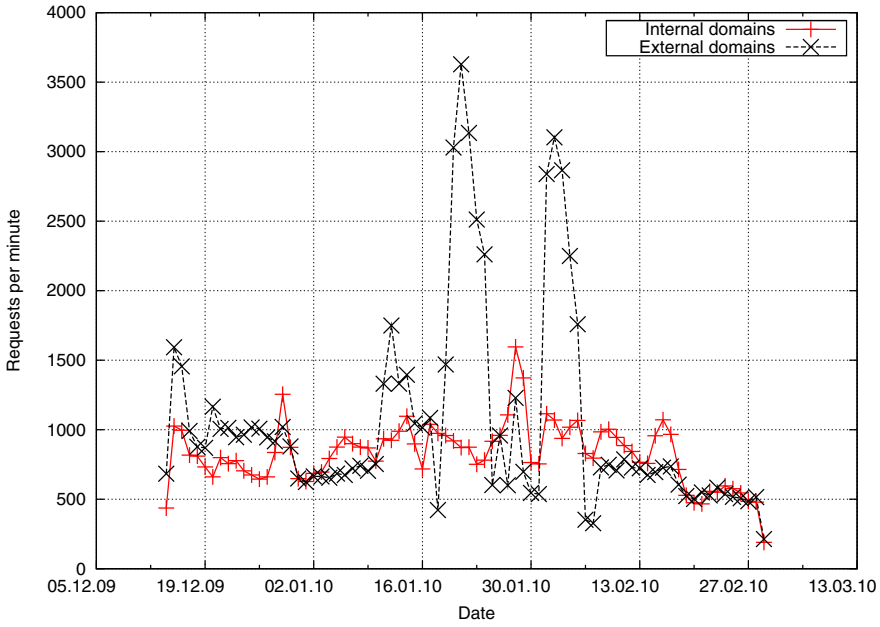

Figure 23: Uni: DNS requests by university-external requesters

architecture. In any case, the reverse lookups are also dominated by lookups for internal hosts. Hence, including reverse lookups in our results would not change the characteristics of the results (Figure 20 and 21).

Client requests show a distinct weekly request pattern with significantly more overall requests during weekdays. Not surprisingly, a similar but less pronounced pattern can be observed in the server requests. More interestingly, the *type* of the user-requested content seems to change between weekdays and weekends as observed in the user's request patterns.

4.3.3 *Computer Science Department DNS Zone*

In the following, we evaluate the DNS requests within the computer science department (*IRB*). We take the sub-zone relationship between university-wide DNS servers and IRB department into account by separating the DNS requests into three subgroups: *IRB-internal domains* (all requests for hosts of the computer science sub-zone), *UPB-internal domains* (all requests for hosts of the university-wide zone, *excluding* the IRB sub-zone), and *external domains* (all requests for hosts outside the university zone).

4.3.3.1 *All Requests (Filtered)*

Figure 24 shows all DNS requests received at the authoritative IRB DNS servers (filtered as described in Section 4.3.1). In total, the IRB DNS servers get much fewer requests compared to the university-

wide DNS servers. This is not surprising as the computer science department is just a single sub-zone of the overall university and many IRB users use the university-wide DNS servers instead of the department-internal DNS servers. Figure 24 shows that approximately 48% of the requests at the IRB DNS servers are for university-external hosts (337 req./min), 40% for IRB-internal hosts (280 req./min), and 12% for university-internal hosts (86 req./min).

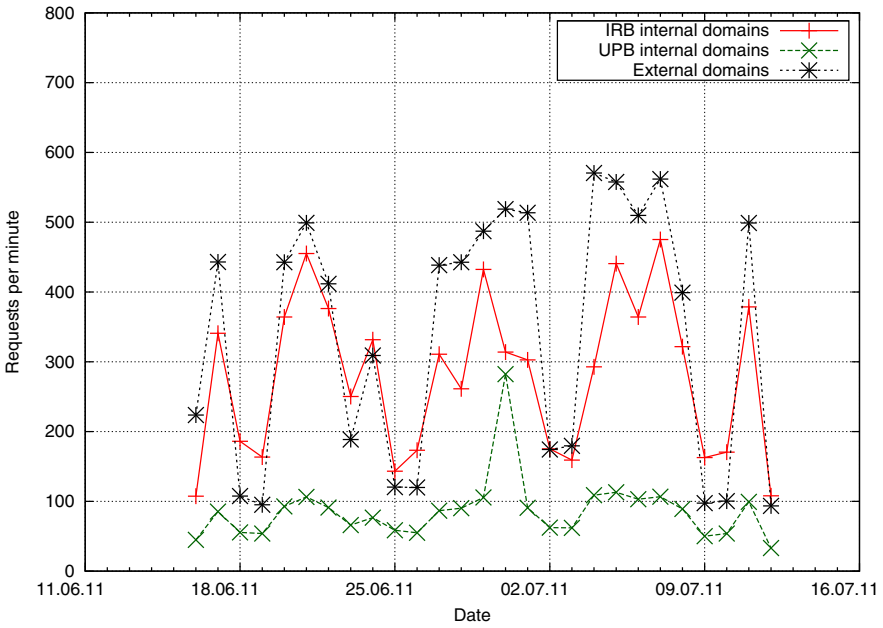

Figure 24: IRB: All DNS requests, *filtered*

4.3.3.2 *Influence of Filtering*

Figure 25 shows the same graphs as Figure 24. However, it contains all additional requests that are filtered in Figure 24, but still excludes requests by university-external requesters. The unfiltered results are very similar to the filtered results. The main difference is an increase in the number of university-internal requests by 192 req./min, which results primarily from reverse lookups (168 req./min) as shown in Figure 26.

4.3.3.3 *Client and Server Requests*

Figure 27 shows all requests by user devices. These are dominated by requests for external hosts (57%). Adding up the IRB-internal requests (32%) and university-internal requests (11%) results in 43% requests for hosts within the overall university network.

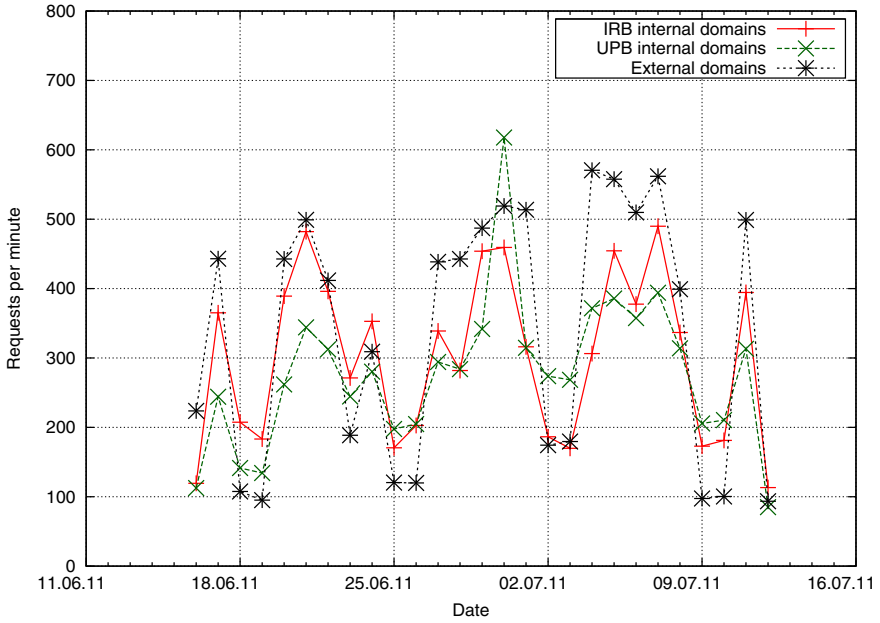

Figure 25: IRB: All DNS requests, *unfiltered*

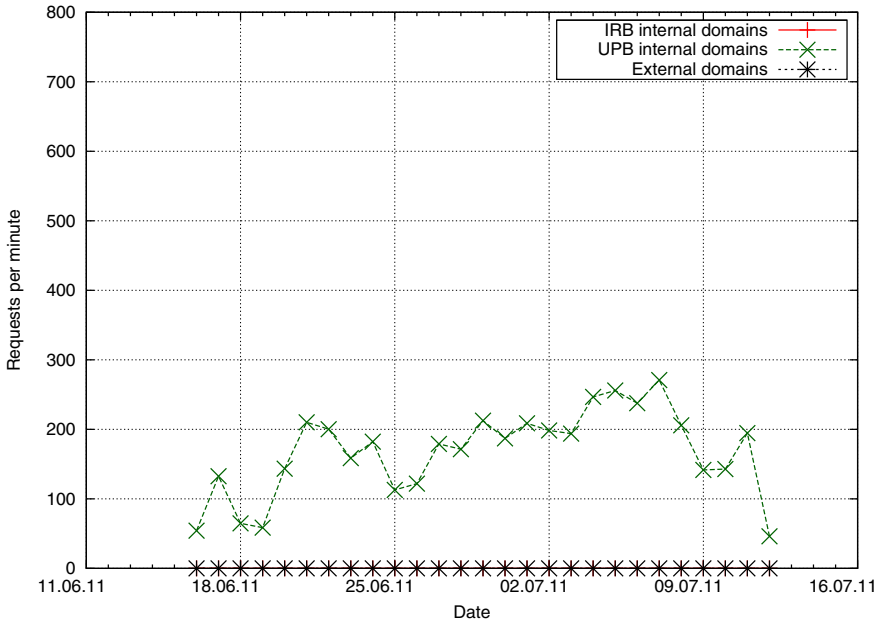

Figure 26: IRB: Reverse lookups

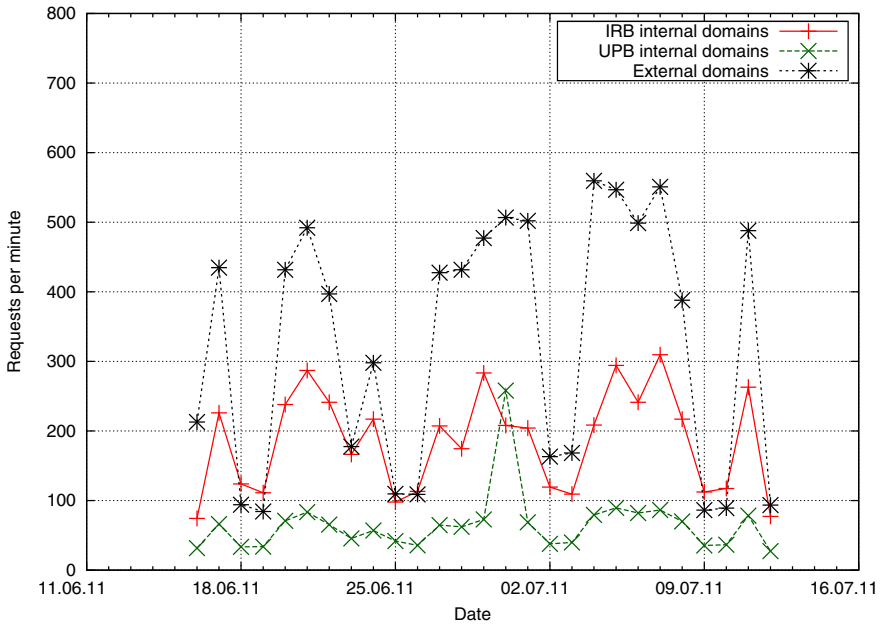

Figure 27: IRB: DNS requests by user devices

Figure 28 shows the same for IRB server requests. The overall number of server requests is much smaller compared to user requests (18% of the overall requests) and is strongly dominated by requests for IRB-internal hosts (75%). The server requests for university-internal and IRB-internal hosts are mainly generated by the IRB LDAP servers as can be seen in Figure 29. The file servers are generating all requests for external hosts (11 req./min).

4.3.3.4 *Summary of Computer Science Zone Results*

In summary, the computer science DNS servers show similar results compared to the university-wide results with a large number of DNS requests for (computer science) internal hosts (40%). However, this number is smaller than the university-wide 71% for internal hosts. The total number of requests is dominated by requests from user devices (82%).

Our evaluation of the sub-zone relationship between the university-wide zone and the computer science zone revealed that a total of 12% of the DNS requests at the computer science DNS servers are for hosts within the higher-level zone (i.e., the university-wide zone). Hence, a total of 52% (40% for the computer science zone + 12% for the university-wide zone) of the requests are for hosts within the overall university zone. This can be relevant from a network perspective as both zones belong to the same physical campus network.

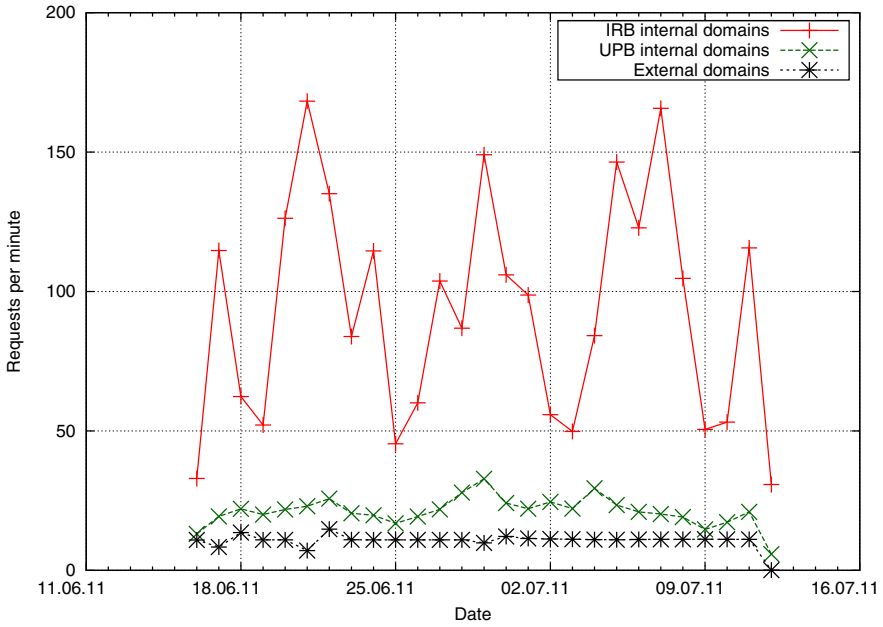

Figure 28: IRB: DNS requests by IRB servers

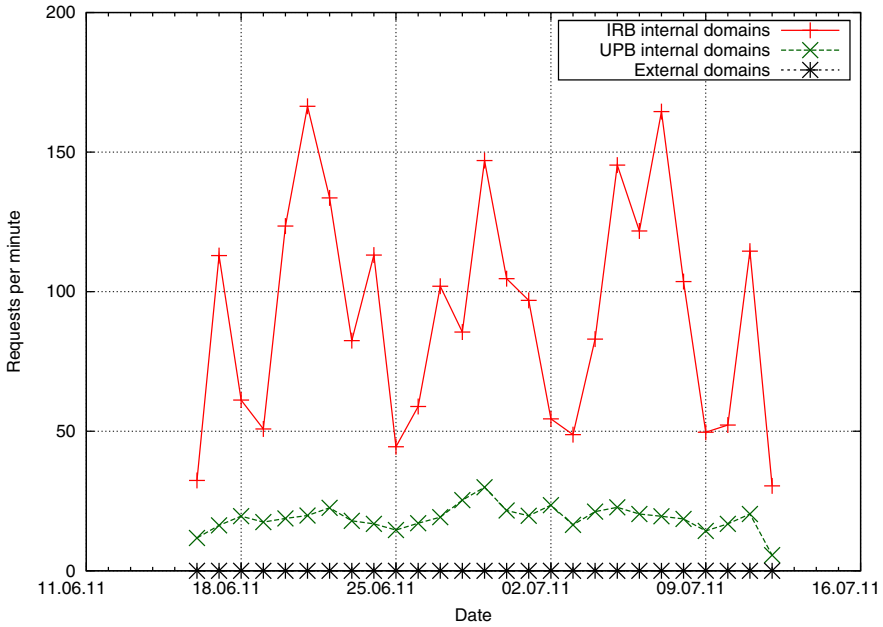

Figure 29: IRB: DNS requests by IRB LDAP servers

4.4 RELATED WORK

Multiple research activities have performed DNS traffic measurements and have focused on different aspects. Several have focused on DNS measurements at the DNS root servers. For example, the Réseaux IP Européens Network Coordination Centre (RIPE NCC) DNS Monitoring Services (DNSMON) [102] provides current monitoring data about several DNS root servers via test requests sent to these servers. The focus is on responsiveness and request latency. Likewise, Castro et al. have evaluated several aspects of the DNS root servers, including traffic growth and usage patterns [103] and general workload at the root servers [104], both based on data collected by the Cooperative Association for Internet Data Analysis (CAIDA). However, traffic analysis at the DNS root servers does not provide us with information about request locality.

To analyze request locality, measurements at lower DNS levels are required. Such measurements have been performed, e.g., by Jung et al. [105] and by Ager et al. [106], however, without analyzing request locality. Jung et al. have analyzed the performance and prevalence of failures and the effectiveness of DNS caching. Ager et al. have focused on data cacheability for the protocols HTTP, BitTorrent, eDonkey, and Network News Transfer Protocol (NNTP). In both cases, the similarity and popularity distribution of DNS requests is of main importance. Similar evaluations have been performed in other areas like P2P networking with the goal to reduce inter-domain traffic by exploiting the request popularity distribution for caching [107]. Although the request popularity distribution and the locality of DNS requests can be related, they are two separate things. For example, although all users of an organization might exclusively query the same 100 hosts, these hosts might all be internal or external hosts or a mixture of both. The latter is the main aspect that we focus on here. Obviously, the request popularity distribution is also relevant for caching in ICN. However, evaluating the neighborhood effect separately is useful as the neighborhood effect can have separate consequences on the ICN architecture, e.g., on server placement. When the neighborhood effect is high, placing servers that host local content in the local network is beneficial as it reduces inter-domain traffic without requiring caching.

We are not aware of any DNS analysis that mainly focuses on request locality as we do. However, a study by McDaniel and Jamin [108] that aimed at developing a secure public-key distribution system has generated some interesting results about DNS request locality as a side effect. To gather test data for their architecture, McDaniel and Jamin have performed DNS measurements at five different networks, four of which include information about request locality. Although their focus is on other aspects and, therefore, their results do not

provide more details about the neighborhood effects, the results do support our main findings in terms of general request locality. Like our results, their results indicate that a large percentage of DNS requests is for local hosts. This effect does not seem to be limited to organizations like universities but is also visible in other networks like the AT&T network. Specifically, they found the following locality results: Their AT&T trace contains about 50% requests for local hosts. The University of Michigan (UoM) trace contains about 65% local requests, the College of Engineering (subdomain of UoM with separate DNS servers) contains 38% local requests, and the Electrical Engineering and Computer Science Department (subdomain of UoM with separate DNS servers) contains 43% local requests.

Likewise, also as a side effect, a study by Ager et al. [1] that focuses on the identification and classification of content hosting and delivery infrastructures generated some interesting results concerning the neighborhood effect at the continent level. Specifically, they performed world-wide DNS measurements for the top 2000 most popular host names according to Alexa[3] and evaluated the percentage of requests that could be answered from within the same continent (including cached content) that the request originated from. Their results shown in Table 2 reveal that, e.g., 58.2% of all requests from North America have been served from within North America. Of course, this high number is partly due to the fact that most of the popular content is hosted in North America. For a comparison, "only" 10.1% of requests originating from South America have been served from within South America. However, these numbers are more relevant when compared to the percent of requests served from a certain continent on average. For example, on average 50% of requests from each continent have been served from North America. In comparison, only about 2.4% of requests from each continent have been served from South America. When we compare these average numbers with the percent of requests that can be served from within the same continent where the requests are generated, we see higher interest in locally available content than average for all continents (except Africa). For example, in North America, the interest for content served from within the continent is 8.2 percent points higher than average ($58.2\% - 50\%$), for South America, it is 7.7 percent points higher ($10.1\% - 2.4\%$). Please note that these results are not quantitatively comparable to the results of our evaluation in this chapter. First of all, the study by Ager et al. does not evaluate the *semantic* relationship between the requesters' location and the requested content as we do in our study (based on the requested URLs). Hence, we can only take the hosting locality as an *indication* that there is some semantic relationship. Moreover, we have to be careful as the study by Ager et al. includes locally cached content, e.g., via CDNs. Never-

3 http://alexa.com/topsites/

theless, the results indicate that the neighborhood effect also exists at higher levels like the continental level. More detailed studies would be required to prove this assumption.

Requested from	Served from					
	Africa	Asia	Europe	N. America	Oceania	S. America
Africa	0.3	18.6	32.0	46.7	0.3	0.8
Asia	0.3	26.0	20.7	49.8	0.3	0.8
Europe	0.3	18.6	32.2	46.6	0.2	0.8
N. America	0.3	18.6	20.7	58.2	0.2	0.8
Oceania	0.3	20.8	20.5	49.2	5.9	0.8
S. America	0.2	18.7	20.6	49.3	0.2	10.1

Table 2: Content matrix for top 2000 most popular host names. Each line provides the percentage of all requests that originate from a given continent. Columns indicate the continent from where content is served (see Table 1 in reference [1]).

4.5 SUMMARY

Our DNS traffic analysis has shown a significant percentage of requests for internal hosts. At the university-wide DNS servers, 71% of the requests are for local hosts. At the computer science department, 41% of the requests are for department-internal hosts and 52% for university-internal hosts. These requests contain requests generated by user devices and by servers. When removing the server requests, which are only indirectly generated by users, we still can observe approximately 42% requests for university-internal hosts directly originating from user devices.

We assume that similar neighborhood effects can be observed in other academic, enterprise, and Internet service provider (ISP) networks. These assumptions are substantiated by results in related work (Section 4.4) and by internal evaluations of partner ISPs. We also assume that similar effects can be observed at higher levels, e.g., at a country-wide and continental level due to cultural, linguistic, and political effects, as outlined in Section 4.4. In general, this effect varies depending on the specific context. For example, some ISPs in developing countries might not provide much local content to their customers, resulting in a low neighborhood effect. In contrast, ISPs in countries with strong cultural, linguistic, or political boundaries might observe a much higher neighborhood effect. It is interesting to note that these locality patterns seem to persist for at least 13 years as evidenced by McDaniel and Jamin [108], although the Internet usage patterns have changed dramatically during that time.

In today's Internet, the neighborhood effect has a positive influence on DNS. It reduces the average resolution latency as resolution can often be performed at the local DNS servers. This effect also helps DNS scalability as most requests do not reach the DNS top level.

In this chapter, we are particularly interested in the consequences of the neighborhood effect on future Internet architectures, especially on ICNs. We note that DNS requests are for hosts whereas ICN requests are for named data objects. However, every request for a data object in the ICN context would translate into a request for a host in the DNS context. Hence, the results are also applicable to the ICN context. Our data filtering is tailored for the ICN context. Specifically, we have filtered reverse lookups as it is currently unclear if they will have an equivalent in the ICN context. Also, it remains to be seen how servers will work in an ICN context. We assume, however, that an information-centric server will produce equivalent name resolution requests in an ICN context as in today's DNS context.

We see three main areas in future Internet architectures and ICN architectures specifically where the neighborhood effect will likely have a strong positive influence: *Name resolution*, *name-based routing*, and *caching*.

NAME RESOLUTION: Some ICN architectures like NetInf rely on a (globally) scalable NRS. Scalability is extremely important because the number of data objects that will be registered in an ICN NRS is expected to be much larger that today's number of DNS domains. Below, we propose some design principles that help exploiting the neighborhood effect to improve the scalability of the NRS and decrease the network load and latency.

Hierarchical structure: A hierarchical structure with multiple levels mapping the underlying local zones helps to exploit the neighborhood effect at multiple levels.

Registration scheme: Local data should be registered in the local resolution domain (i.e., local NRS) to benefit from the high local popularity of the local content. To make content available for a wider audience and exploit locality at higher levels, also register the content in the higher resolution domains.

Resolution scheme: In accordance with the registration scheme, users should first query the local resolution domain before iteratively querying higher-level domains. Thereby, they will retrieve the closest available copy, reducing latency, global network traffic, and inter-domain traffic. This scheme also prevents resolution requests from propagating farther than necessary. Consequently, most requests will be answered at lower levels, reducing the load at the critical global level, hence, improving scalability.

The general NRS framework and its two implementations MDHT and HSkip presented in Chapter 5 follow these design principles. In

Chapter 5, we also present a quantitative analysis and simulation results of the neighborhood effect influence on NRS latency and load distribution among NRS nodes.

NAME-BASED ROUTING: Several ICN architectures like CCN and NetInf (in addition to name resolution) use a name-based routing scheme. To exploit the neighborhood effect with name-based routing, the routing scheme has to ensure that traffic for local content stays within the local network. Given this requirement, a name-based routing scheme will benefit from the neighborhood effect via reduced global network traffic and reduced costs for inter-domain traffic.

CACHING: Due to the high popularity of local content, caching of local content will be efficient, i.e., high cache hit rates can be expected, which can be used to improve the information dissemination locally (as more sources are available) and to reduce the load at the original server. This effect is also demonstrated in our prototype measurements in Chapter 6. In addition, we expect that the extensive use of caching in ICN architectures will even further increase the magnitude of the neighborhood effect compared to today's host-centric Internet. Due to in-network caches placed in local provider networks, copies of popular, non-local content will be cached in the local provider networks. As these locally cached copies will also be registered at the local NRS in architectures like NetInf, requests for these cached copies will further increase the share of requests for locally registered content. To reduce the *data traffic* in all preceding cases, it is required to host a copy of the local content in the local network, either the original source or a cached copy.

This evaluation can just be a first step to evaluate the neighborhood effect. Although related work seems to support our assumption that the neighborhood effect exists at multiple levels, more evaluation results are needed to validate this assumption. For example, the neighborhood effect should be evaluated at the regional, country, and continent level, and in other types of networks, including enterprise networks and ISP networks. Also, the influence of mobile users on the neighborhood effect, investigated, e.g., via the neighborhood effect in mobile carrier networks, seems interesting. Finally, the results could be validated and detailed via web traffic analysis that takes the size of requested objects into account. Web traffic analysis would also take additional client requests into account that might be missing in the DNS request patterns due to DNS lookup result caching at the client side.

To conclude, we believe that the neighborhood effect as evaluated in this chapter can have a significant positive influence on future Internet architectures in general and ICN architectures particularly, increasing scalability of ICN, reducing latency, and reducing overall

network traffic. The positive influence on NetInf's NRS framework is illustrated and discussed in detail in the following Chapter 5.

5 HIERARCHICAL NAME RESOLUTION

This chapter is based on work published in references [35, 36, 37, 38].

In this chapter, we design a general hierarchical name resolution framework and two specific hierarchical name resolution services for NetInf that expose a trade-off between different design choices. In addition, we describe a global *Resolution Exchange (REX)* system, which can further improve the global scalability of our NRSes. These NRSes build the foundation for scalable, name-resolution-based object retrieval in NetInf.

5.1 INTRODUCTION

As mentioned before, current ICN architectures can be divided into systems with hierarchical object namespace (e.g. CCN) and systems with a flat object namespace (e.g., NetInf, DONA, PSIRP). For hierarchical namespaces, proven resolution systems with acceptable latency exist (e.g., DNS). Developing an NRS for flat namespaces is more difficult as hierarchy-based aggregation is not possible. In this chapter, we focus on building a global NRS for flat namespaces that meets the specific ICN requirements as discussed in Section 5.2. Note that the flat IDs are used to identify the *named data objects* in ICNs. This flat object namespace is independent from the namespace used for network nodes, which might be hierarchical.

We have developed a general NRS framework (Section 5.3) for flat namespaces that can benefit from any network copy while automatically preferring "close" copies, reducing network load, providing low latency, and being resilient to network partitioning. Our NRS framework is based on the following two observations:

1. Existing networks are a connection of autonomous systems that are inter-connected in an essentially hierarchical fashion. We assume that this will also be the case for the foreseeable future.

2. Registration and requests for information follow some locality pattern due to underlying usage patterns and human interest as illustrated in Chapter 4.

Consequently, our NRS framework is based on a hierarchy of independent resolution domains (i.e. subsystems), each typically relying on DHT technology internally. The underlying DHT technology can

be chosen freely, leading to multiple specific incarnations of our general NRS framework that can be divided into two main categories concerning the intra- and inter-domain routing: a *heterogeneous* approach with several (potentially) different, interconnected DHTs and a *homogeneous* approach based on a single system-wide DHT. This choice has strong implications on the overall NRS characteristics. Specifically, it provides an inherent trade-off between request latency, maintenance overhead, and memory requirements on the one hand and resolution-domain autonomy (with significant influence on deployability) on the other hand.

We explore the design space between homogeneous and heterogeneous routing in more detail in this chapter. Based on our general NRS framework (Section 5.3.1), we have developed two alternative NRS systems that explore both sides of this trade-off. The MDHT system (Section 5.3.2) is a generic framework for interconnecting multiple *heterogeneous* resolution systems into a global name resolution system. Each independent resolution domain can be based on its own internal DHT technology, hence, offering autonomy of resolution domains. The *HSkip* architecture (Section 5.3.3) extends the SkipNet architecture [109] into a hierarchical NRS with *homogeneous* routing/forwarding protocol, thereby improving the overall system latency. To further improve global scalability, we have developed a complementary REX system (Section 5.4) that benefits from aggregation at the top level if the namespace contains some basic structure like the NetInf naming scheme (i.e., (type | authenticator | label) as described in Chapter 3). We have performed a theoretical analysis (Section 5.7) and a complementing simulation (Section 5.8) with a focus on latency, load distribution among the NRS nodes, and the overall system characteristics. We conclude by discussing related work in Section 5.9 and summarizing and discussing the results of this chapter in Section 5.10.

Although this chapter focuses on the needs of ICN architectures, the presented requirements and resulting name resolution solutions are also applicable in other areas like P2P-supported streaming and file download.

5.2 REQUIREMENTS

The following requirements should be fulfilled by NetInf's NRS.

LATENCY: *Low latency* is important to ensure user acceptance of the system. Especially applications like web browsing require low latency. A study [110] conducted by Forrester Consulting on behalf of Akamai revealed that 47% of the questioned consumers expects a web page to be loaded within 2 s. 40% of the shoppers will wait no more than 3 s before abandoning the site. Hence, we belief that

a name resolution lookup should take well below 1 s. In contrast, much larger average latencies might be acceptable for the download of large files where the time to transfer the data is much larger and interactivity is not required. An exception is audio/video streaming where a short time until the media starts is desirable although the overall transfer time is large.

SCALABILITY: The current number of world-wide "data pieces" (documents, web pages, sensor data, etc.) is approximately 10^{17} and more than doubling every two years [111], while the number of web pages indexed by the main search engines is about 10^{10} in 2012[1]. We expect that a significant fraction hereof might be registered in an ICN. Hence, the system has to be extremely *scalable* to support a large number of data objects as well as up to 10^{10} users and 10^{9} publishers, which is based on the assumption that about 10% of the users publish content (globally).

LOCALITY AND NETWORK EFFICIENCY: To support efficient data dissemination and increase data availability, the NRS should make use of *any available copy* and should keep resolution and data retrieval local to improve network efficiency. Additional metrics like latency and server load should also be usable for locator selection. If a local object copy is available for retrieval, resolution for this object ID should happen locally (i.e., within the same subnetwork) (called *resolution locality*) and the resolver should return the local object locator (called *content locality*). If the two communication endpoints (i.e., requester and NRS) are in the same network domain, the *resolution path* should also be contained within that network (called *resolution path locality*). The same is obviously also desirable for the data path, which is, however, potentially not under the control of the NRS.

Supporting content locality has significant potential to reduce costly inter-domain traffic, overall network traffic, and latency. This is mainly owing to two effects: (1) *request pattern similarity* and (2) the *neighborhood effect*, as described in Section 4.1.

1. Request pattern similarity of homogeneous groups can be exploited by caching requested content and answering subsequent resolution requests with the locator of the locally cached copy.

2. Due to the neighborhood effect, users often pose requests for "local content" as defined in Section 4.1, i.e., content that has a close semantic relationship to the requester's network, e.g., a company's web page or intranet content. If a copy of this local content is stored (and registered at the NRS) within the local network, significant inter-domain and overall traffic can be saved and network latency can be reduced.

1 http://www.worldwidewebsize.com

AGILITY: The NRS has to be *agile* to support frequently created/disappearing copies as well as moving data objects.

CONTROL/SCOPING: The *scope* of registered copies, i.e., where a copy is made available, should be controllable. For example, a user might want to make a data copy known and accessible only within the local network but not outside this network.

DEPLOYABILITY: For a real-world system, *deployability* is important. Deployment is simplified if the system can be deployed "from the edges", i.e., parts of the NRS can be deployed and used without immediately requiring a world-wide deployment. Giving providers *administrative autonomy* of their NRS resolution domain further helps deployment. They should be able to adapt the local NRS to their local network topology (e.g., building a local hierarchy) and local needs, e.g., by adding nodes to the local network independently of the global NRS. Giving providers the flexibility to choose their internal mechanisms (e.g., DHT algorithms) also helps deployment. In addition, giving ISPs as well as companies and institutions (i.e., network providers in general) control of the name resolution process within their own network offers important incentives for them to deploy such an NRS in their networks: Control of the local resolution process enables providers to optimize utilization of their local network, reduce inter-domain traffic, reduce latency, and control registration and access to documents within their own network.

Based on these requirements, we define the following design objectives for our NRS:

- O1: Low latency
- O2: Scalability
- O3: Locality and efficient network utilization
- O4: Agility
- O5: Scoping
- O6: Deployability

5.3 HIERARCHICAL NRS ARCHITECTURE

In this section, we first introduce our general NRS framework (Section 5.3.1). The general NRS framework can be used for a "pure" name-resolution-based object retrieval approach as well as for a hybrid object retrieval approach as proposed by NetInf. After introducing the general NRS framework, we discuss the two specific NRS incarnations MDHT (Section 5.3.2) and HSkip (Section 5.3.3) in more

detail. Both systems build on the idea of hierarchical DHTs, constructing a hierarchy of DHT-based resolvers. Their main difference is the way the hierarchy of DHTs is constructed and how inter- and intra-domain routing/forwarding is performed. We compare both approaches in detail in Section 5.3.4. Section 5.3 does not assume the REX system; REX is introduced in Section 5.4 as additional option to further improve global scalability.

5.3.1 *General NRS Framework*

Our general NRS framework constructs a global, hierarchical, distributed dictionary that is topologically embedded in the underlying network, i.e., the structure of our NRS system is adapted to the structure of the underlying network topology so that neighboring NRS nodes are also topologically close to each other in the underlying network topology. As described in Chapter 2, it stores bindings between object IDs and corresponding routing hints pointing to available object copies. In addition, entries may also contain metadata related to the requested object. The objects themselves are stored on some independent system, e.g., on common web servers. The NRS is independent of the underlying routing/forwarding and transport layer and can run on top of any infrastructure, e.g., TCP/IP.

5.3.1.1 *Resolution Domains*

We divide a larger network (e.g., the Internet) into multiple administratively autonomous *resolution domains*, typically corresponding to a provider's or company's network. This approach simplifies deployment (objective O6 – Section 5.2). Each resolution domain has its own resolver consisting of one or more NRS nodes which are all infrastructure nodes, i.e., user terminals are not part of the NRS in this hierarchical NRS framework[2].

5.3.1.2 *Hierarchy of Resolution Domains*

We hierarchically interconnect all resolution domains into a global, tree-like distributed dictionary. The hierarchy is topologically embedded in the underlying network, i.e., each resolution domain corresponds to a part of the underlying network topology and the interconnection of the resolution domains corresponds to the network topology. The exact embedding can be freely adapted by the NRS provider (i.e., typically the network provider) to specific requirements. The topological embedding minimizes the routing stretch inefficiencies of common DHTs, i.e., reducing the overall latency (objective O1).

2 This does not preclude user nodes from serving as NRS in general as NetInf supports multiple different kinds of NRSes in parallel (Section 2.2.6). For example, user nodes can serve as NRS nodes in a DTN-like scenario.

Lower-level resolution domains are mapped to smaller, lower-level networks, e.g., a company network or a provider subnetwork (e.g., a PoP). Higher-level resolution domains are mapped to larger, higher-level networks, e.g., an autonomous system (AS) consisting of a set of lower-level networks (Figure 30). Higher-level resolution domains are built by combining and interconnecting all lower-level NRS nodes of the higher-level-resolution-domain's subtree. Each NRS node is part of multiple (or all) resolution levels, i.e., the same *physical node/machine* participates in the higher-level resolution domains as *virtual node*. In other words, the higher-level nodes are typically virtual nodes that are running on the same physical machines as the access nodes at the lowest level. We call this approach *nested*.

In general, we only require the NRS to be a directed acyclic graph (DAG). Hence, it is possible to add additional edges to the tree, e.g., to model peering agreements between providers. For simplicity, we assume a tree structure in the rest of the chapter.

5.3.1.3 *Intra- and Inter-Domain Routing/Forwarding*

Forwarding of queries in the NRS can be divided into intra- and inter-domain, i.e., within and between resolution domains. The detailed mechanisms to perform intra- and inter-domain routing/forwarding can be freely chosen in the respective system implementation. We describe details for the two systems: MDHT (with two variants) in Section 5.3.2 and HSkip in Section 5.3.3. Both are based on a DHT approach. Hence, they inherit the good scaling properties of DHTs for flat namespaces (objective O2).

5.3.1.4 *Data Registration*

The NRS primitive *PUT(ID, metadata)* is used to register bindings in the dictionary, i.e., the ID is made public and bound to a set of locators and/or metadata. Content is first registered by the publisher at the local, lowest-level nodes (called *access node (AN)*, Figure 30) and is subsequently registered in resolution domains between this AN and the root domain (i.e., top-level domain). As a consequence, the system provides inherent redundancy as binding records pointing to object copies are stored at multiple levels, thereby increasing system robustness. Note that due to the fact that content is first registered at its local resolution node, the *resolution information* and the *content* are *both* available locally.

The NRS framework supports multiple *binding schemes*, i.e., different ways how object IDs are (directly or indirectly) bound to network locations object copies. Binding schemes can be designed based on specific requirements. Any kind of routing hint as defined in Section 2.1.1 can be registered at the NRS, including *location bindings*, which map IDs to network locations, other types of *protocol-specific*

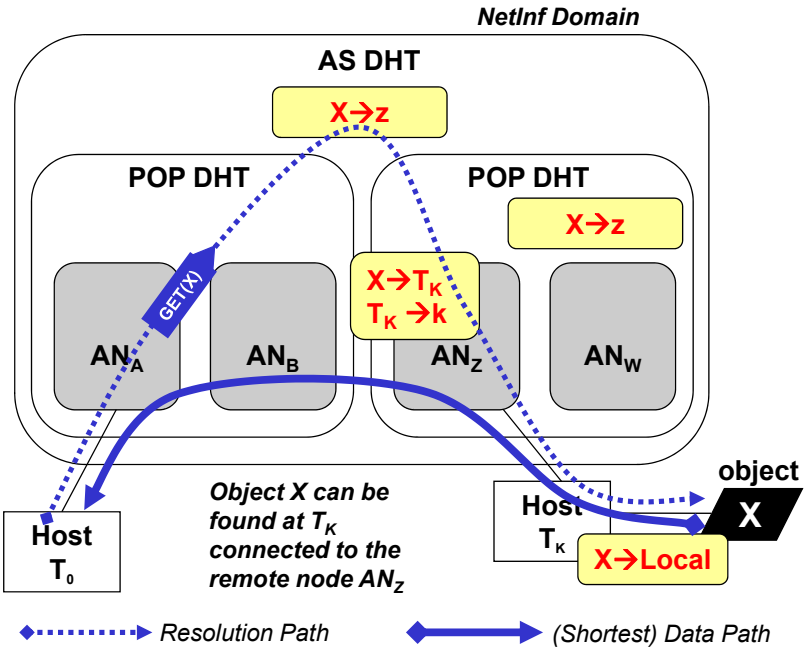

Figure 30: Binding entries and resolution of object X

routing hints, and *indirection bindings*, which map IDs to other IDs. To simplify the description, we subsequently use locators and indirections as binding examples in the remainder of this chapter.

Figure 30 shows the resulting binding entries from an exemplary registration process which uses two levels of indirection and binds object X to its access node and in the access node to its host's ID (the registration entries at each level are shown in the yellow boxes). First, the user's device ID T_K is registered in the NRS system. On the AN, T_K is mapped to its local address k, which can be private. Thereafter, the data object can be registered in the user's AN to make it accessible for local users connected to the same AN. Registering a new object creates a new entry storing an indirection binding that maps the object ID X to the user device ID T_K. Second, the registration request is propagated up the NRS tree (using the intra- and inter-domain routing mechanisms) so that a new binding for object X is recorded in the upper resolution domains along the path from leaf (AN) to root (AS DHT). At all levels, hash(ID) is used as DHT key.

Note that, in the upper resolution domains, the object ID is mapped to the address z of access node AN_Z, where X can be reached via the address k of host T_K. This binding scheme allows to keep host addresses private and prevents firewall issues as responses are received from the initially queried access node. In addition, this indirection scheme has good mobility support for mobile users/objects as the access node can perform a role similar to a mobile IP "home agent", redirecting requests to the new location of node T_K. It also sup-

ports multihoming and traffic redirection towards application servers. Note that other binding schemes are possible to achieve other goals.

To provide control of ID registrations, the publisher can specify the *scope* of the registration for each ID, i.e., the registration message contains information up to which level registration should take place. The scope limits the propagation of entries within the tree. For example, a publication can be restricted to the local company network (objective O5). As the local NRS nodes are controlled by the local company itself, the publisher can trust these NRS nodes to comply to the scoping instructions. Contractual relationships with higher-level providers can ensure compliance at higher levels.

As the number of levels is typically bound by a small constant, only a small constant number of steps is required to add a new registration or perform an update, ensuring system agility (objective O4). Note that binding records for the same object at different levels are not identical but are localized (which does not restrict the redundancy property), i.e., they each contain the bindings to local copies in their respective subtree in the hierarchy of resolution domains.

5.3.1.5 *Resolution*

The NRS primitive *GET(ID)* returns a list of routing hints (typically object locators, i.e., where to retrieve the object) or the object itself. In addition, related metadata can be obtained. Besides the registration process as described above, Figure 30 also shows a user (host T_0) *requesting* a data object by ID X. The request is first processed in the access node AN_A. From the client's point of view, the AN is somewhat similar to a local DNS server. Clients can learn about their local AN in a similar way as today about a DNS server, e.g., via DHCP. When resolution in the AN fails because the ID is unknown (i.e., no copy is registered in this area yet), the request is propagated to the next higher resolution domain until a hit is found or the resolution fails at the highest level. The actual data transfer is subsequently performed directly between requester and source.

By starting the resolution process at the AN and propagating the request higher only if required, the (resolution) routing stretch of the NRS is reduced (objective O1) because NRS nodes in lower resolution domains are generally closer to each other than nodes in higher resolution domains (in terms of *number of hops* and *hop latency*). The hierarchical registration/request scheme is also instrumental to achieve content, resolution, and resolution path locality and to efficiently utilize the network (objective O3). It naturally exploits the neighborhood effect. Locally hosted content (which is typically frequently requested) is registered locally and these registration bindings point to all locally available copies. Likewise, it exploits request-pattern similarity of homogeneous groups as popular content can be cached locally and these copies are then also registered locally. Both effects

can significantly reduce inter-domain traffic. These advantages provide important deployment incentives for network providers (objective O6).

5.3.1.6 *Caching and Replication*

When a new copy is stored, e.g., at a user's local machine (peer-side caching) or in the network (in-network caching), the new copy can be made available for other users by registering its locator (or some other routing hint) in the NRS system. The scope of each published copy, i.e., the level up to which the copy is registered, can be limited via scoping. For example, limiting the registration to a few lower levels can limit the load on the user's node/caching server.

5.3.1.7 *Locator Selection*

Our NRS framework supports three different ways to select appropriate locators. In the *requester-controlled mode*, the NRS returns the list of locators to the requester who subsequently selects a suitable subset and triggers the download of the data. The requester has full control over the process. It is important to note that, due to the hierarchical registration/request approach, the set of locators returned by the NRS automatically consists of close-by locators. Additional locators can be requested explicitly by querying higher-level resolution domains.

In the *NRS-controlled mode*, the NRS system performs the locator selection. Because the NRS is embedded in the underlying network infrastructure, various algorithms can be developed that make use of internal knowledge about the network topology and current network and server statuses to select the best locator(s). In addition, the NRS system can also trigger the data transfer from the source(s) to the requester in this scenario, i.e., the resolution would become transparent to the requester. This integration of resolution and data forwarding phase typically also reduces the overall routing stretch as it eliminates one extra communication step between NRS node and requester.

In the *hybrid mode*, the NRS returns a *ranked* locator list, based on its *network knowledge*. The requester can choose the desired locator(s) based on the ranking and other factors, and downloads the data. This mode combines best locator selection based on network knowledge with full requester control.

5.3.1.8 *Security and Trust*

Our NRS framework gives a network provider full control over the resolution process within its own network, thereby enabling trusted scoping within its network (i.e., nodes do not forward registrations outside the provider network if not desired) and preventing DoS at-

tacks and traffic analysis of NRS requests as long as the network is not compromised.

When NRS registrations/requests leave the provider's own network, i.e., at higher levels, other NRS nodes might become responsible for the provider's own object IDs as well as for answering the provider's NRS requests (i.e., its customer's requests). In general, any misbehavior of NRS nodes can be penalized. To enforce this, authentication (based on public/private key signatures and a chain of trust) is required for each NRS node before joining the overall NRS network. This results in a global network of trusted NRS nodes that can each be secured with common means. However, like in today's networks, security is at risk if an attacker gains control of an NRS node. In the following, we will discuss security risks and technical barriers of our NRS framework that make misbehavior/attacks harder even if an attacker gains control of an NRS node.

Assuming a security breach at an NRS node, four main problems can occur. First, a resolver could return a wrong locator to foist wrong information on the requester. This is not an issue in ICNs due to their information-centric security concepts. An ICN typically supports *name–data integrity*, which ensures that the requester can verify that the received content corresponds to the queried object name.

Second, in a DoS attack, a node responsible for a given ID might maliciously not return locators for this ID. To address this problem, user nodes can have direct, redundant connections to higher levels to circumvent malicious or erroneous nodes. Hence, a requester can send its request directly to a higher level, which typically also contains the requested information as a result of the inherent redundancy that guarantees that resolution entries typically exist at all/multiple levels. Hence, to perform a dedicated DoS attack on a given ID (range), an attacker would have to control all nodes that are responsible for this ID at all levels. Such an attack can further be complicated by preventing NRS nodes from choosing their own numerical node ID themselves.

Third, an NRS node might not forward NRS requests. This can be detected by other NRS nodes, e.g., via repeated requests, and misbehaving NRS nodes can be bypassed due to the fact that both MDHT and HSkip rely on DHT systems that typically have redundant connections within the DHT network like Chord's multiple "finger pointers" [112].

Forth, the NRS system could be used to run a DoS attack on other targeted systems. For example, an attacker that gains control of NRS nodes could redirect a flood of object requests to the targeted system by returning the locator of the targeted system as the source of popular data objects. As a consequence, the targeted system would be overwhelmed by a flow of redirected object requests. As in today's

Internet, such attacks have to be precluded by preventing access to the NRS nodes.

5.3.2 *MDHT*

The Multi-Level Distributed Hash Table (MDHT) system is a *heterogeneous* incarnation of our general hierarchical NRS framework, i.e., it builds on multiple interconnected DHTs.

HIERARCHY OF RESOLUTION DOMAINS: The tree structure is hierarchical and can consist of any number of levels and can be unbalanced and asymmetric as required. Interconnecting the separate resolution domains into an overall system is done by configuring the affiliation of NRS nodes to specific resolution domains. Each NRS node can choose the number of levels that it participates in. However, in practice, the provider has to ensure that there are still enough NRS nodes at each level. NRS nodes should typically participate in all levels to simplify inter-domain forwarding.

INTRA- & INTER-DOMAIN ROUTING/FORWARDING: The MDHT system uses separate mechanisms for intra- and inter-domain request routing/forwarding. For *intra*-domain routing/forwarding, each resolution domain (i.e. subsystem) can choose its own distributed dictionary technology depending on the specific needs, which can include DHT mechanisms as well as other mechanisms. For large resolution domains that require many NRS nodes, an $\mathcal{O}(\log n)$ (n = number of NRS nodes) DHT approach is suitable. However, the number of nodes in most resolution domains is expected to be rather small (typically from tens to thousands of nodes) and these nodes are carrier-grade, stable and reliable with almost no churn. Hence, further performance improvements can be obtained by using $\mathcal{O}(1)$ DHT protocols for intra-domain forwarding. This reduces round-trip delays and keeps the routing stretch in each domain close to 1, that is, it shortens the whole resolution and routing path.

For *inter*-domain routing, the MDHT system supports two different mechanisms, an *independent* approach and an *entangled* approach. In the following, we always refer to both approaches unless explicitly stated.

Neither approach requires a separate routing table or routing algorithm for *inter*-domain forwarding as all required information is included in the configuration of the NRS hierarchy. Inter-domain forwarding is performed by iterating through the resolution levels starting at the lowest level, the local AN. Levels are numbered top–down. Assuming that node a would be responsible for the requested ID at level i and resolution fails at this level, the request is forwarded by node a to the next higher level $i-1$. As a result of the nested design,

node a is most likely also part of level $i-1$. Hence, inter-domain forwarding can be performed internally at node a simply by handing the request over to the local intra-domain forwarding process of level $i-1$. This allows fast, robust inter-domain forwarding. In the rare case that a node is not part of the next higher level $i-1$, the request is forwarded to the next node at level i that is also part of level $i-1$. If no node of level i is part of the next higher level, node a uses a set of (redundant) links to nodes at level $i-1$ which has to configured by the NRS provider. Similarly, each level also contains some redundant links to all other higher levels to bypass levels in case of problems.

The overall latency of this *independent* approach can be improved significantly by the *entangled* mechanism. The only difference between both mechanisms is that the entangled mechanism assumes a certain level of similarity between the different resolution domain technologies. More precisely, it requires that NRS nodes have the same node ID at all levels, all resolution domains use the same key for the same object ID, and they use a *similar rule* to define the node responsible for a given object ID. In this case, "similar rule" means that the rule does not have to be identical but has to express a generally similar idea which node is responsible for an ID. For example, let us assume that in resolution domain 1, the node with the next higher ID is responsible for a given object ID; in resolution domain 2, the node with the next smaller ID is responsible for the same object ID. In this case, a request that is forwarded from resolution domain 1 to resolution domain 2 might have to travel completely around the DHT ring again although the responsible ID was missed only a little bit (assuming that resolution domain 2 does only allow to travel the DHT ring in one direction). If these requirements are fulfilled, the forwarding steps performed at level i also bring the request closer to the responsible node at level $i-1$. Hence, resolution at level $i-1$ already starts close to the NRS node responsible for the requested object ID, thereby reducing the number of resolution steps.

While the *independent* MDHT approach has an average overall latency of $\mathcal{O}(m \cdot \log n)$ (m = number of levels; n = number of NRS nodes) assuming the worst case that all levels have to be consulted, the *entangled* MDHT approach has an average latency of $\mathcal{O}(\log n)$. Details are discussed in Section 5.7.

5.3.3 *HSkip*

This section first gives a short overview of the original SkipNet architecture. Further details can be found in the original publication [109]. Based on this overview, we then describe our Hierarchical SkipNet (HSkip) architecture.

5.3.3.1 *SkipNet Overview*

SkipNet is a DHT architecture based on the idea of Skip Lists. In SkipNet, each object has a *string ID* and each node has a similar string ID and, in addition, a *flat numeric ID* chosen randomly from a uniform distribution. The string IDs can be hierarchical. SkipNet can do routing/forwarding in both namespaces with an average latency of $\mathcal{O}(\log n)$. When forwarding within the string namespace, SkipNet supports the two properties *resolution path locality*[3] and *binding record locality*[4]. Binding record locality allows to define on which (subset of) node(s) an object should be stored.

STRING FORWARDING: SkipNet constructs a doubly-linked ring of all nodes, sorted by string ID. Each node has $2 \cdot \log n$ pointers that leap over an exponentially increasing number of nodes, similar to Chord's finger table. When forwarding to a certain node ID, SkipNet follows these pointers and leaps over as many nodes as possible without surpassing the destination ID until the destination is reached.

NUMERIC FORWARDING: The SkipNet architecture constructs multiple doubly-linked rings by dividing the main ring into two rings, which are each subsequently divided again, and so on. The binary, numeric node ID determines in which ring a node participates. Forwarding happens by finding the next node in the main ring that shares the same first numeric digit with the requested numeric ID. SkipNet then changes into the next ring containing this node and repeats the same process with the next digit until no more progress can be made. Then, the node with the closest numeric ID is chosen.

CONSTRAINED LOAD BALANCING: SkipNet uses a combination of string forwarding and numeric forwarding to achieve a load balancing mechanism constrained to a subset of nodes, called constrained load balancing (CLB). To achieve this, SkipNet divides each object ID into two parts. First, the *subgroup part* that specifies the subset of nodes, i.e., all nodes with this common string ID prefix. Second, the *object part* that serves as unique object ID within this subgroup. The *hash(object part)* is used to determine the specific node within the subgroup of nodes that is responsible for this object ID. CLB forwarding is performed in two steps. First, finding any node that is part of the subgroup via string forwarding as described above. Second, finding the responsible node within this subgroup by numeric forwarding. Here, the above described numeric forwarding mechanism is slightly

3 Called *path locality* in SkipNet, fully corresponding the our definition of *resolution path locality*.

4 Called *content locality* in SkipNet, however, not fully corresponding to our *content locality* definition in Section 5.2 as SkipNet generally stores content and not resolution bindings as we do.

modified to make sure that the subgroup of nodes is not exited during forwarding. This is done by reversing the forwarding direction when the boundary of the subgroup (as specified by the common string ID prefix) is reached.

One might think that the original SkipNet architecture can directly be used to provide a hierarchical NRS for flat namespaces. First, SkipNet provides some kind of hierarchy (hierarchy of rings in the *numeric namespace*). However, the numeric IDs cannot be used to construct a topologically-embedded hierarchy as needed by our NRS as the numeric SkipNet IDs have to be uniformly distributed. Second, SkipNet provides resolution path and binding record locality only for the *string IDs*. However, because of the required topological embedding in the underlying hierarchical network topology, the string IDs have to be hierarchical, contradicting our requirement of a flat namespace. Moreover, SkipNet generally does not provide any means to discover the nearest object copy. Hence, we have to find a way to provide resolution path locality and binding record locality for a flat namespace, combined with a topologically-embedded hierarchy to find the nearest object copy. For this reason, we have developed HSkip.

5.3.3.2 *HSkip Architecture*

HSkip is a *homogeneous* hierarchical DHT based on SkipNet. HSkip differs in two main aspects from SkipNet. First, we construct a nested hierarchy of resolution domains, embedded in the underlying network topology. Second, we provide binding record and resolution path locality while naming objects with the *numeric IDs*. This is essential to perform our hierarchical registration/request scheme. In addition, in consequence of naming objects with *numeric* IDs, transparent remapping of objects to a new resolution domain is possible, which is not possible in SkipNet as the domain is encoded in the object's string ID.

HIERARCHY OF RESOLUTION DOMAINS: Our hierarchical design is based on a single SkipNet DHT. SkipNet allows to use a hierarchical string namespace for naming SkipNet nodes. We use this to construct our hierarchy of nested, topologically embedded domains. Each HSkip node is named using a string ID chosen from a hierarchical namespace corresponding to the underlying network topology. For example, DNS-like names (e.g., *country.provider.organization.node*) can be used. All nodes with a common name prefix are grouped together into one domain. For example, all nodes with the name prefix *countryX* are part of the domain *countryX*. All nodes with the name prefix *countryX.providerJ* are part of the domain *countryX.providerJ*. By allocating names in this way, the nested hierarchy is constructed automatically when nodes join the HSkip network, using the standard

SkipNet join process [109]. The numeric node IDs are chosen randomly from a uniform distribution without special constraints, similar to the original SkipNet design.

Figure 31 shows an example HSkip resolution domain hierarchy. The leafs of the tree (circles) represent physical nodes. The complete string ID of a node is given by following the path from the root of the tree to the node, i.e., the left-most node has the ID *x.j.a.1*. All other tree nodes (squares) represent higher-level resolution domains. Each resolution domain consists of all nodes in their subtree. For example, the resolution domain *x.j* contains the nodes *x.j.a.1*, *x.j.b.1*, and *x.j.b.2*. Note that the HSkip nodes are not explicitly connected in this hierarchical way. Rather, the hierarchy is only given implicitly by the nodes' string IDs which are used by our forwarding protocol to traverse the HSkip network in this hierarchical way as described subsequently. It is important to note that this hierarchical structure imposed by the string IDs has no correspondence to the tree-like SkipNet levels created via the numeric IDs.

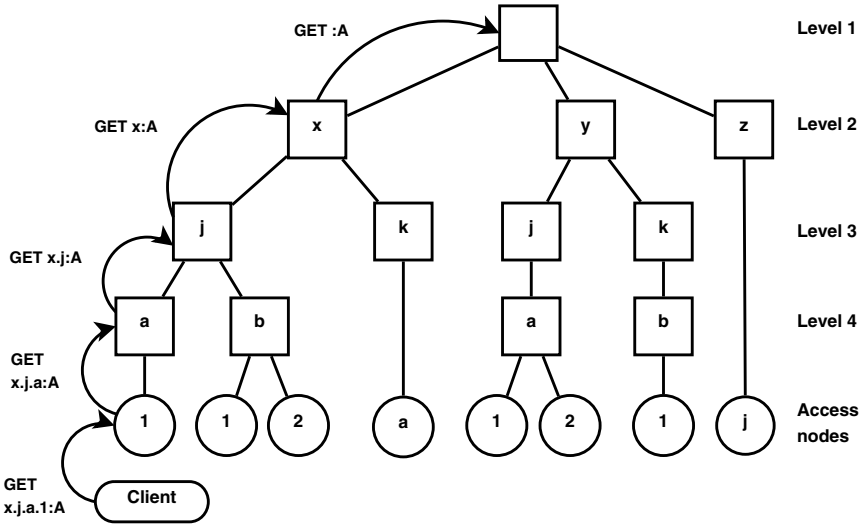

Figure 31: HSkip hierarchy with GET request for object *A*

LOCALITY PROPERTIES WITH NUMERIC IDS: The original SkipNet design supports binding record locality and resolution path locality. These are important properties for our NRS. Resolution path locality is a direct requirement defined in Section 5.2 and binding record locality can be used to achieve resolution locality and scoping (Section 5.3.1) when storing binding records for locally available content within the local resolution domain.

A major SkipNet drawback for us is that it only supports those locality properties when objects are named with the string IDs. SkipNet achieves those properties via CLB. The name of the resolution node/domain where the entry should be stored is attached to the object

name. In our case, where the underlying network topology is hierarchical and, consequently, the resolution domains are constructed and named in a hierarchical fashion, this approach would result in hierarchical object names. However, as described in Section 5.1, we require an NRS for a flat namespace. Therefore, HSkip names objects with the flat numeric namespace. To preserve binding record locality and resolution path locality for the numeric namespace, we introduce our own intra- and inter-domain forwarding protocol.

INTRA- & INTER-DOMAIN ROUTING/FORWARDING: Our intra-domain forwarding protocol is based on SkipNet's CLB mechanism. The routing tables are constructed like in the original SkipNet design. Intra-domain forwarding at different hierarchical levels can be done with the same set of routing tables, i.e., HSkip requires only a single routing table per node. Therefore, we characterize HSkip as *homogeneous* hierarchical DHT.

As shown in Figure 31, a request for object ID A is started by first querying the local access node (AN) *x.j.a.1*. Note that Figure 31 shows the alphanumerical string ID of the HSkip *node x.j.a.1* (which is hierarchical as described previously) separated by colon from the *flat* object name A. The hierarchical node ID is not part of the flat numerical object ID and is only used to perform SkipNet's CLB forwarding as described below.

If resolution fails at the local AN, resolution continues at the next level *x.j.a*. If resolution also fails here, the node responsible for this ID at level *x.j.a* forwards the request to the next higher level *x.j* until resolution succeeds or fails at the top level as described in Section 5.3.1.5. At each level, resolution is constrained to the specific resolution domain via SkipNet's CLB forwarding. For example, in the resolution domain *x.j.a*, resolution is restricted to nodes with the prefix *x.j.a*. Levels can be bypassed if required, e.g., due to problems at a certain level, by continuing directly with a smaller prefix.

As explained in Section 5.3.3.1, the CLB mechanism consists of two steps in SkipNet: (1) forwarding the request to a node that is part of the constrained domain (*string forwarding*), and (2) forwarding to the responsible node within the CLB domain (*numeric forwarding*). As a result of the way that a request is started and forwarded through the hierarchical resolution domains in HSkip, the request path by definition starts at a node that is part of the current restricted CLB domain. Hence, string forwarding of the CLB mechanism can be omitted. Only the numeric forwarding has to be carried out.

SkipNet's numeric forwarding requires $\mathcal{O}(\log n)$ steps at each level. HSkip's inter-domain forwarding works similar to the forwarding of the *entangled* MDHT, i.e., the forwarding steps performed at level i bring the request closer to the responsible node at level $i-1$. Hence, even assuming the worst case that all levels have to be consulted,

HSkip only requires $\mathcal{O}(\log n)$ resolution steps in total. This is shown in detail in Section 5.7.

5.3.4 *System Comparison: MDHT vs. HSkip*

Due to its homogeneous design, HSkip requires less memory on the NRS nodes. Each node only requires a single routing table whereas MDHT requires a separate routing table for each level, which are each smaller than the HSkip routing table but still contain some redundancy. In addition, HSkip has a lower setup and maintenance overhead compared to MDHT. Adding/removing nodes in HSkip is easier as the hierarchy is built implicitly based on HSkip's node IDs. In MDHT, adding/removing nodes requires some separate configuration at each level (e.g., joining/leaving the separate DHTs).

HSkip's drawback is that it has no strict separation of resolution domains. As a consequence, nodes always have to be part of all levels and providers cannot choose their own intra-domain routing/forwarding technology. In contrast, the *independent* MDHT offers providers full intra-domain autonomy. They can freely choose the best intra-domain routing/forwarding technology for their specific requirements. Moreover, they can change and extend their hierarchical intra-domain substructure and can add local nodes independently of the global system structure.

The main advantage of the *entangled* MDHT and HSkip (compared to the *independent* MDHT) is the better latency due to gradually closing in on the responsible node ID at each level. In exchange, the entangled MDHT gives up some of the resolution domains' independence as node IDs have to be identical at each level and the choice of DHT technologies is somewhat restricted as discussed in Section 5.3.2. In HSkip, all resolution domains have to use the same SkipNet-based DHT.

Both MDHT systems offer full control over resolution domain interconnection. As a consequence, MDHT does not necessarily have to start as a single global resolution system but can gradually grow from the edges of the Internet. This simplifies deployment and migration significantly compared to HSkip. In HSkip, partial deployment is more difficult as all HSkip nodes have to be interconnected into a single large HSkip right from the beginning, hence, requiring more coordination between the NRS providers.

In summary, both MDHT and HSkip meet our requirements as defined in Section 5.2. However, there is an inherent trade-off between autonomy of resolution domains on the one hand and reduced latency, maintenance cost, and memory needs on the other hand. This trade-off exists between MDHT and HSkip in general as well as (at a smaller scale) between the *independent* MDHT approach and the *entangled* MDHT approach.

5.4 GLOBAL NAME RESOLUTION

Global resolution at the highest NRS level must be highly scalable because of the large number of globally available data objects. If the namespace is flat without any structure, the top level can use a global DHT composed of all/most of the NRS nodes from the lower levels. As a result of the hierarchical NRS structure and the expected locality of requests, most requests can be answered at lower levels. Hence, load on the top level will be limited. Our simulation results (Section 5.8) support this expectation.

If the namespace is structured, e.g., names contain a name prefix like in NetInf (Chapter 3), global resolution of MDHT and HSkip can be further improved by combining either with our Resolution Exchange (REX) system at the top level. The global REX system can be managed by an independent, trusted third party, just like the DNS top-level domains. This third party guarantees its clients the correct management of their resolution bindings based on a business agreement.

REX is based on the idea that each owner/publisher has a *primary resolution domain* where its IDs are stored (typically the resolution domain of the object owner's ISP). The REX system performs aggregation based on object name prefixes (i.e., the *authenticator* field as described in Chapter 3) and redirects requests to the owner's primary resolution domain. Let us assume an ID structure *A:L* (equivalent to the NetInf naming structure as presented in Chapter 3), where *A* is a name prefix with some semantic (a `hash(PK)` in the NetInf case), and *L* is a label, which unambiguously identifies the object in the scope of *A*. This structure can be used to aggregate data objects with the same name prefix into a single binding, i.e., the REX system has to scale only with the number of prefixes and not with the much larger number of data objects[5]. The REX system stores only redirects to the primary resolution system. The *A* part of each ID can be mapped and redirected to the primary resolution system, which is responsible for managing bindings for all IDs with that prefix.

Although the REX system can be designed to handle and redirect the resolution traffic itself, we think of it more as an administrative entity. Instead of performing resolution itself, the REX system only manages registrations, updates, and aggregation of bindings at the global level. These aggregated bindings are then cached within the second-level resolution domains. By using these cached bindings, requests for IDs registered in other resolution domains can be redirected to the appropriate primary resolution system at the level below

5 We assume here that a publisher uses a limited set of name prefixes to name his/her objects. Although each NetInf object can have a different prefix in theory, we belief that this will not be the case in reality. In addition, users can be incentivized to limit their use of prefixes, e.g., by requesting a fee for additional prefixes. This is obviously a trade-off with owner pseudonymity.

REX. Hence, the requests do not have to be forwarded to the REX system.

When assuming up to 10^9 aggregated REX entries (the expected number of publishers) at an estimated size of 1 KB each, the complete REX database size is 1 TB. Hence, it is feasible for Tier 1 providers to cache the complete REX database in their second-level resolution domains (distributed among the many NRS nodes at this level). Regional providers can choose to download only a subset of bindings associated to close networks, and/or use a default route to upstream providers with a larger cache.

Resolution domain providers can choose to directly interconnect their resolution domains below the top level to improve the topological embedding of the resolution domains, e.g., at a regional or national level. This is similar to today's peering between network providers for directly exchanging traffic. For example, providers can build a joined DHT at a certain NRS level. If no sufficient trust relationship exists, providers can still interconnect their resolution domains by locally caching the aggregated REX entries of the IDs registered in the respective other resolution domain. In both cases, shortcuts for efficient redirects between resolution domains are created, reducing resolution load at the higher levels and improving resolution latency.

5.5 LOAD BALANCING

It is desirable to achieve an equal load distribution among resolution nodes within each resolution domain as well as among all resolution domains. The load distribution among the nodes depends on the load balancing properties of the utilized structured P2P technology. Fortunately, both our Chord-based MDHT implementation and SkipNet-based HSkip implementation provide a good load balancing among the resolution nodes as shown in Section 5.8.3. To further improve the load distribution, additional load balancing mechanisms for structured P2P networks can be utilized.

Due to their strict data placement rules, structured P2P networks are often prone to three main load imbalances: *range skew*, *data skew*, and *execution skew*:

RANGE SKEW refers to an uneven partition size among peers, i.e., it is the ratio between the size of the smallest and the largest partition of the ID space that a peer is responsible for [113].

DATA SKEW refers to an uneven distribution of data items across the peers' partitions [114]. Data skew typically occurs in situations when uniform hashing of data items cannot be applied. For example,

this may occur when uniform hash functions cannot be used, e.g., as they destroy the locality of data items for range queries.

EXECUTION SKEW refers to non-uniform data access across the peers' partitions [114]. For web traffic [115] and multiple P2P applications [116], the popularity of data items follows a Zipf-like distribution; peers in charge of popular data items receive significantly more requests than others.

Many of today's load balancing algorithms focus only on range skew and assume the network data rate to be the bottleneck. However, in services that focus on distributed request processing such as complex name resolution mechanisms that involve, e.g., security checks like in NetInf, these assumptions are not valid.[6] In these services, messages are typically small but can produce significant load at the application level. Here, data skew and execution skew are most important and the NRS performance is limited by the number of resolution requests a peer can process.

To provide a solution for load balancing in such scenarios, including NetInf's NRS, we have developed a new load balancing algorithm that is based on ID management. Our algorithm collects statistics of overlay link usage during normal operation and uses this information to provide suitable IDs to joining peers. Without using regular maintenance messages, it improves the rate of successfully answered requests by a factor of up to 3 in typical scenarios. We have evaluated the algorithm via extensive simulation that also includes scenarios with churn and heterogeneous peers. This work presents the first load balancing algorithm that can handle all three types of skew in scenarios that focus on processed application requests as the bottleneck. Hence, this load balancing algorithm is a suitable option to further improve the load balancing properties of MDHT, HSkip, and other NRS solution. To keep this thesis focused on the main NetInf aspects, the load balancing work is only summarized in this thesis. Details can be found in references [37, 38].

5.6 SCALABILITY AND NODE PERFORMANCE ANALYSIS

In the following, we give a preliminary and simplified assessment of the system scalability in terms of required number of world-wide NRS nodes and the nodes' system requirements. For this assessment, we assume a world-wide NRS system with 9 levels and all nodes participating in all levels. Our following estimation is independent of the usage of the REX system and is valid for both the MDHT and

6 Other services focusing on distributed request processing include search services for complex queries as discussed in Section 2.2.8.

HSkip system as the storage requirements for binding entries is the same for both systems.

As low latency is important, the binding records cannot be stored on a conventional hard drive. However, solid state disk (SSD) memory offers sufficiently fast access ($\approx$ 25 µs). Current state-of-the-art SSD storage servers have up to 24 TB of memory (e.g., RamSan [117]). In the following analysis, we assume 18 TB of memory. Since all nodes equally participate in the top DHT and in the nested levels of the hierarchy, they all roughly store the same amount of bindings. Assuming 10^{15} objects in total (as outlined in Section 5.2) with an estimated binding record size of 1 KB, each NRS node can store $2 \cdot 10^9$ binding records on each of the 9 NRS levels with 18 TB of memory. Therefore, roughly 10^6 NRS nodes are required for a world-wide NRS system with 10^{15} objects. This estimation assumes on average two binding records per object at each level. In fact, some popular objects can have more than two binding records per level (at most one per resolution domain plus some redundancy against failures), whereas many other objects may only be locally registered, i.e., they do not require binding records at higher levels at all.

The estimated number of 10^6 NRS nodes is only about 1/10th of the roughly 12 million DNS nodes in today's Internet [118]. If all binding records are stored in a balanced binary tree (BBT), up to 30 tree levels are necessary for a resolution lookup inside a single dictionary node. Hence, the storage access latency of a single `GET` operation is about 750 µs (25 µs access time $\times$ 30 tree levels).

To evaluate the number of supported users, we assume that the number of `GET` requests significantly dominates the overall number of requests. With more than 450 000 memory access operations/s supported by current SSD storage servers, an NRS node is able to manage more than 15 000 `GET` requests/s (when assuming 30 levels in the BBT as calculated above).

DNS measurements [119] have shown that user DNS requests per second are typically well below 0.1 requests/s and for most users on average well below 0.01 requests/s. For future information-centric `GET` requests, we use estimates based on the number of DNS requests. However, it can be assumed that there will be more `GET` requests in an information-centric network as DNS requests are for complete domains whereas ICN `GET` requests are for individual objects. Hence, we estimate on average 1 ICN `GET` request/s per user, which is two orders of magnitude larger to account for this difference. With on average one `GET` request/s per user and requests going up on average half the hierarchy before finding a hit, i.e., 4.5 hierarchy levels, a single NRS node can still handle more than 3300 users with a single storage unit. Therefore, 10^6 NRS nodes can handle more than $3.3 \cdot 10^9$ users concurrently. Further improvements can be obtained by replicating access nodes to exploit parallelism.

Another important issue of dynamic NRS systems is the bandwidth required to *perform binding refreshes*. We assume a certain level of aggregation of refresh messages (e.g., each packet of 1500 bytes contains on average 10 binding refreshes). Assuming 0.1% changing entries per day, 18 million entries would change on a single NRS node, resulting in 1.8 million refresh messages or 2.7 GB. Therefore, a node would require ≈ 0.25 Mbit/s of continuous refresh bandwidth to handle 0.1% changing entries per day. Even for 1% changing entries per day, only 2.5 Mbit/s refresh bandwidth would be required.

5.7 SYSTEM ANALYSIS

In this section, we evaluate the expected latency behavior of MDHT and HSkip for a world-wide system with up to 12 million NRS nodes based on a theoretical analysis, with a focus on latency and on the overall system characteristics.

We define resolution latency as the time between sending a resolution request and retrieving the answer, i.e., a set of locators. Latency on the last mile between user and first NRS access node is excluded as it heavily depends on the access technology and would not differ between MDHT and HSkip anyway. We make the following assumptions and definitions:

- We assume a full k-ary tree structure of depth m, i.e., every node except for leafs has k children.
- Levels are numbered top–down, *excluding* the lowest level (access nodes, Figure 31). Hence, $m = 1$ equals a flat DHT.
- The total number of *physical* NRS nodes (i.e., leafs in the tree; note the *nested* design) is $n = k^m$ for $k \geqslant 2$.
- We vary n and m. Consequently, k is given as: $k = n^{\frac{1}{m}}$.
- We neglect processing delays at NRS nodes, which are comparably small. They are dominated by dictionary lookups, which are typically $\leqslant 0.75$ ms as outlined in Section 5.6.
- We consider the total number of objects indirectly via the number of NRS nodes. The number of nodes increases with the number of objects to ensure efficient object handling at each node.

5.7.1 *General Analysis Approach*

In general, the name resolution delay is a combination of processing delay and network latency. As outlined above, we neglect the processing delay as it is comparably small and only required once per

MDHT level. Hence, the resolution latency is dominated by the network latency.

If resolution at level i fails, it is performed at the next higher level until a hit is found or the top level is reached. No network latency occurs at the AN level ($i = m + 1$) as it only consists of separate ANs. Hops *between* levels typically happen on the same physical node because each node typically participates in all levels. In some rare cases (but at *most* once per level), inter-level hops can happen between two separate nodes. However, these nodes are by definition close to each other with a latency of $\leqslant 0.5\,\text{ms}$ based on our latency model. Hence, in both cases, inter-level hops do not add significant latency and can be neglected for real-world systems with a limited number of levels.

We define:

- L: Random variable, overall resolution latency
- L_1: Random variable, latency to find the answer
- L_2: Random variable, latency to return the answer
- Y_i: Event, object found exactly at level i
- $p_i = P(Y_i)$
- $\overline{h}_i$: Average number of overlay hops at level i
- $\overline{l}_i$: Average overlay hop latency at level i
- $\overline{x}_i$: Average latency to find the answer (excl. the return path) when object is found at level i

$\overline{x}_i$ can be calculated by summing up the average latency for all overlay hops from level m up to level i:

$$\overline{x}_i = E[L_1|Y_i] = \sum_{j=i}^{m} \overline{l}_j \cdot \overline{h}_j \tag{1}$$

We can calculate the average overall resolution latency $E[L]$ as the sum of finding the answer and returning the answer. Finding the answer potentially requires multiple hops within each DHT ring and at different levels. Returning the answer can be done in one overlay hop. The latency of this overlay hop depends on the level where the answer was found. With the general definition of the expected value, it follows:

$$\begin{aligned} E[L] &= E[L_1] + E[L_2] = \sum_{i=1}^{m} \left(P(Y_i) \cdot E[L_1|Y_i]\right) + \sum_{i=1}^{m} \left(P(Y_i) \cdot \bar{l}_i\right) \\ &= \sum_{i=1}^{m} \left(p_i \cdot \bar{x}_i\right) + \sum_{i=1}^{m} \left(p_i \cdot \bar{l}_i\right) \\ &= \sum_{i=1}^{m} \left(p_i \cdot \sum_{j=i}^{m} \left(\bar{l}_j \cdot \bar{h}_j\right) + p_i \cdot \bar{l}_i \right) \end{aligned} \tag{2}$$

The same equation can be applied to MDHT and HSkip by using different functions for $\bar{h}_i$, which we will derive in the next sections. First, we derive p_i and $\bar{l}_i$, which apply to both MDHT and HSkip.

p_i depends on the *neighborhood effect*. We call the probability that an item that is registered at a specific level is requested the *level probability (LP)*. For simplicity, we assume the same LP value for all levels, except the top level. At the top level, all remaining requests are resolved or fail if the object is globally unknown (compare Eq. 3). LP describes the probability for requesting "local" content. For example, $LP = 0.3$ means that 30% of all requests at the lowest level $m+1$ are for objects registered at level $m+1$, again 30% of the requests reaching the next higher level m are for objects registered at level m, and so on. Note that requesting an item registered at level i results in a hit at level i. Hence, the probability p_i can be modeled based on a truncated geometric probability distribution[7]. The probability for a hit at the AN level ($i = m+1$) is LP. The probability for a hit at the next higher level m is $LP \cdot (1 - LP)$, and so on. At the top level ($i = 1$), all remaining answers are found as the top level stores all object IDs. Assuming a limited number of levels in a real-world system, it follows:

$$p_i = P(Y_i) = \begin{cases} LP \cdot (1 - LP)^{m+1-i} & \text{for } 2 \leqslant i \leqslant m+1 \\ 1 - \sum_{j=2}^{m+1} p_j & \text{for } i = 1 \end{cases} \tag{3}$$

The average hop latency $\bar{l}_i$ at level i is generally influenced by *network queueing*, *transmission*, and *propagation delay*. In consequence of the over-dimensioning adopted in most backbone networks, the queueing delay can be neglected. The same applies to the transmission delay due to high bit rates and the small packet size of resolution messages. The propagation delay in an infrastructure system mainly depends on the speed of light and cable material and can be approximated as 200 000 km/s. We assume an average hop distance of 10 000 km at the global level (i.e., $\bar{l}_{max} = 50$ ms), 1000 km for national networks (i.e., 5 ms), and 100 km for regional networks (i.e.,

7 For simplicity reasons, we assume independent probability variables representing the level probability, although some correlation might be possible between the levels.

0.5 ms). These assumptions are consistent with P2P latency measurements by, e.g., Steiner and Biersack [120], when subtracting the last mile latency. Based on these assumptions, we model an exponential relationship of the average hop latency $\bar{l}_i$ at level i as:

$$\bar{l}_i = \sqrt[m]{\bar{l}_{max}^{\,m+1-i}} \tag{4}$$

This represents a good approximation for a limited number of levels as expected for an MDHT and HSkip system.

5.7.2 *Independent MDHT*

The average number of overlay hops $\overline{h}_i$ within an MDHT resolution domain at level i depends on the underlying DHT technology, which can be independently chosen by each provider. We first consider the case that the MDHT system uses $\mathcal{O}(\log n)$ DHTs at all levels. In Section 5.7.4, we evaluate optimizations using $\mathcal{O}(1)$ DHTs. With $k = n^{\frac{1}{m}}$, it holds:

$$\overline{h}_{i_{MDHT}} = \mathcal{O}\left(\log\left(k^{m+1-i}\right)\right) = \mathcal{O}\left(\log\left(n^{\frac{m+1-i}{m}}\right)\right) = \frac{m+1-i}{m}\,\mathcal{O}(\log n) \tag{5}$$

In the worst case, the object ID is only known at the global level or completely unknown. In both cases, all levels have to be consulted, resulting in an average number of overall required hops $\overline{h}_{W_{MDHT}}$ of $\mathcal{O}(m \cdot \log n)$ for the independent MDHT in this worst case:

$$\begin{aligned}
\overline{h}_{W_{MDHT}} &= \sum_{i=1}^{m} \mathcal{O}\left(\log\left(n^{\frac{m+1-i}{m}}\right)\right) = \mathcal{O}(\log n) \cdot \sum_{i=1}^{m}\left(\frac{m+1}{m} - \frac{i}{m}\right) \\
&= \mathcal{O}(\log n) \cdot \left(\frac{m\,(m+1)}{m} - \frac{m\,(m+1)}{2m}\right) \\
&= \frac{m+1}{2} \cdot \mathcal{O}(\log n) \subset \mathcal{O}(m \cdot \log n)
\end{aligned} \tag{6}$$

5.7.3 *HSkip and Entangled MDHT*

HSkip only has to perform the *numeric forwarding* of the SkipNet CLB process because routing at every HSkip level by definition starts at a node with the correct string ID prefix (Section 5.3.3.2). In numeric forwarding, SkipNet searches the node with the longest prefix shared with the requested ID. SkipNet iteratively increases the shared prefix by one digit in a *constant number of hops*, which leads to SkipNet's $\mathcal{O}(\log n)$ expected number of hops as node IDs have $\log n$ digits.

HSkip iteratively performs the numeric forwarding at each HSkip level until the requested ID is found. However, in contrast to the *independent* MDHT system, the forwarding steps performed at level i also bring the request closer to the responsible node at level $i-1$ (Section 5.3.3.2). The same is true for the *entangled* MDHT (Section 5.3.2).

For HSkip, the expected number of hops per level depends on the average number of shared prefix bits that can be achieved at each level. Due to the uniformly distributed numeric node IDs, on average $n/2^i$ of n nodes have a shared prefix of i digits with the searched ID. Consequently, with n_i nodes at level i, HSkip can make a progress of $\log_2 n_i$ digits on average at level i. When repeating the same process at the next higher level $i-1$, HSkip starts this process at a node with a shared prefix of $\log_2 n_i$ digits and forwards to a node with a shared prefix of $\log_2 n_{i-1}$ digits. Hence, at level $i-1$, only the remaining $\log_2 n_{i-1} - \log_2 n_i = \log_2\left(\frac{n_{i-1}}{n_i}\right) = \log_2 k$ digits of the ID have to be changed, leading to the following average number of overlay hops at level i:

$$\overline{h}_{i_{\text{HSkip}}} = \log_2 k = \log_2\left(n^{\frac{1}{m}}\right) = \frac{1}{m} \cdot \log_2 n \tag{7}$$

In the worst case, HSkip has to perform the numeric forwarding at each HSkip level. In this case, the expected number of overall required hops $\overline{h}_{W_{\text{HSkip}}}$ is given as:

$$\overline{h}_{W_{\text{HSkip}}} = \sum_{i=1}^{m} \frac{1}{m} \cdot \log_2 n = \log_2 n \in \mathcal{O}(\log n) \tag{8}$$

The *entangled*-MDHT forwarding works similarly to the HSkip forwarding as explained in Section 5.3.3.2. The main difference lies in the intra-domain forwarding, which depends on the underlying DHT system. Assuming an intra-domain forwarding of $\mathcal{O}(\log n)$, considerations similar to the HSkip analysis also lead to an expected latency for the *entangled* MDHT system of $\mathcal{O}(\log n)$ in the worst case that all levels have to be consulted. Hence, HSkip and the entangled MDHT can be expected to show a better latency than the independent MDHT.

5.7.4 *Analysis Results*

In the following result plots, we set the DHT-specific $\mathcal{O}$ constants to "1" and use the binary logarithm, i.e., $\mathcal{O}(\log n)$ equals $\log_2 n$. Note that several DHT systems can do better than $\log_2 n$, e.g., Chord has an average path length of $\frac{1}{2}\log_2 n$ [112], so our analysis results can be seen as an upper estimate, i.e., better results can be achieved depending on the utilized DHT system.

To get a better understanding of the trade-offs between system efficiency and resolution domain autonomy, we focus on comparing HSkip with the *independent*-MDHT inter-domain forwarding approach subsequently. The entangled MDHT approach is not shown here but behaves similar to HSkip as detailed in Section 5.7.3. Although 10^6 nodes should be sufficient for a world-wide MDHT/HSkip system as estimated in Section 5.6, we evaluate a system with up to 12 million nodes, equaling the current number of world-wide DNS nodes to investigate scalability.

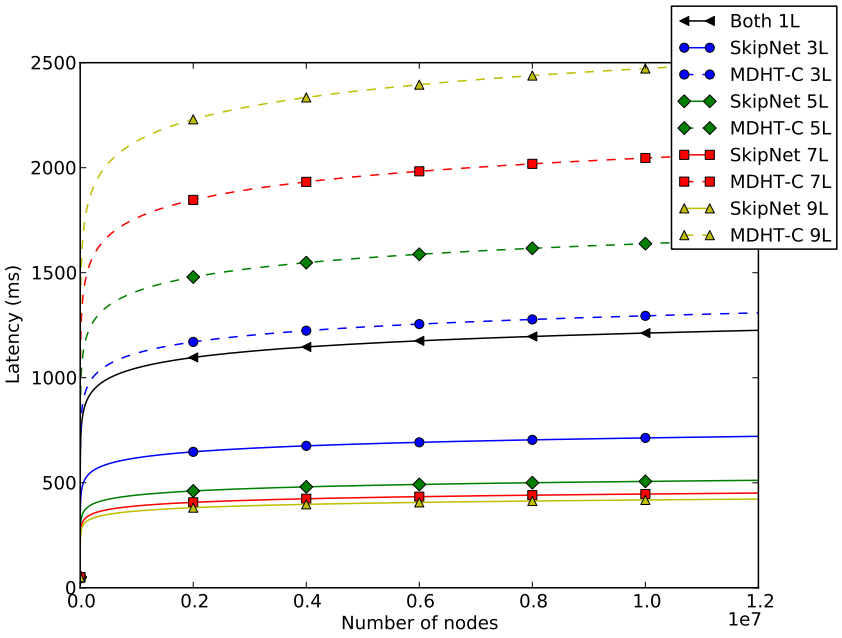

Figure 32: Analysis: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 1–9 levels (L); LP=0; dashed line = MDHT, solid = HSkip

Figure 32 shows the average latency of MDHT (with $\mathcal{O}(\log n)$ DHTs, denoted as "MDHT-C"[8] in the figure) and HSkip (denoted as "SkipNet") with up to 12 million nodes and a varying number of levels (3L, 5L, 7L, 9L) based on the preceding analysis. It compares the results to the latency of a flat $\mathcal{O}(\log n)$ DHT system which equals an MDHT or HSkip system with 1 level (denoted as "Both 1L" in the figure). Figure 32 assumes no neighborhood effect, i.e., LP is zero. In this case, the MDHT latency is generally higher than the latency of a flat DHT and of the HSkip system. In contrast, HSkip shows much better latency than a flat DHT system. It is interesting to note that the MDHT latency *increases* with an increasing number of levels whereas the HSkip latency *decreases* with an increasing number of levels for

8 The "C" indicates the usage of Chord. This is for consistency reasons as our simulation uses Chord as resolution domain DHT.

LP=0. The next figures show that this behavior is strongly influenced by the level probability, which will be discussed below.

Figure 33 shows the same analysis results for LP = 0.3. Increasing LP to 0.3 significantly decreases both the MDHT and the HSkip latency due to the higher probability to find a hit at a lower level. Both MDHT and HSkip now have lower latencies than a flat DHT system. The latency of the flat DHT system (not shown here for a better y-axis scale) is independent of the level probability, hence, is identical to the graph shown in Figure 32 ($\approx$ 1200 ms with 12 million nodes). With 12 million nodes, 9 levels, and LP=0.3, MDHT has an average latency of $\approx$ 240 ms and HSkip has an average latency of $\approx$ 50 ms. Interestingly, for LP = 0.3, the MDHT latency is now *decreasing* for an increasing number of levels. We will evaluate this effect in more detail in the following.

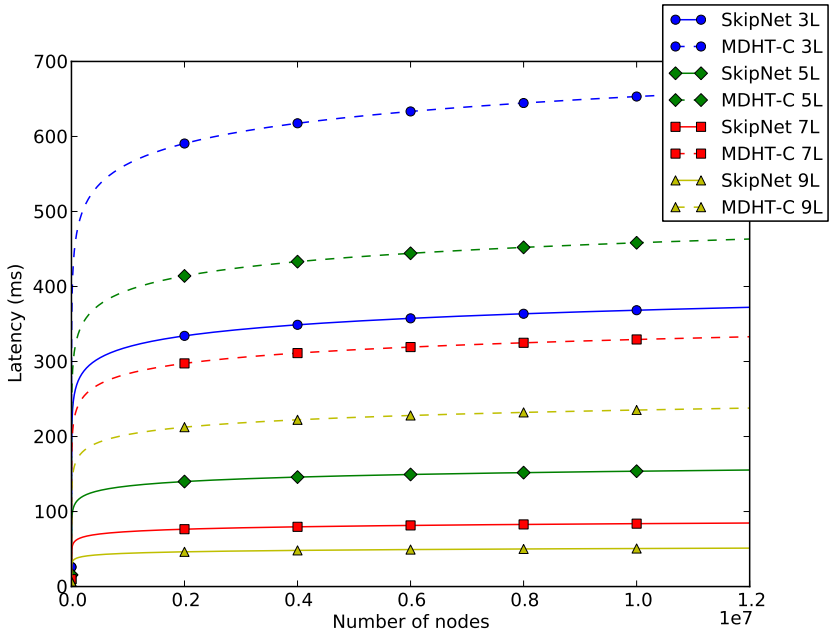

Figure 33: Analysis: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 3–9 levels (L); LP=0.3; dashed = MDHT, solid = HSkip

The influence of the level probability on latency and system characteristics is illustrated in more detail in Figure 34. It reveals an interesting aspect of the *independent* MDHT system. The overall system characteristics change depending on the magnitude of the level probability. For a small LP value, the average latency is *increasing* with an increasing number of levels. But for a larger LP value, the latency is *decreasing* with more levels. This effect can be explained as follows. If LP is small, the likelihood that all levels have to be consulted is high (Eq. 3). Hence, adding more levels increases the number of overlay hops. The latency is *increasing* with the number of MDHT levels.

However, for a larger LP value, the likelihood that all levels have to be consulted decreases. Here, the slightly increased number of overlay hops is overcompensated by the fact that more levels result in a more fine-grained topological embedding with lower per-hop latencies at the lower levels (Eq. 4), resulting in an overall reduced latency. The LP intersection point where the system characteristics change mainly depends on the resolution-domain DHT technology (e.g., $\mathcal{O}(\log n)$, $\mathcal{O}(1)$) and any constants and lower-order terms neglected in the Landau ($\mathcal{O}$) notation.

Note that this effect is unique for the *independent* MDHT and does not occur in HSkip and the *entangled* MDHT. In the independent MDHT, each level separately requires $\mathcal{O}(\log n)$ hops. In contrast, the entangled MDHT and HSkip levels cooperate. A request is getting closer to the searched ID with each level as analyzed in Section 5.7.3. Therefore, higher levels do not add as many additional hops and the more fine-grained topological embedding dominates the behavior for all LP values.

The fact that the latency decreases with more levels has strong implications on the system design. The hierarchical NRS can be adapted to the network topology in a fine-grained manner with more levels while at the same time benefiting from a decreased latency.

Figure 34 also illustrates the strong influence of the level probability on the overall latency. Both MDHT's and HSkip's latency is significantly reduced with an increasing LP. For comparison, recall that our DNS analysis (Chapter 4) shows a level probability between 0.41 and 0.71. For a level probability between 0.41 and 0.71, a system with 9 levels shows the lowest latency for both MDHT and HSkip in our evaluation.

There are two more options to reduce the average latency for both HSkip and MDHT. First, the MDHT system can use $\mathcal{O}(1)$ DHTs in the lower-layer resolution domains, which is feasible due to the relatively small and stable number of nodes within each resolution domain. HSkip can achieve a similar effect by increasing the number of references stored in the SkipNet routing table to achieve a constant hop count within each resolution domain. Second, MDHT and HSkip can use the REX system (Section 5.4) at the top level instead of an $\mathcal{O}(\log n)$ DHT (which requires a relatively high number of overlay hops) as assumed in the preceding MDHT evaluation. The REX system aggregates entries based on name prefix information and caches these entries at level 2 of the NRS system. Hence, a lookup in the REX system requires $\mathcal{O}(1)$ hops to find the cached aggregated entry at level 2, one global hop to get to the resolution domain responsible for this owner's objects, and $\mathcal{O}(1)$ hops within this domain to find the object. Therefore, both MDHT and HSkip can achieve a constant lookup latency (i.e., independent of the total number of nodes) when using the

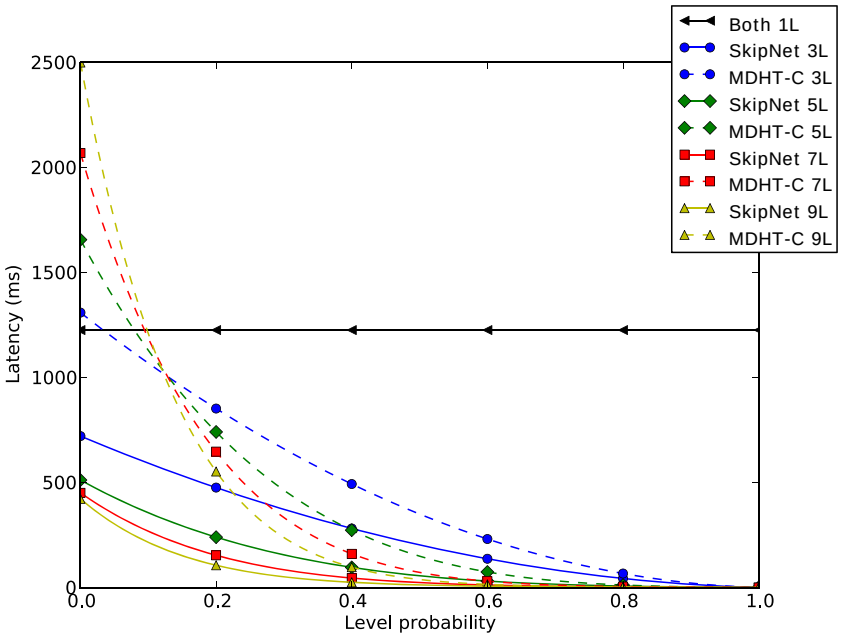

Figure 34: Analysis: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 1–9 levels (L); 12 million nodes; dashed = MDHT, solid = HSkip

REX system and constant-hop resolution domains, i.e., resulting in a *hierarchical constant-hop NRS system*.

For the *independent* MDHT, the effect of combining REX at the top level and $\mathcal{O}(1)$ DHTs at the lower levels is shown in Figure 35. This "full-blown" MDHT system achieves a latency comparable to and in most cases better than HSkip (without REX and constant-hop resolution domains) and much better than a flat DHT. The *independent* MDHT achieves an average latency of well below 200 ms for any number of levels and any level probability compared to an average latency above 1200 ms for a flat DHT. The latency is also smaller than a flat $\mathcal{O}(1)$ DHT (equaling the *MDHT-O 1L* graph) for LP$\geqslant 0.1$.

5.8 SIMULATION

In the following, we focus again on the *independent* MDHT system compared to HSkip. We evaluate the average request latency, the number of levels a request has to consult before a hit is found, and the load distribution among the NRS nodes. We consider the following main factors: total number of NRS nodes and client nodes, total number of levels, and the level probability. We start with a brief overview of the simulation setup and the assumed load model. Some results in this section are based on and some of the description is taken from work presented in reference [121].

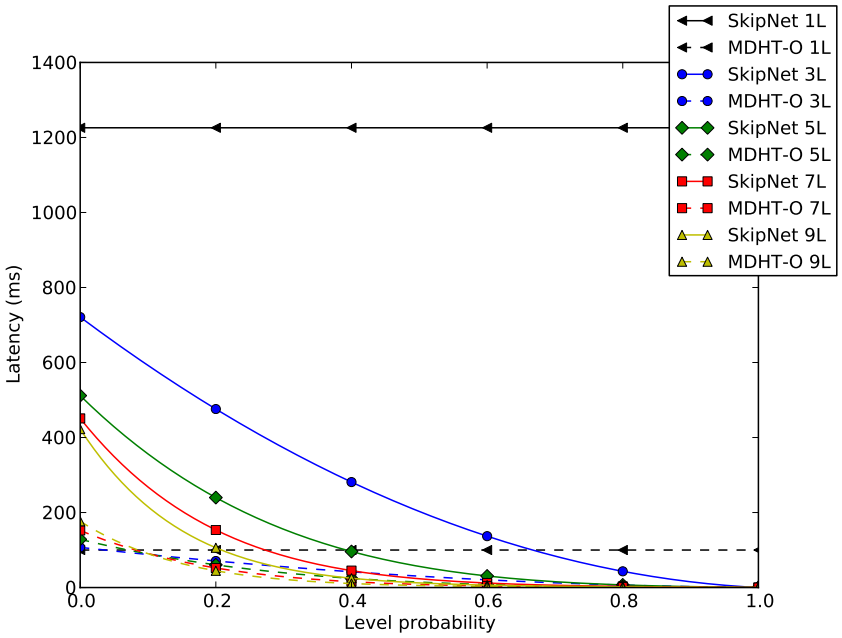

Figure 35: Analysis: Latency of REX+MDHT-O ($\mathcal{O}(1)$) and HSkip; 1–9 levels (L); 12 million nodes; dashed = MDHT, solid = HSkip

5.8.1 *Simulation Setup and Assumptions*

The simulation is based on OMNeT++[9], a generic simulation framework. For implementing the P2P aspects of our two NRS systems, we rely on the OverSim[10] framework that runs on top of OMNeT++. We use the OverSim Chord implementation for the internal DHTs in MDHT and have built our own SkipNet implementation as basis for HSkip. Both NRS implementations build on an underlay network that simulates exponentially decreasing latencies on the different hierarchical levels with a maximum latency of 50 ms at the top level (Eq. 2). We simulate up to 1500 MDHT/HSkip NRS nodes and every NRS node acts as an access node for 3 clients on average, resulting in approximately 4500 client nodes. The number of NRS nodes is constant during the simulation, i.e., we simulate no churn as both MDHT and HSkip are intended to be infrastructure networks, hence, churn plays a minor role. Prior to our measurements, each node registers on average 300 objects in the NRS, resulting in approximately 450 000 overall objects. Objects can change during the simulation similar to web pages. The time between changes is exponentially distributed following the observed distribution of inter-change times of web pages [122]. Similarly, the lifetime of copies cached on user nodes also follows an exponential distribution conforming to the lifetime distribution of P2P nodes and data in P2P networks [123].

9 http://www.omnetpp.org
10 http://www.oversim.org

During our measurements, all clients request objects following a web browsing model as described by Walters [124] with Weibull-distributed inter-session times of browsing sessions and user request inter-arrival times and a lognormal-distributed number of web requests per session and client request inter-arrival time. To simulate the neighborhood effect, we use a truncated geometric probability distribution based on the assumptions in Section 5.7.1. Object popularity follows a Zipf-like probability distribution similar to the popularity of web pages [125], i.e., the popularity of the *i*-th object is proportional to $1/i^{\alpha}$ with α between 0.7 and 1.0. All following figures show confidence intervals at a 95% confidence level which, however, are in most cases too small to be visible.

In the simulation, we chose to use a slightly different implementation of the neighborhood effect. We still calculate the hit probability p_i according to Eq. 3 but subsequently choose an object from *all* objects registered in the respective domain at level i, including all subdomains. This approach allows us to use information about the level probability from real-world measurements when the exact distribution of requests to subdomains is unknown.

5.8.2 *Results: Latency*

Figure 36 illustrates the influence of the number of nodes and the number of levels on the average MDHT and HSkip latency with LP=0.3. Unsurprisingly, the latency increases with an increasing number of NRS nodes as more NRS nodes result in more required hops in both MDHT and HSkip. The increase is sub-linear due to the $\mathcal{O}(\log n)$ characteristics of both Chord (which is underlying the MDHT system) and Skipnet (which is underlying the HSkip system). More importantly, for both the (independent) MDHT and HSkip, the latency is *decreasing* with an increasing total number of levels, confirming the previous analysis results. The HSkip latency is generally lower than the *independent*-MDHT latency for the same number of levels and nodes. With 9 levels, 1500 nodes, and LP= 0.3, the independent MDHT and HSkip have a latency of approximately 85 ms and 38 ms, which is consistent with the analysis results shown in Figure 33.

Figure 37 confirms the strong influence of the level probability on latency. The latency is significantly decreasing for both MDHT and HSkip with an increasing LP. For example, when increasing LP from 0 to 0.2, 0.4, and 0.6, the average latency of an MDHT system with 7 levels is decreasing by 48%, 86%, and 97%. The average latency of a similar HSkip system is decreasing by 46%, 81%, and 95%, respectively. Using the results from our DNS measurements in Chapter 4 ($0.41 \leqslant \mathrm{LP} \leqslant 0.71$) would result in average latencies of, e.g., 10–65 ms for the independent MDHT and 10–30 ms for HSkip for a system with

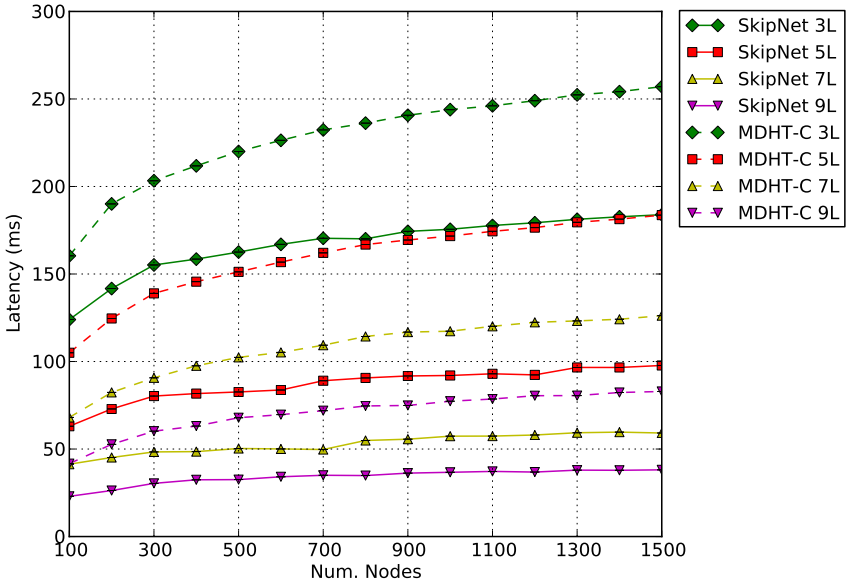

Figure 36: Simulation: Latency of MDHT ($\mathcal{O}(\log n)$ DHTs) and HSkip; 3–9 levels (L); LP=0.3; dashed = MDHT, solid = HSkip

7 levels and 1500 NRS nodes. Again, the HSkip latency is generally lower than the independent-MDHT latency. In addition, Figure 37 confirms the LP-dependent change of the MDHT system characteristics (i.e., increasing/decreasing latency with increasing/decreasing number of levels) as analyzed in Section 5.7.4.

To get a better understanding of the latency behavior, Figure 38 shows the number of levels L_{avg} required on average to answer a specific name resolution request *relative* to the total no. of levels L_{total}, i.e., L_{avg}/L_{total}. Note that zero used levels refers to hits directly at the access node in this figure, i.e., the access node is excluded in the level count.

As expected, the number of required levels decreases with an increasing level probability as more hits can be found at lower levels, hence, reducing the load at the top level. LP= 0.0 represents the case where we assume no level probability at all. Nevertheless, all presented cases only have to consult at most 72% of all levels on average; i.e., even with LP= 0.0, many requests can be answered at levels below the top level. This is due to the hierarchical structure of both NRS approaches. In both approaches, the requested information is also stored in the lower layers, providing inherent redundancy. Hence, many requests can already be answered before reaching the top level, thereby reducing the overall latency and the load at the top level even without assuming any neighborhood effect.

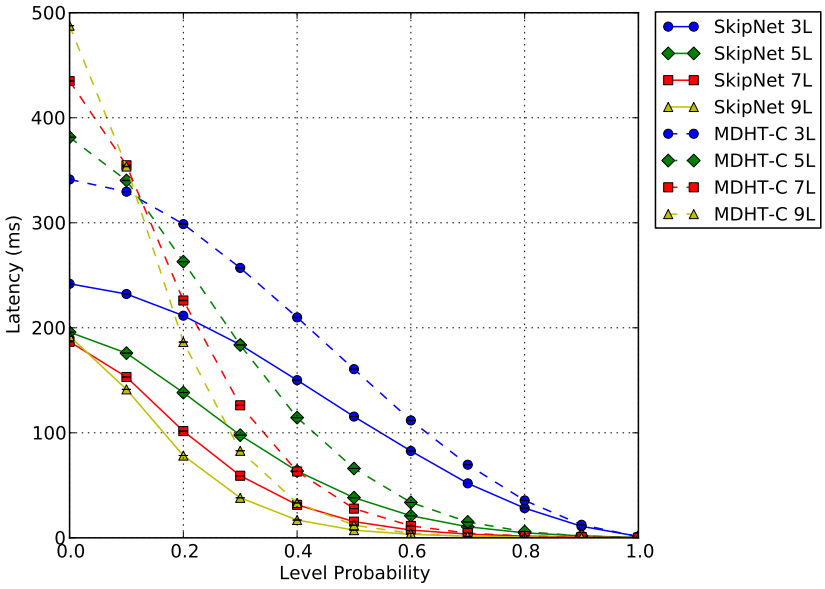

Figure 37: Simulation: Latency MDHT ($\mathcal{O}(\log n)$), HSkip: 1500 nodes; 3–9 levels (L); dashed = MDHT, solid = HSkip

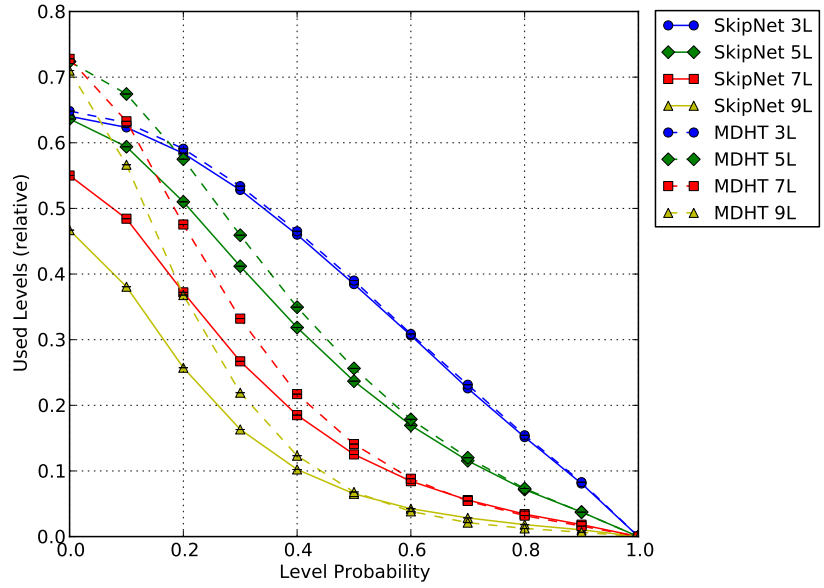

Figure 38: Simulation: Relative number of used levels; 1500 nodes; 3–9 levels (L); dashed = MDHT, solid = HSkip

5.8.3 *Results: Work Load Distribution*

Figure 39 and 40 show histograms of the load distributed over the independent-MDHT nodes with LP=0.0 and LP=0.3. Figure 41 and 42 illustrate the same for the HSkip nodes. We have used 7 levels, 1000 NRS nodes (all of which participate in all levels as virtual nodes, resulting in 7000 virtual NRS nodes), and 3000 client nodes. The figures show the accumulated number of requests processed by each "physical" NRS node. A request is processed once at each level that it visits. Forwarding of requests inside the DHT underlay is not counted.

For both MDHT and HSkip, increasing the level probability from LP= 0.0 to LP= 0.3 decreases the variance σ and shifts the mean μ from $\mu \approx 43\,000$ to $\mu \approx 33\,000$. This is due to the fact that a higher LP reduces the number of levels that a request has to visit as discussed previously and, hence, also reduces the average number of processing operations for each request. Consequently, an increased level probability has two positive effects on the system load: It improves the load distribution and reduces the overall NRS load.

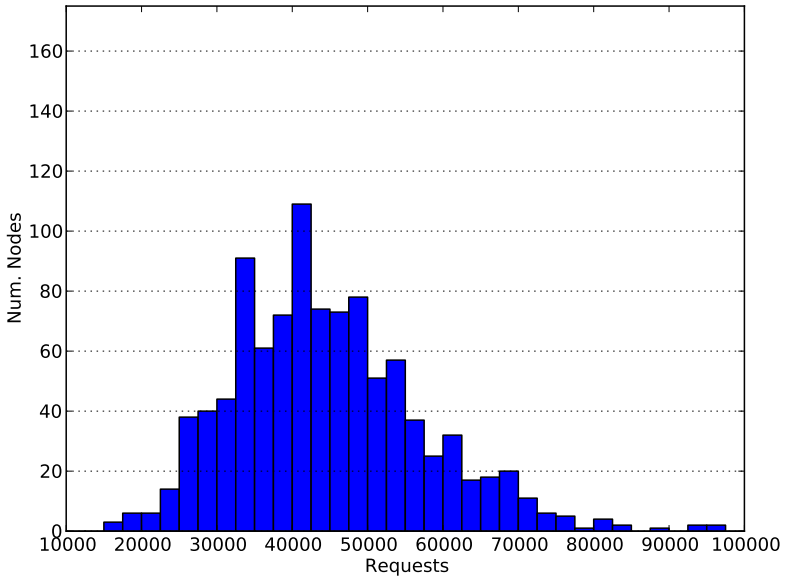

Figure 39: No. of requests per node: MDHT, LP=0

5.9 RELATED WORK

Because of our flat-namespace requirement, we cannot rely on name resolution approaches that require hierarchical names for aggregation

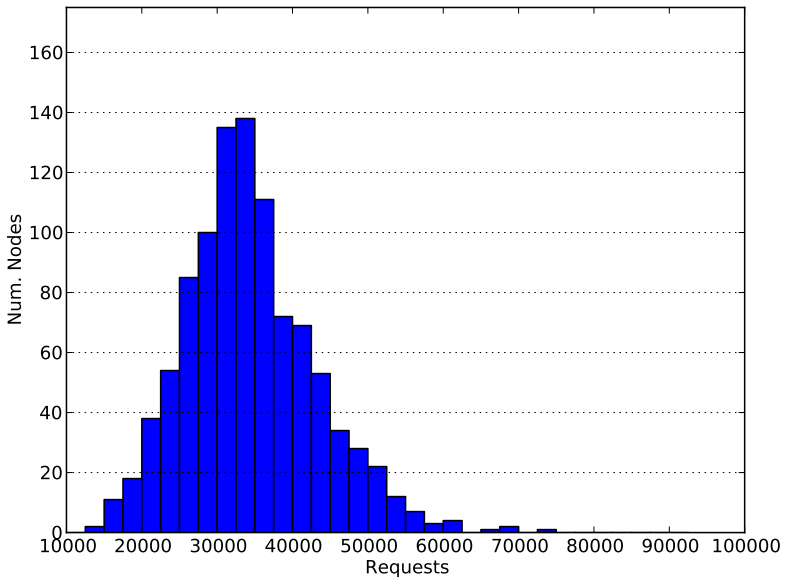

Figure 40: No. of requests per node: MDHT, LP=0.3

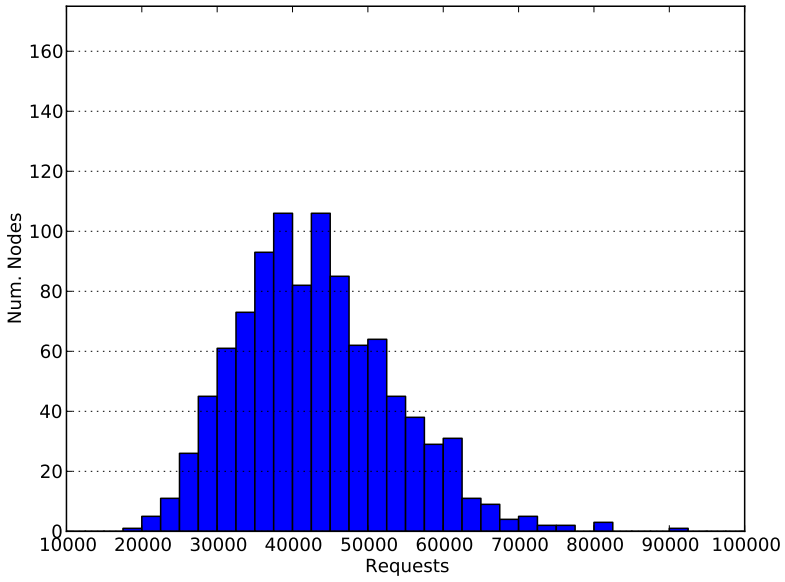

Figure 41: No. of requests per node: HSkip, LP=0

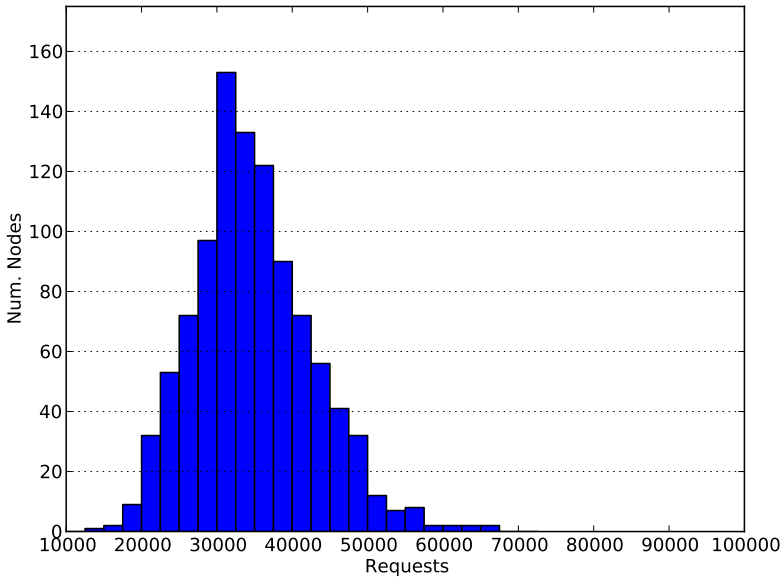

Figure 42: No. of requests per node: HSkip, LP=0.3

to scale, e.g., like today's DNS. *Flat*, structured DHT systems such as Chord [112], Pastry [61], Tapestry [126], CAN [127], and Kademlia [128] are scalable solutions for handling flat namespaces and provide desirable properties like robustness, self-configuration, and self-maintenance. NRS approaches such as CoDoNS [129], DDNS [130], SFR [131], and LISP-DHT [132] use such flat DHTs to build an NRS that meets some of our requirements. For example, LISP-DHT modifies Chord to give object owners control over the resolution of their objects' names, a property that is also relevant for an ICN NRS. However, the main problem with NRS approaches based on *flat* DHT systems, as also concluded by Cox et al. [130], is that they suffer from high latencies when distributed globally. Even more important, it is difficult in a *flat* DHT to perform efficient information dissemination by selecting close-by copies (i.e., locality-aware resolution) as the DHT typically has no topological information and is not topologically embedded. Hence, additional topological information would be required, potentially from a separate "oracle" as proposed, e.g., by ALTO [48] and P4P [49].

Alternatively, the underlying network layer could forward to a close-by copy, e.g., via IP anycast. If all servers of identical object copies would have the same IP address, we could use IP anycast to find the closest copy in a similar way as proposed, e.g., for interconnecting CDN caches via IP anycast [133, 134]. However, the object copies in an ICN are typically distributed on servers with various different IP addresses, hence, IP anycast is not a feasible solution to our problem.

To solve these problems, our NRS framework is based on a *hierarchical DHT (HDHT)* structure that is topologically embedded in the underlying network, thereby enabling locality-aware resolution. Several HDHT systems [135] have been developed previously. These HDHTs like Cyclone [136], the Generic Hierarchical DHT Framework [137], Hierarchical Rings [138], Canon [139], and SkipCluster [140] use the hierarchy to reduce resolution latency and increase robustness of the DHT. Our hierarchical DHT systems also benefit from reduced latency and increased robustness. However, our main focus is on exploiting the hierarchy to build an NRS that increases the network efficiency by returning close-by locators to requesters. More specifically, we define a common way for MDHT and HSkip how to make use of the hierarchical structure for registering and requesting name–locator bindings so that close-by locators are returned automatically.

Using hierarchies for efficient name resolution has a proven history, including DNS. DONA and PSIRP also use a hierarchical approach, however, not for name resolution but for name-based routing. Their setup has some similarities with our system (as well as with DNS) in the way that they exploit the hierarchy for subsequently searching from closer to further apart. However, we focus on name resolution. In addition, both our NRS systems are fully based on a DHT approach and we provide the Resolution Exchange (REX) system at the top level to improve scalability and latency.

The MDHT system is a heterogeneous DHT that consists of multiple separate resolution domains that can each use a different DHT protocol internally such as Chord or Pastry. These protocols are able to route messages typically in $\mathcal{O}(\log n)$ routing steps, with compact routing tables of $\mathcal{O}(\log n)$ states, where n is again the number of nodes. Alternatively, MDHT can also use $\mathcal{O}(1)$ DHTs (e.g., Structured Superpeers [141], Beehive [142]) within resolution domains to further reduce latency.

HSkip is based on SkipNet [109] and constructs a hierarchical DHT based on a single SkipNet network. The original SkipNet provides several locality properties based on SkipNet's *string* IDs. However, these string identifiers are structured and do not match the flat ICN namespaces. Flat ICN IDs are more similar to SkipNet's *numeric* IDs. Hence, among other changes, HSkip extends SkipNet to provide the same locality properties for SkipNet's *numeric* IDs.

5.10 SUMMARY

We have developed and evaluated three different incarnations of our hierarchical NRS framework: HSkip, the *independent* MDHT, and the *entangled* MDHT. These three systems demonstrate the inherent trade-off between latency, maintenance overhead, and memory con-

sumption on the one hand and resolution-domain autonomy on the other hand.

The results presented in this chapter indicate that our hierarchical NRS framework offers a suitable solution for global name resolution of flat namespaces. We estimate that we need approximately 10^6 NRS nodes for a world-wide system. Our analysis with up to 12 million NRS nodes as well as our simulations illustrate the systems' scalability. Even with 12 million nodes, the *independent* MDHT shows an average request latency of below 100 ms with 9 levels and a modest level probability of 0.4. The *entangled* MDHT and HSkip show an even better latency of about 25 ms. Higher level probabilities and using constant-hop DHTs in MDHT can further reduce these latencies. In addition, using our REX system at the top level also improves these latencies and global scalability of all presented NRS systems. The fine-grained topological embedding of the NRS results in significantly reduced load at the global resolution level, even when assuming no neighborhood effect. In addition to the presented analysis and simulation, we have published an open source implementation of the MDHT system[11]. This implementation is evaluated in Chapter 6.

The neighborhood effect has turned out to have a major influence on the overall system behavior, with higher LP values having significant positive influence on the average latency as well as on the overall load and load distribution. The DNS analysis in Chapter 4 has shown that a level probability between 0.41 and 0.71 can be observed in some domains today. The domains that we investigated in Chapter 4 correspond to the two lowest levels in our hierarchical NRS, i.e., these domains would become the two lowest levels in our hierarchical NRSes.

We believe that resolution-domain autonomy is important to simplify migration and deployment. It not only gives providers autonomy to freely choose their intra-domain architecture and technology but also simplifies gradually growing the system from the edges of the Internet. From this perspective, the independent MDHT system might be more favorable for a global NRS system. On the other hand, HSkip offers some inherent advantages like further reduced latency, maintenance overhead, and memory requirements. Hence, the entangled MDHT system might present an interesting compromise that offers decreased latency while providing some degree of resolution domain autonomy.

11 http://www.netinf.org

6

PROTOTYPING

This chapter is based on work published in references [44, 45, 39, 40, 41, 42, 43, 143].

This chapter evaluates the overall NetInf architecture and its main components via prototyping. For this purpose, we have built a prototype of the overall general NetInf architecture, called *OpenNetInf*. OpenNetInf is published as open source software[1]. In this chapter, we focus on the overall NetInf prototype implementation and on the development of information-centric applications to test the NetInf architecture. We discuss lessons learned, implementation alternatives, consequences for the NetInf architecture, and present testbed measurements.

6.1 INTRODUCTION

The prototype work described in this chapter has five goals:

1. *Evaluate and validate the NetInf architecture:* We evaluate the consistency and feasibility of the overall NetInf architecture, protocols, interfaces, and of the different NetInf components with a focus on naming and security, name resolution, caching, and the object model.

2. *Evaluate NetInf advantages:* We demonstrate the advantages with a wide variety of application scenarios and discuss traffic measurements.

3. *Migration:* We evaluate difficulties to migrate towards a NetInf architecture and requirements to start benefiting from the introduction of NetInf.

4. *Application development:* We focus on four main areas: *media distribution*, *legacy application support*, *context-awareness*, and *smartphone applications*.

5. *Feasibility to extend the core NetInf architecture:* We evaluate the integration of *search* and *event notification services* with the NetInf architecture.

1 http://www.netinf.org

Section 6.2 describes the OpenNetInf prototype and its components. Section 6.3 introduces our ICN applications. In Section 6.4, we present measurement results demonstrating the influence of our MDHT NRS and caching implementation on inter-domain traffic. Section 6.5 gives a brief overview of other existing ICN prototypes before we discuss our overall prototyping results and the lessons learned in Section 6.6.

6.2 OPENNETINF PROTOTYPE

In this section, we first give an overview of the overall OpenNetInf prototype architecture and subsequently describe the main components of OpenNetInf in more detail.

6.2.1 *Overview*

The OpenNetInf prototype is based on our Future Internet Toolbox (FIT), which is a framework that we have developed to accelerate prototyping of future Internet architectures. FIT is described in detail in reference [40]. The NetInf architecture can run on top of many underlying network technologies, including IP, which is currently the case for OpenNetInf. OpenNetInf is implemented in Java to be portable and is tested on FreeBSD, Linux, and Windows.

The central element of the OpenNetInf prototype is the *OpenNetInf node* (Figure 43), subsequently called *NetInf node*. A NetInf node is a process that supports the NetInf *node-to-node (N2N) interface*. It runs on all machines participating in the Network of Information. Via the N2N interface, NetInf nodes communicate with each other and offer-/consume services via their *components*. Components are parts of a NetInf node that encapsulate specific functionality like caching and name resolution. NetInf nodes can be specialized containing different components described in more detail in the subsequent sections. In general, components can be divided into the *NetInf core* and *additional components* (*AddOns*). Applications access the component functionality via the *information-centric API*.

6.2.2 *Interfaces*

As illustrated in Figure 43, a NetInf node has two different interfaces: the *information-centric API* that allows applications to retrieve, publish, modify, or delete NDOs, and the *N2N interface*, which is used for communication between NetInf nodes.

We have initially experimented with a single interface serving as API *and* N2N interface. However, a common interface either exposes too much information to applications or restricts the communication between NetInf nodes too much. Most notably, this involves object

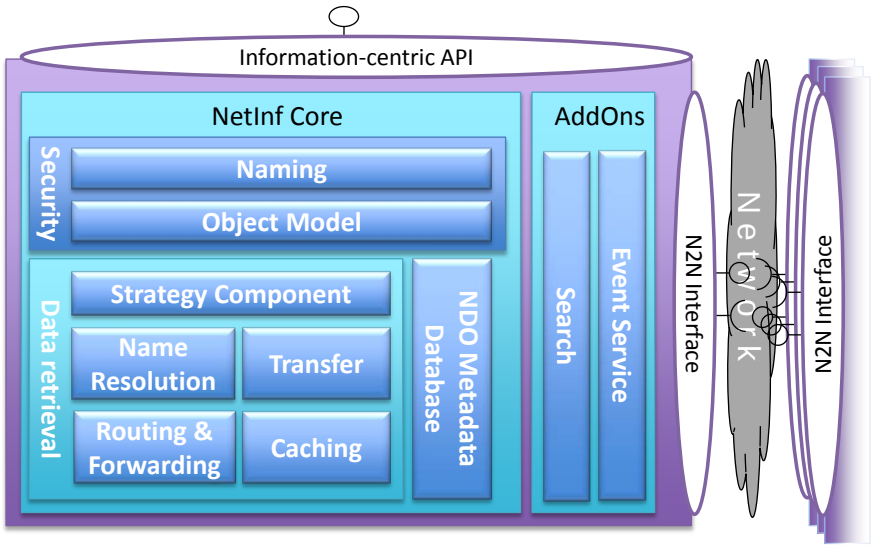

Figure 43: OpenNetInf node with *NetInf Core* and *AddOn* components, connected to other NetInf nodes via the N2N interface

locators. On the one hand, NetInf nodes have to exchange object locators when communicating with each other, e.g., as response to a name resolution request. On the other hand, applications should not be aware of object locators. Passing object locators to applications would allow applications to store such locators for future use, thereby, e.g., bypassing NetInf's mechanisms to retrieve the "best" copy depending on the current (network) context, discarding the advantages of persistent object names, producing problems with object mobility, etc.

Support for legacy applications, i.e., enabling legacy applications to utilize the information-centric network, is very important for our OpenNetInf prototype and similar network prototypes. Most importantly, it is critical to simplify migration. In addition, from a prototype point of view, support for legacy applications significantly simplifies rapid development of test applications. The goal is to provide legacy applications with a means to easily access the OpenNetInf information-centric API in a way that is already supported by legacy applications. Hence, the OpenNetInf prototype offers an HTTP-based gateway to access the information-centric API, i.e., the HTTP communication of legacy applications can be redirected to talk to an OpenNetInf node.

We have found that an HTTP-based approach is a very good choice as it has two major advantages. First, as the requests are ordinary HTTP requests from an application point of view, many legacy applications supporting HTTP requests can natively run on top of OpenNetInf. Second, this is a common approach that application developers are already familiar with, significantly reducing the learning curve to write new applications using OpenNetInf.

To facilitate this approach, we have integrated a web server into the NetInf node, which functions as a layer seven gateway between HTTP and the NetInf protocol (hidden in the *information-centric API* in Figure 43). We follow the Representational State Transfer (REST) model [144] that offers a flexible and well-known approach to access and modify objects. Our implementation supports the HTTP/1.1 methods `GET`, `HEAD`, `POST`, `PUT`, and `DELETE`. Via these methods, we can address an NDO (e.g., a video), related metadata, and we can perform a search for object IDs.

To retrieve an object via OpenNetInf, an application sends an HTTP request "`GET /bo?objID HTTP/1.1`" to the next known NetInf node, i.e., the *initial NetInf node*. The initial NetInf node would typically be a NetInf node process running on the user's local machine. However, a remote initial NetInf node is also possible. This can be very handy if the local machine cannot/should not run a NetInf node process by itself, e.g., to preserve resources. The HTTP–NetInf gateway component of the initial NetInf node takes the HTTP request and translates it into a NetInf-internal object request. Subsequently, the request is handed over to the appropriate component to process the request. Finally, the HTTP–NetInf gateway returns the answer to the requesting application. The application has to worry neither that the `GET` request contains a NetInf ID nor how NetInf retrieves the object.

Java applications do not have to use the RESTful interface but can directly use the OpenNetInf Java API. This API basically offers the same functionality but in addition provides some convenient ways to access named data objects.

While the information-centric API is responsible for communication between applications and the initial NetInf node, the N2N interface is responsible for inter-NetInf-node communication, performed by the node components. Node components communicate with other NetInf nodes via the N2N interface to fulfill their specific tasks. For example, the name resolution component communicates with other NetInf nodes to resolve an NDO ID into a network locator.

OpenNetInf can encode N2N messages in two ways: Google Protobuf[2] and XML. Protobuf provides a simple, high-performance binary encoding, allowing efficient encoding/decoding. Libraries are available for many programming languages. XML is an alternative for unsupported languages, however, XML messages are bloated and parsing requires more memory and computing power.

6.2.3 *Named Data Objects and Security*

As intended by the NetInf architecture, the naming scheme plays a major role in OpenNetInf to achieve the information-centric security goals. The prototype supports the main information-centric security

2 http://code.google.com/p/protobuf/

functionality described in Chapter 3, including *owner pseudonymity*, *owner identification*, and *name–data integrity*. Name–data integrity can be checked by the *initial NetInf node* so that not all applications have to implement name–data integrity checking by themselves. In addition, applications can check name–data integrity themselves if they decide not to trust the initial NetInf node.

OpenNetInf provides a flexible NDO metadata model. The metadata contains mandatory security-related metadata (for the naming scheme, Section 3.3.3) and optional application metadata. Choosing the right metadata format has proven non-trivial. Several factors like developer acceptance, functionality, and complexity have to be taken into consideration. We have experimented with several formats, including RDF[3], which supports rich functionality like reasoning to infer logical consequences, but has disadvantages concerning acceptance and complexity. Alternatives include a MIME-based format as proposed in Section 2.2.1 or an XML-based format. The current implementation supports an RDF-based metadata model. Independent of the format, our object model has proven to support a wide variety of use cases defined, e.g., in reference [23]. This includes support for static and dynamic files, audio/video streams, and real-world entities, as discussed in Section 6.3.

6.2.4 *Name Resolution and Metadata Storage*

Our prototype currently implements a two-step name resolution concept where the initial NetInf node first performs a dedicated name-resolution step to resolve the name into a set of locators, and second retrieves the object via an underlying transfer protocol like HTTP.[4]

As described in Section 2.2.6 and Section 5.3.1, the NetInf architecture supports multiple NRSes that are potentially interconnected into a global NRS. Prototyping has shown that this approach supports a much wider set of use cases and simplifies migration as each network provider can host its own local NRS. The following NRS implementations currently exist: a node-local NRS to resolve names of locally stored objects into network locators, a local-network NRS based on local broadcast, an NRS for global resolution that is based on a single flat P2P network, and a global NRS based on the hierarchical Multi-Level Distributed Hash Table (MDHT) concept (Chapter 5).

Metadata is stored either in a separate metadata repository or together with the actual data. Our implementation experience has shown that closely combining the metadata with the NRS can be very useful. Hence, each OpenNetInf NRS can also store and deliver the metadata pertaining to a requested object. Thereby, we can uti-

3 http://www.w3.org/RDF/

4 The prototype is prepared for but currently does not implement a one-step approach and a name-based routing approach.

lize metadata for better informed resolution decisions, reduce latencies, and eliminate the need for a separate globally scalable metadata repository.

6.2.5 *Caching*

OpenNetInf supports two types of caches: *peer-side caches*, integrated in the initial NetInf nodes (i.e., typically user terminals), and dedicated *in-network caches* that are part of the network infrastructure. Both are conceptually similar and are based on the *Ehcache*[5] open-source library.

The OpenNetInf *peer-side cache* serves the two purposes described in Section 2.2.5: First, it serves as application-independent cache for all local NetInf applications. Second, the peer-side cache can serve as cache for other NetInf nodes.

NetInf supports both *on-path* (on the request/data path) and *off-path* caching. OpenNetInf focuses on off-path caching and how to trigger caching and find cached copies. Our implementation experience has led to the architectural decision to closely integrate off-path caches with the resolution system. This provides several advantages: The integration with MDHT has proven particularly useful because of its hierarchical structure. As described in Chapter 5, the MDHT resolution nodes in a certain resolution domain receive all requests from clients in their resolution domain (e.g., in Figure 48, node *MDHT 1* receives all requests from client 1–4 at level 3) and can collect statistics about the objects' popularity in this part of the network. Based on these statistics, the MDHT nodes can decide which objects should be cached. Global knowledge is not required for this decision as an in-network cache typically only serves requests from its local part of the network (i.e., the specific resolution domain). Subsequently, an MDHT node would trigger an off-path cache in the same resolution domain to cache the specified objects. After caching an object, the OpenNetInf in-network cache registers this copy with the MDHT node that triggered the caching process to make the copy available for all NetInf nodes within this resolution domain. As shown in Figure 48, each separate resolution domain at a level can contain its own in-network cache(s).

6.2.6 *Data Transfer*

As previously mentioned, the OpenNetInf prototype focuses on a name-resolution-based two-step approach for retrieving data. Both steps are triggered by the initial NetInf node. While the first step, the name resolution, is performed by the name resolution component,

5 http://www.ehcache.org

the second step, the actual data transfer, is performed by the data transfer component.

The data transfer component selects one or more locators from the resolved locator list and initiates the data transfer. Depending on the locator type and other aspects, it can select between multiple different transfer protocols like HTTP or the File Transfer Protocol (FTP). This allows support for many existing data transfer protocols, thereby simplifying migration. Relying on existing transfer protocols is sufficient for a fully connected network scenario. Extending NetInf towards challenging network scenarios like DTN could be done by either utilizing existing protocols like the Bundle Protocol [145] or by developing NetInf-specific transfer mechanisms. Such a NetInf-specific transfer mechanism, e.g., based on name-based routing, can be integrated in OpenNetInf as well. This would be done by skipping the name-resolution step and adding a new transfer service that performs transfer directly based on the object name.

Besides handling complete objects, the OpenNetInf prototype also supports chunking of objects, i.e., dividing larger objects into pieces and handling these pieces separately. This enables a NetInf node to simultaneously download an object from multiple sources and to perform streaming. The integrity of single chunks can also be validated. Chunking is implemented based on HTTP range requests.

6.2.7 *Additional Services*

Beyond the NetInf core functionality, we have also evaluated the integration of additional services into the NetInf architecture (Figure 43). We focus on search services and event notification services in this section.

6.2.7.1 *Search*

The goal of a NetInf search service is to find object names based on some search criteria. The user/application can send a `SEARCH` query to a search service, which returns a list of object names matching the query. OpenNetInf supports multiple kinds of search services like keyword-based search, semantic-web-based search (based on the SPARQL Protocol and RDF Query Language (SPARQL)[6]), and geographic search (i.e., searching for objects in a certain region). A search service can access the NDO's metadata as well as the NDO's main data. The implementation of a search service itself is out of scope of NetInf.

A search service can be offered by a local NetInf node, e.g., to search for locally available information, as well as by a global (i.e., Google-like) search service. A global search service is conceptually in-

6 Recursive acronym; http://www.w3.org/TR/rdf-sparql-query/

dependent of the NetInf infrastructure, hence, not adding additional requirements on the scalability of the NetInf infrastructure. Integration with the NetInf infrastructure is based on the respective API calls and N2N messages.

6.2.7.2 *Event Service*

We have integrated an event service in the OpenNetInf prototype. The event service allows users/applications to subscribe to changes of a specified NDO. This is currently implemented for the NDO metadata and allows to subscribe to changes of all metadata or selected attributes. For example, a collaborative editing application could subscribe to changes of the "last modified" attribute of a document to be informed about document changes.

The OpenNetInf event service is based on a separate notification infrastructure, hence, the scalability of OpenNetInf is independent of the scalability of the event service. Our current implementation uses the SIENA event service [67], however, OpenNetInf is independent of the specific event service implementation. This is achieved via an adapter module that we have developed that translates the OpenNetInf commands into the specific commands of the utilized event service. Via the adapter module, OpenNetInf can even support multiple different event services in parallel.

We have also implemented a closer integration of search services and the event service. For example, an OpenNetInf search service can subscribe at the event service for changes of objects that it has indexed. Thereby, it will automatically be informed about changes, enabling quicker index updates and eliminating the need for frequent polling to check for changes. This scenario is described in more detail in Section 6.3.4.1. More details about the event service can be found in the OpenNetInf documentation [146].

6.3 APPLICATION DEVELOPMENT

We have developed and tested three main application scenarios to test our OpenNetInf prototype implementation, analyze advantages, and evaluate the application development process for an ICN architecture. We have focused on *media distribution*, *legacy application support*, and *context-aware smartphone applications*.

6.3.1 *Media Distribution*

Videos currently are the main share of Internet traffic [5]. Hence, we have tested several applications for distributing and streaming video via the OpenNetInf infrastructure to demonstrate efficient dissemination of large files.

In consequence of our RESTful HTTP API, many existing streaming applications run on top of OpenNetInf out of the box, e.g., the VLC media player[7] and Firefox. Hence, no extra development is required, simplifying NetInf migration. The existing applications can retrieve the content via the underlying NetInf and directly benefit from NetInf advantages. The initial NetInf node serves as proxy between the application and NetInf. It uses the HTTP API and a common URL, which transparently encodes the NetInf object ID as described in Section 6.2.2.

Figure 44 shows video streaming in Firefox via OpenNetInf. The content URL highlighted in the figure encodes the initial NetInf node in the host part (`localhost:8181`) and the object ID in the query string (`hash_of_pk=b172494aaaf...`)[8]. Once the NetInf node receives this request, it extracts the object ID, resolves the ID into a set of locators using a suitable NRS, retrieves the data from these locations, checks name–data integrity, and streams the content back to the requesting application via its integrated web server. The transfer component responsible for the data transfer can automatically switch to new locations (which have previously been retrieved by the name resolution component) when the current locator fails, e.g., because of network congestion, server overload, or a temporarily interrupted network link. It also supports multi-source downloads. Hence, using OpenNetInf allows video applications to benefit from any available source (incl. peer-side caches and in-network caches) and can in many cases increase data availability (i.e., continued video streaming) even during challenging conditions.

Specifically, we have performed a high-level evaluation of video streaming in a network scenario with intermittent connectivity [143], using the VLC media player running on top of NetInf. During the video streaming process, the connection between the local client network and the Internet can be interrupted, including the connection to the original video streaming server. Due to the lost connectivity, the NetInf underlay automatically triggers a new name resolution process. As no global (NRS) infrastructure is reachable due to the lost Internet connection, NetInf can use a local broadcast name resolution service to find a new video source in the local network and can seamlessly switch the video connection to the new local source.

6.3.2 *InFox*

Our Firefox web browser plugin called *InFox* further demonstrates OpenNetInf's legacy support. It allows to embed *native* NetInf IDs

7 http://www.videolan.org/vlc/

8 OpenNetInf currently uses an early naming format with slightly different syntax compared to the ni URI format defined in our recent ni URI scheme RFC [28]. However, the main semantics are the same.

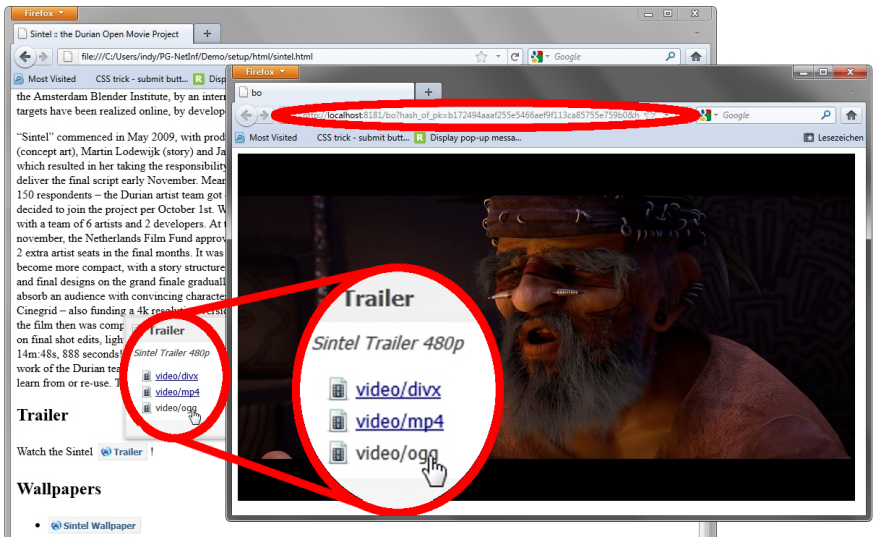

Figure 44: Video streaming in Firefox via OpenNetInf

in web pages, eliminating the need to encode the NetInf ID in the URL query string as described in Section 6.2.2[9]. NetInf IDs can be embedded in an HTML page in a format compatible to today's web links (Listing 3). These links contain a common URL as well as a NetInf ID. The links work with regular web browsers and with InFox. The InFox plugin ignores the common URL and uses the NetInf ID to retrieve the object via NetInf. A common web browser lacking the InFox plugin ignores the NetInf ID and uses the common URL instead. The compatible NetInf links are demonstrated and can be tested at `http://www.netinf.org`.

Listing 3: Compatible NetInf link (NetInf ID is curtailed)

```
<a href="http://www.netinf.org/" ni="ni:HASH_OF_PK=8
    c4e559 ...">NetInf</a>
```

In addition to linking directly to a named data object, InFox can also link to NDO metadata that shows up as pop-up window when clicked. This is very helpful, e.g., to manage different encodings of the same video. Figure 44 shows a small pop-up window that opens when the underlying link for the movie *trailer* is clicked. The pop-up window displays the NDO metadata containing links to NDOs representing the same video in different video encodings. As mentioned previously, the NDO metadata is stored in the metadata repository which is combined with the name resolution service in OpenNetInf. Hence, the metadata is returned to the requester during the name resolution process by the NRS. In this example, this *encoding-independent* NDO does not contain locators (as it does not represent any specific

9 Another alternative for legacy support is the "well-known" URI scheme [91] as described in our ni URI scheme [28].

video file) but only contains the metadata which points to several other video NDOs representing the same video in different encodings.

The user can select the appropriately encoded video in the pop-up window. This transparently triggers another name resolution step to resolve the selected NDO ID into a set of locators for this specific video file, which then downloads via the underlying NetInf and starts playing. Obviously, the appropriately encoded video version can also be chosen automatically based on the device's capabilities.

Besides the users, the main beneficiary of this mechanism is the web page maintainer. The maintainer has to link only to the *encoding-independent* NDO on the web page without having to bother about available encodings. Whenever someone creates a new encoding of this video and adds this information to the NDO metadata, the new video version will automatically be accessible via the encoding-independent NDO link. Unauthorized write access to the metadata can be identified via name–data integrity checking as described in Section 3.4.1 to prevent misuse.

The preceding example illustrates the power of NetInf's object model. Applications such as the InFox plugin can make use of the flexible application-specific metadata to improve the user experience and invent innovative features that can also easily be used by other applications as the NDOs are jointly used by applications.

6.3.3 *InBird*

Email addresses have become an important personal identifier for Internet users. However, this identifier is not persistent. Business email addresses typically change when the user changes the employer and private email addresses change when changing the email provider. Changing email addresses produces significant hassle to inform all contacts of the new email address and might even result in lost contacts.

This problem can be solved by using NetInf NDOs as persistent representations of people, called *named person objects (NPOs)*, which are a kind of dynamic digital business card. A user can create a digital business card stored as NDO with up-to-date contact information and provide his/her contacts with the persistent name of this business card NDO.

An NPO can contain a varying amount of information about the NPO's owner and access to this information can be restricted as desired. Any amount of data can be represented by the NPO. As with any NDO, the NPO ID is simply resolved into a set of locators where the actual data is stored. However, the amount of information to be stored in the digital business card is typically small. In such cases, the NPO data can also be stored directly in the NPO's metadata for

efficiency reasons. This has the advantage that NPO metadata can be directly hosted by and retrieved from the OpenNetInf NRS during name resolution, eliminating an extra step to retrieve the data from a separate location after performing name resolution.

For our use case of persistent email communication, the NPO has to contain at least one valid email address of the owner. The important point is that instead of propagating his/her current email address, the user gives his/her NPO's ID to contacts.[10] Hence, these contacts store the NPO's persistent ID as contact information in their email client's address book instead of the user's email address. Hence, when the user's email address changes, the user only has to update the email address in the NPO instead of propagating the new email address to all contacts. This is possible because NetInf IDs (including the IDs of the NPOs) are inherently persistent even in the event of a provider change.

To send an email to the user, contacts can retrieve the user's *current* email address from the user's NPO. This process can be automated transparently using a small email client plugin. We have developed such a plugin called *InBird* for the Thunderbird email client. InBird integrates with the Thunderbird address book so that Thunderbird users can continue to address emails using the recipient's name. When sending an email, Thunderbird first automatically translates the user name into the email address value stored in Thunderbird's address book. This is typically the user's email address. However, in our case, we store the NPO's ID here. Subsequently, the InBird plugin transparently translates the NPO's ID into the current email address of the email recipient using the core NetInf functionality. First, the plugin retrieves the NPO metadata corresponding to the specified ID using NetInf's name resolution mechanism. Thereafter, the plugin extracts the email address from the retrieved NPO metadata, uses this email address as destination address, and hands the email back to Thunderbird for transmission. Thereby, the InBird plugin automates the process of retrieving the current destination email address and allows Thunderbird users to directly utilize persistent NetInf IDs as substitute for email addresses. Similar plugins can obviously be provided for other email clients.

6.3.4 *Context-Aware Applications*

To test NetInf's usability not only with common applications but also with more advanced, innovative applications, we have developed two context-aware applications. Both applications make use of the addi-

10 When the NPO's ID should be communicated orally, e.g., via phone, the human-friendly form of the ni naming scheme as described in Section 3.6 becomes very useful.

tional NetInf services like the event service, search services, and the integration of both.

6.3.4.1 *Shopping Application*

We have developed a context-aware shopping scenario with integrated collaborative editing functionality that enables users to jointly manage shopping lists and informs users about available items in close-by shops. The scenario is shown in Figure 45. It makes use of NetInf's NDO object model to represent shopping information, an NDO *storage server* to store NDOs, an *event service* to subscribe to and be informed of NDO changes, and a specialized inventory list *search service* that subscribes at the event service for changes of NDOs. Obviously, the following example scenario just illustrates one possibility how this NetInf functionality can be used in a context-aware scenario.

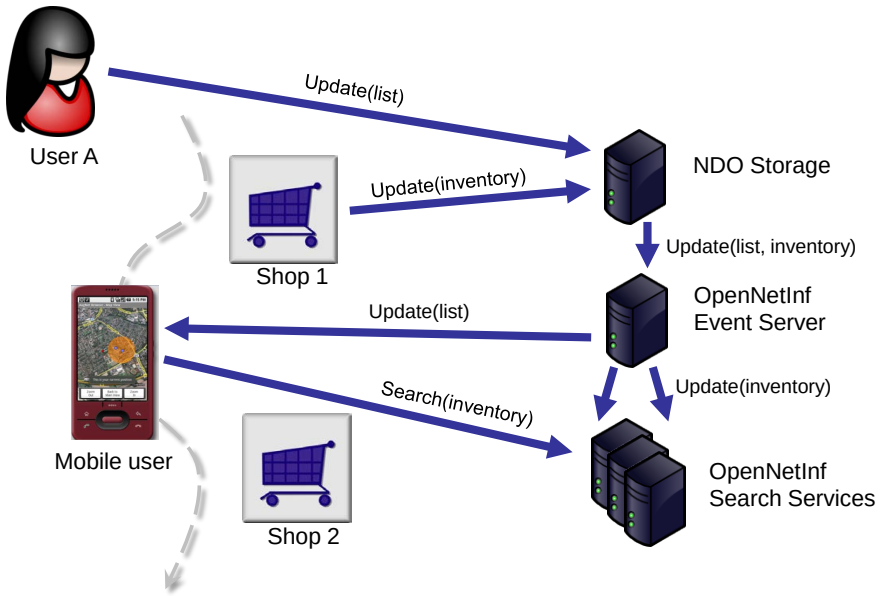

Figure 45: Shopping application use case scenario

In our example scenario, *user A* and a *mobile user* share a common shopping list, which is stored in an NDO at an NDO storage server. While the mobile user is on the go, user A can update the joint shopping list from back home (e.g., adding milk). As soon as the shopping list is updated, the *event server* is informed about these changes by the storage server. The event service informs any participant subscribed to this specific NDO about the update, which is only the mobile user in our scenario. Hence, the mobile user is automatically informed about any changes of the joint shopping list and can download the latest version. Similarly, the mobile user can also edit the joint shopping list.

The *shops* in our example scenario are using a similar mechanism to frequently update their inventory lists that are also stored in NDOs. Any inventory updates are again immediately propagated to all subscribers by the event service, in this case to some connected *search services*. Thereby, the search services can immediately update their search indexes of the shops' inventory via this push mechanism without requiring a cumbersome pull mechanism (e.g., web crawlers) as commonly used by today's search engines.

Finally, while passing by a shop, the mobile user can use his/her shopping application to automatically search the search service(s) for any item that is on his/her shopping list and contained in the shop's (up-to-date) inventory. The search can be triggered either by the user him-/herself, automatically by the mobile application based on the user's position and some knowledge about shop positions, or by the event service that can optionally be informed about the user's movements[11]. To limit results to near-by shops, the search service uses a geographic search (Section 6.2.7.1) that can use the user's geographical location and the shops' geographical locations to match local shops with the user's position.

More details about this scenario and the developed demo applications are available in reference [146].

6.3.4.2 *Augmented Internet Browser*

We have developed the *AugNet browser*, a context-aware browser that enables real-world/Internet integration based on the Augmented Internet concept [42]. This application is developed for mobile devices based on the Android operating system to test NetInf's usability for smartphone development as well as for context-aware applications. In the following, we will give a brief overview of the idea behind the AugNet browser and the Augmented Internet concept. More details can be found in references [43, 42].

The Internet contains a lot of information that can be useful to support users in real-world activities and while on the go. Unfortunately, this information is currently cumbersome to access and retrieving the information disrupts the users' workflow. Several Internet applications have been developed in recent years that provide a better real-world/Internet integration, e.g., for the iPhone and Android operating system [147, 148]. We call these applications *AugNet* applications. For example, this includes applications that provide users with information corresponding to their current surroundings and lets them (virtually) interact with local objects as illustrated in Figure 46. However, AugNet applications are currently difficult to develop on a large

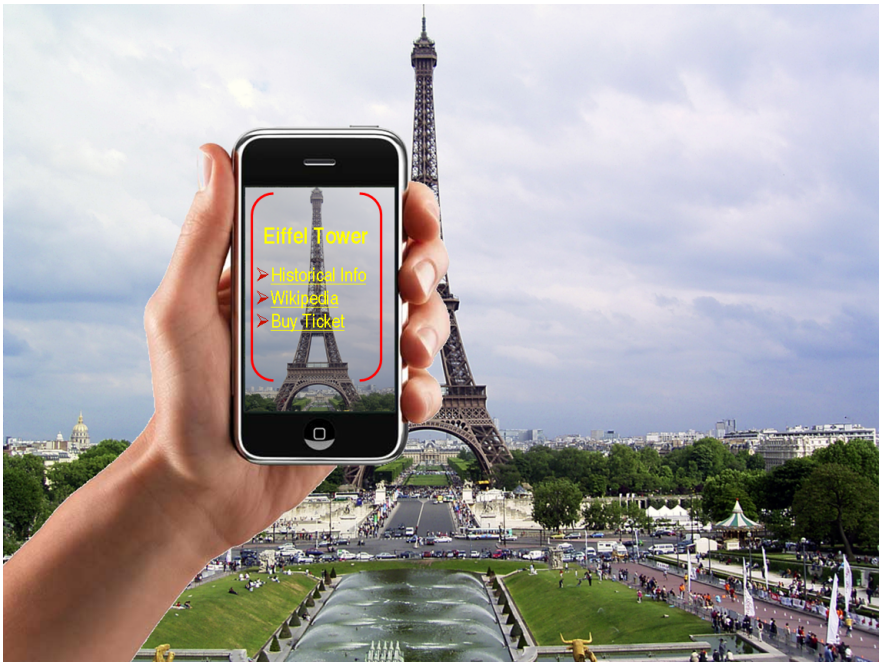

Figure 46: Mockup of AugNet application showing an information overlay for the Eiffel Tower

scale because conceptual support for such applications is missing in today's network architecture.

In contrast, the NetInf architecture inherently provides conceptual support for AugNet applications: NDOs can be used to represent real-world entities like buildings, places, and objects in the Internet, thereby becoming *virtual entities* that represent the respective real-world entities. Based on real-world attributes like Global Positioning System (GPS) coordinates or radio-frequency identification (RFID) tags, users can search for virtual entities via specialized search services, e.g., a geographic search service. Such a search service returns the IDs of virtual entities that match the query. Subsequently, these IDs can be resolved into the respective virtual entities via NetInf's name resolution services.

We have implemented an information-centric, location-aware AugNet application called *AugNet browser* running on an Android phone. The AugNet browser is a simple Android application that illustrates how to implement such applications easily by using NetInf. With the AugNet browser, a user can create virtual entities and can bind them to real-world entities via GPS coordinates, RFID, etc. Likewise, users can search for virtual entities based on real-world attributes and can present them, e.g., on a map (Figure 47) to easily access context-sensitive information.

11 The latter option obviously has some privacy concerns as the event service could constantly track the user's movements.

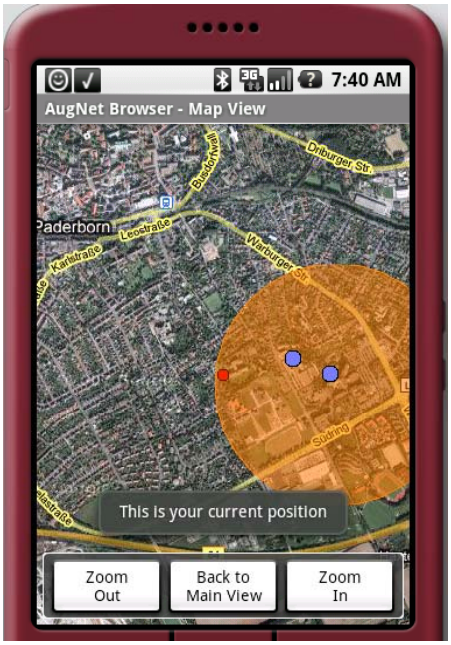

Figure 47: AugNet browser application showing the users current position (red dot) and two retrieved virtual entities (blue dots) in the user's search area (orange circle area)

We have found the AugNet browser implementation to be straightforward using the NetInf architecture. The main concepts like virtual entities are directly supported via NetInf's object model. The location-independent storage of virtual entities is helpful to store information locally close to their physical counterparts, thereby reducing latency, which is often critical for AugNet applications. The additional NetInf services like search and event notification can further simplify the development of AugNet applications as these services are in many cases important for AugNet applications to inform users about real-world events and to search for relevant information. At the same time, the NetInf architecture has proven to be compatible with application development for a mobile platform like Android. NetInf's ability to run the initial NetInf node remotely has been especially helpful. Thereby, the initial NetInf node does not have to run on the mobile device itself, which reduces the load on resource-constrained devices. However, running a dedicated NetInf node on a mobile device is feasible and provides the device with all NetInf benefits such as application-independent caching and infrastructure-less, ad hoc NetInf interaction with other NetInf devices.

6.4 EVALUATION

We have built a testbed running our OpenNetInf code to evaluate the NetInf architecture in a real setup. In the following, we evaluate the

NetInf advantages for a media distribution scenario in a setup similar to a small company network. More precisely, we evaluate the influence of the hierarchical MDHT NRS and our caching implementation (off-path and peer-side caching) on inter-domain traffic.

To limit the required hardware, we have used virtualization. Each node in the network (i.e., all MDHT nodes, caches, routers, clients, and the web server, see Figure 48) is a virtual machine, all running on a blade with 64 GB main memory and 16 CPU cores. This virtual setup simplified management while not influencing our measurement results as we focus on network traffic in bytes as main metric. For, e.g., network latency measurements, this kind of setup would not be suitable without additional adjustments that simulate the appropriate latencies.

As usual, quantitative results of testbed prototype evaluations depend heavily on the scenario setup. Nonetheless, they can give some intuition and can point out strengths and weaknesses of an architecture. In addition, this kind of evaluation proved very helpful to recognize conceptual problems and to verify that the prototype implements the architectural concepts correctly, e.g., concerning the cache behavior.

6.4.1 *Measurement Setup*

The general testbed setup is shown in Figure 48. The testbed represents a small company network with a hierarchical setup of 4 levels, representing 2 separate networks (e.g., in different buildings) interconnected via *router 1*. Each building has two separate internal networks, interconnected via *router 1.1* and *router 1.2* respectively. In our scenario, a small number of 16 employees collaborate on a joint project. The 16 employees are divided into 4 groups with 4 employees in each subnetwork. As we are not interested in the effectiveness of cache replacement algorithms and the influence of cache size, we eliminate these factors by focusing on a single file in our measurement setup. A varying number of employees has to access the same (initially external) 1 MB file one after the other. The request order is random.

In this setup, we focus on network traffic reduction. We have measured traffic in terms of inter-domain traffic, i.e., between the different levels. We measure the traffic between level 1 and 2, representing traffic to and from the outside ISP (at *router 1*). Due to our caching configuration as described below, there is no inter-building traffic at *router 1*. Traffic reduction at *router 1* is especially important for the company as the company has to pay for traffic to and from its ISP. We also measure traffic between level 2 and 3 (at *router 1.1* and *router 1.2*). This is intra-building traffic. We measure all traffic passing the respective router interfaces including any management traffic

and other overhead. The *original file server* serving the requested file resides outside the company network, i.e., it is connected at the top level. At levels 1–3, there is one in-network cache per network, resulting in 7 in-network caches in total. The overall setup consists of 11 distinct networks connected by 7 routers. Figure 48 also shows the setup of the hierarchical MDHT NRS system with three levels and a total of 12 MDHT nodes (the MDHT nodes at level 1 and 2 are virtual nodes that physically reside at level 3).

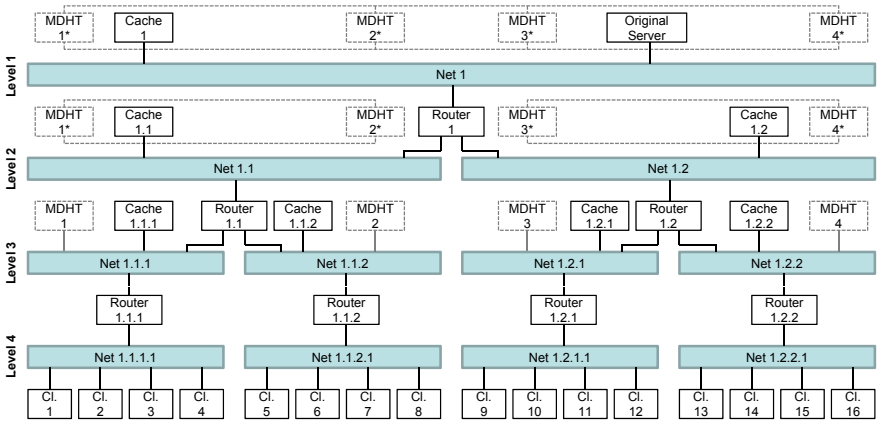

Figure 48: Measurement scenario and hierarchical MDHT setup (Cl = client)

We have used the following measurement settings: Peer-side caches serve other NetInf peers only in the same local network at level 4 (e.g., in the local wireless LAN in our example scenario). In-network caches serve all nodes in the respective subtree (e.g., *cache 1.1* serves *clients 1–8*) and cache an object as soon as it is requested the first time in its subtree. Sources are used in the following order: Peer-side caches, in-network caches at lower levels, in-network caches at higher levels. Multi-source download is turned off and there are no file updates.

6.4.2 *Results*

Figure 49 shows the cumulative inter-domain traffic between levels 1/2 and levels 2/3. We have measured the inter-domain traffic of the OpenNetInf system with all caching turned off, peer-side *or* in-network caching turned on, and both peer-side *and* in-network caching turned on. For comparison, we have measured the same scenario with "pure" HTTP, i.e., without any NetInf nodes and NetInf infrastructure. At first glance, P2P or CDN technology would be a better comparison. However, NetInf does not try to be *better* than P2P networks or CDNs but rather, *among others*, to integrate similar functionality into a general, consistent network architecture.

All measurements have been performed with a varying number of up to 16 downloading clients. The subset of clients is chosen randomly from all clients. The first client always downloads from the original server and all caches are initially empty. In the NetInf case, subsequent clients can access available (in-network and peer-side) caches. All figures show confidence intervals at a 95% confidence level.

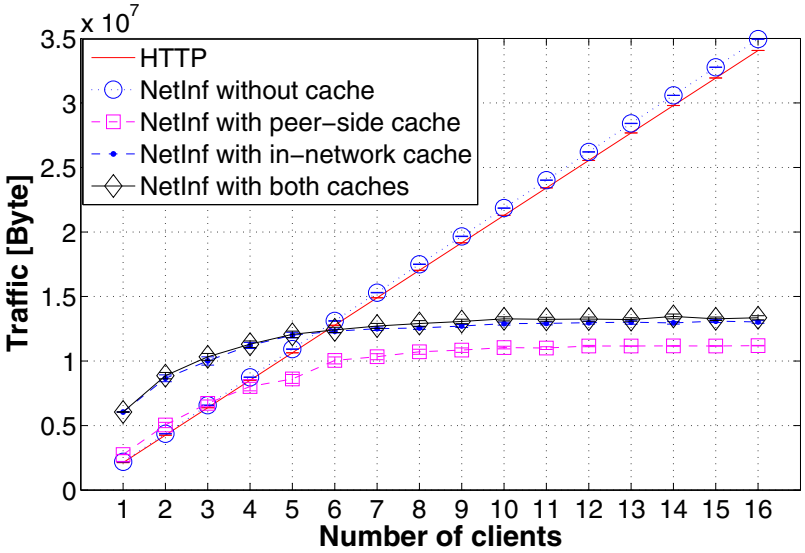

Figure 49: Cumulative inter-domain traffic between levels 1 & 2 and levels 2 & 3

The HTTP graph in Figure 49 shows traffic increasing linearly with the number of downloading clients, with about 34 MB cumulative traffic at both inter-domain borders for 16 downloading clients. This is as expected as all clients have to download the 1 MB file from the original server, which is connected to the top level, hence, producing $2 \cdot 16$ MB plus about 5% overhead from packet headers[12]. NetInf without caching shows a similar behavior. Comparing both cases shows that the (*unoptimized*) OpenNetInf implementation produces traffic overhead of about 2.5% compared to HTTP for a 1 MB file size. This overhead mainly consists of management packets generated by the MDHT system and NetInf request/response messages that contain also metadata.

Figure 49 demonstrates the positive influence of caching in our scenario. Both in-network caching and peer-side caching (combined as well as separately) result in a significantly reduced inter-domain traffic, keeping traffic as local as possible as a result of caching and the MDHT resolution approach. The traffic increases with the number of clients until at least one peer-side cache per subnetwork has cached the object or until the in-network caches contain the object, respec-

12 This number is consistent with theoretical calculations of packet overhead.

tively. Thereafter, the inter-domain traffic remains stable as all following clients can retrieve the object from the caches without producing additional inter-domain traffic. This also enables data access even if the network link to the company-external network at *router 1* is interrupted, i.e., supporting connectivity in fragmented networks.

There are conceptually two different ways how to fill caches, both with specific advantages and disadvantages. First, caches can be filled from other already existing local in-network caches and peer-side caches. This has the advantage that no extra inter-domain traffic is generated. However, the load for the serving local caches is increased. This might be especially critical for peer-side caches that are connected via an asymmetric connection like Asymmetric Digital Subscriber Line (ADSL), hence, only having a limited upstream data rate. Second, caches can be filled from (one of) the original source(s), thereby not increasing load at local caches but increasing inter-domain traffic.

In our measurements shown in Figure 49, we have used the second option. Hence, the NetInf case with only peer-side caching turned on produces the best results as filling the in-network caches produces additional inter-domain traffic in our sample setup. This also results in higher traffic compared to HTTP when the number of clients is small. In contrast, the peer caches are filled automatically whenever a peer requests the object anyways. However, whether peer-side caching or in-network caching yields better results heavily depends on the scenario, e.g., on the number of clients per subnetwork, on the number of subnetworks sharing a common in-network cache, local caching decisions by peers, and the ability of peers to cache at all. It also depends on the observed inter-domain level as shown in the following figures.

Figure 50 and Figure 51 show the inter-domain traffic for both inter-domain borders separately. As expected, the HTTP traffic is identical for both borders as all traffic passes through both borders when downloaded from the original server. Using peer-side caching results in significantly reduced inter-domain traffic at both borders. Each client subnetwork has to download the object once from the original source and can subsequently share the object within the local network at level 4. Using in-network caching results in significant savings as well. Interestingly, much more inter-domain traffic is saved at the higher border. This is due to the fact that most in-network caches are at the lower levels (level 2/3). Hence, client requests can be served from these caches without generating inter-domain traffic at the higher levels. A similar cache placement strategy can be expected in a real network setup as caches are typically placed at lower levels, i.e., closer to the network edge.

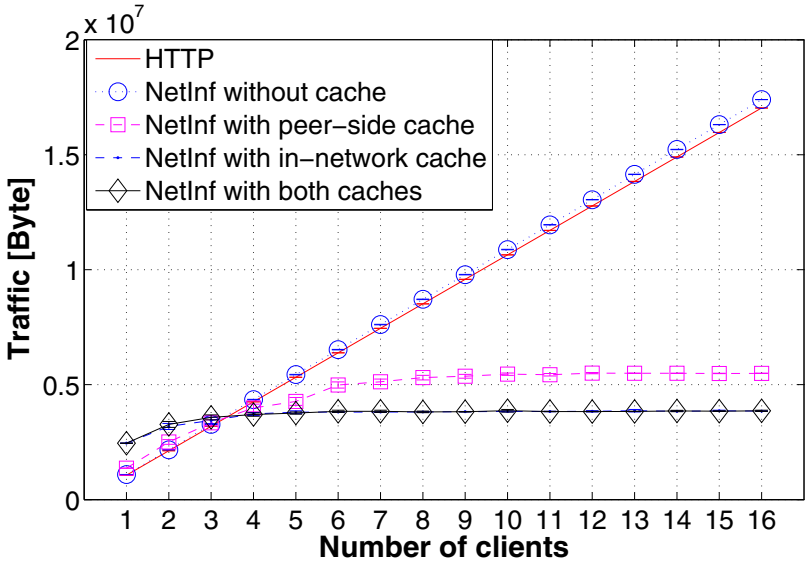

Figure 50: Inter-domain traffic between levels 1 & 2

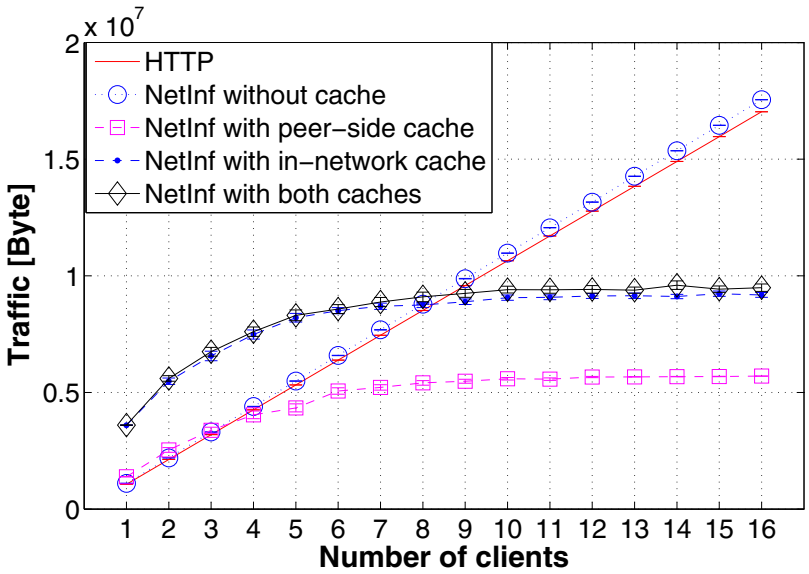

Figure 51: Inter-domain traffic between levels 2 & 3

Figure 50 also illustrates the potential load reduction at the original server. The inter-domain traffic is reduced by a factor of up to 4 with caching turned on, which translates into a similar load reduction at the original server. In addition, the load at the original server is reduced even more as the remaining traffic is shared between the top-level caches and the original server.

6.5 RELATED WORK

There are several other prototypes from other ICN projects and within the NetInf project.

The main prototype of the CCN/NDN project is called *CCNx*[13]. CCNx contains open protocol specifications and a reference implementation. Among others, an interactive voice over CCN application has been developed using CCNx [149]. The CCNx implementation is available as open source.

For the PSIRP/PURSUIT architecture, there are two prototypes available called *Blackhawk*[14] and *Blackadder*[15]. Blackhawk is a publish/subscribe prototype for FreeBSD. Blackadder is PURSUIT's new publish/subscribe prototype for FreeBSD and Linux. Both provide several features based on the information-centric communication paradigm, including packet forwarding, which is based on PSIRP's zFilters approach [86]. Blackhawk and Blackadder are both available as open source.

A prototype implementation of the DONA architecture has also been developed. The prototype is a stand-alone user-level Java daemon. It uses a TUN/TAP device to place itself between the transport and IP layer. Experiments have been performed on Planetlab as detailed in reference [12]. To our knowledge, the DONA prototype is not publicly available.

Besides OpenNetInf, there is also an open source NetInf software tool set available[16]. This open source project contains packages implementing various NetInf features in different languages, including the ni URI scheme, the convergence layer concept, forwarding, and caching. Languages include C, Clojure, Java, PHP, Python, and Ruby. As part of the SAIL project, there are also some additional (yet unpublished) NetInf prototypes, including an implementation of the GIN approach [87] and a router platform called NEC NetInf Router Platform (NNRP) [150].

The OpenNetInf prototype has also been used as basis for several information-centric research activities, e.g., for the development of an information-centric code repository [151], a rendezvous server serv-

13 http://www.ccnx.org
14 http://users.piuha.net/blackhawk/0.3/
15 https://github.com/fp7-pursuit/blackadder/
16 https://sourceforge.net/projects/netinf/

ing as name resolution and caching platform [150], and a mobile Android NetInf node [150].

6.6 SUMMARY

The OpenNetInf prototype has been very helpful in supporting the NetInf architecture design and in achieving the five prototyping goals stated in Section 6.1:

1. *Evaluate and validate the NetInf architecture:* We have evaluated and validated the overall NetInf architecture in multiple different scenarios and with several different applications. As a result, our prototyping experience indicates that the overall NetInf architecture is consistent and suitable for achieving significant advantages over today's host-centric architecture as described subsequently.

2. *Evaluate NetInf advantages:* The OpenNetInf prototype demonstrates many of these advantages, including efficient data dissemination with reduced network and inter-domain traffic, increased data availability (even in fragmented networks), and improved information-centric security functionality, which proved to be feasible and does not produce much overhead. All these advantages are also available for legacy applications.

3. *Migration:* The OpenNetInf prototype runs on top of IP. It can support different data transfer protocols like HTTP and FTP (and others like BitTorrent can easily be integrated) as a result of the flexible data transfer concept. Both aspects are crucial for migration and simplify future extensibility. The name-resolution-based two step approach offers many of the NetInf advantages while requiring no changes in today's underlying network and transport layers, especially not in the routers. Simply adding NetInf functionality on the user-terminal side already provides benefits, e.g., in local and disconnected network scenarios. Adding a name resolution infrastructure and in-network caches significantly extends these advantages. Both can be added gradually. Most importantly, the current Internet infrastructure can coexist and can be used in an information-centric way as well as in today's host-centric way.

4. *Application development:* Due to the RESTful HTTP API, migration is also simplified at the application level. The API can be used by legacy applications out of the box and development of new applications is fostered as the HTTP-based API is well understood by today's application developers. The API also proved to be generic enough for a wide variety of very different applications, ranging from video streaming to context-aware

real-world browser applications. The NetInf concepts are also appropriate for smartphones, which have different application development models and lower performance.

5. *Feasibility to extend the core NetInf architecture:* The additional services (search and event notification) have demonstrated to be helpful in several use cases. They have further simplified and sped up application development. At the same time, we have found no explicit need to closer integrate these additional services with the NetInf infrastructure. The integration via the API and N2N interface seems to be sufficient and is flexible with respect to future advancements.

The OpenNetInf code is available at `http://www.netinf.org`, including a detailed documentation [146].

7

CONCLUSION AND FUTURE WORK

This chapter is based on work published in references [2, 3, 4].

This chapter concludes this thesis with a discussion of deployment considerations, a summary and implications discussion, and future work items.

7.1 DEPLOYMENT

Independent of the specific architectural approach, it is important to consider deployment and the migration path from today's Internet to an ICN architecture for the ICN idea in general to be successful. We consider five types of actors that influence deployment:

- *end users* (private persons or organizations)
- *access network operators* that provide network access service to end users
- *connectivity network operators* that provide connectivity for other operators
- *content or service providers*
- *application developers*

These definitions are deliberately stereotyped and merely serve to make the following discussion a bit clearer. The main aspects discussed here remain intact when players take on a mixed role as is common in reality.

Both end users and application developers have to adopt the ICN approach to make it successful. Both actors would experience significant advantages in a successful information-centric network, like increased content availability, improved network performance, and a simplified API that better matches many of today's information-centric applications. At the same time, no significant investment is required by either actor. On the end user side, an ICN architecture might demand some storage and bandwidth resources. However, these are readily available and users are already accustomed to sharing these resources, e.g., for today's P2P applications.

To benefit from the full potential of an ICN architecture, the network operators' involvement is important. Network operators need

an incentive to start deploying the network functionality of an information-centric network. This functionality boils down to two/three main network components: *in-network caching* of data objects, *name resolution*, and potentially *forwarding/routing*.

We believe that the access network operators are the critical actors here. Access network operators get their revenue from their end users. They have to pay for their own network infrastructure to handle intra-domain traffic and connectivity network operators for inbound and outbound traffic that their users generate. The fee depends on the capacity of the link and/or the volume of traffic in both directions. Consequently, access operators have significant incentives to deploy ICN technology because its caching functionality will reduce the intra-domain and inter-domain traffic volume and, hence, their cost. However, there is a new cost incurred by this deployment, since in-network storage for performing ICN caching is typically not part of operators' network equipment today. The question is whether this investment cost is low enough to motivate deployment.

The relationship and the resulting incentive structure between access network operators and connectivity network operators as well as among connectivity operators is more complex. Nowadays, there is an ongoing struggle between network providers who is paying whom for inbound and outbound traffic. The payment flow heavily depends on the power balance between the actors. In any case, the party that has to pay for the incoming traffic has an incentive to deploy ICN caching to reduce their traffic costs. However, cache deployment will consequently change the power balance between the players and could lead to new power struggles and legal disputes such as legality of serving copyrighted content from caches. To prevent such struggles from hindering a widespread ICN adoption, it is important that new business models are negotiated among the different actors that reflect the new situation.

Likewise, there is currently a struggle between access network providers and content/service providers where the former group wants a share of the latter's revenue in order to finance the investment in network equipment. ICN might be an opportunity to introduce new business models resolving this tussle. Today, large content providers have to pay CDN operators for serving their content. The content-provider's in-house equipment simply does not suffice for popular sites and content. The content providers are, thus, indirectly paying for the investment in CDN caches within the access operators' networks. In an information-centric network, separate CDN caches would become superfluous because caching is performed directly by the access operators, which would facilitate new business models between access operators and content/service providers. At the same time, it is clearly infeasible for a content/service provider to have

agreements with all access operators for using their caches, which might provide new opportunities for current CDN operators.

7.2 SUMMARY AND IMPLICATIONS

ICN is a promising paradigm that offers significant advantages for a wide variety of scenarios. It has the potential to resolve many problems with today's large-scale information distribution, which is currently dominated by rapidly increasing video traffic. In addition, ICN is beneficial to all applications and network interactions that can be modeled after the paradigm of providing access to NDOs as a first-class networking service. This includes, e.g., ad hoc communication in challenged environments with intermittent connectivity.

ICN puts accessing named data objects, name-based routing and name resolution, in-network storage, and data object security into the thin waist of the network' hour glass – removing the need for application-specific overlays in many scenarios. This enables more efficient network communication and enables operators to manage data transport much better, e.g., resulting in significantly reduced inter-domain traffic.

As described in this thesis, the NetInf architecture combines some distinctive functionality into a unique information-centric network architecture. This mainly includes the naming and security model with a unique set of features that does not rely on a PKI for the basic security features, the flexible object model, the hybrid object retrieval method based on name resolution and name-based routing, flexibility to scale to very different network environments, and support for on-path as well as off-path caching, including peer nodes.

In this thesis, I have described the overall NetInf architecture and selected components with a focus on the secure naming scheme and the hierarchical name resolution framework (including a DNS traffic analysis). Secure naming is a key enabler for the overall NetInf architecture as it allows to access any available copy without requiring trust in the data source. The name resolution approach is especially important in the migration phase (and beyond) as it allows to run NetInf as an overlay without requiring changes to the underlying network protocols, thereby simplifying deployment.

I have evaluated the NetInf architecture and its components in several ways as shown in Figure 52. Especially, I have evaluated the feasibility of critical components, including the naming scheme, caching, and several NRS systems via prototyping, and have done multiple different performance evaluations of the NRS component. In addition, I have evaluated the feasibility of the overall NetInf architecture via prototyping of the overall architecture in combination with several ICN applications and have evaluated the performance of the overall architecture in media distribution and video streaming scenarios.

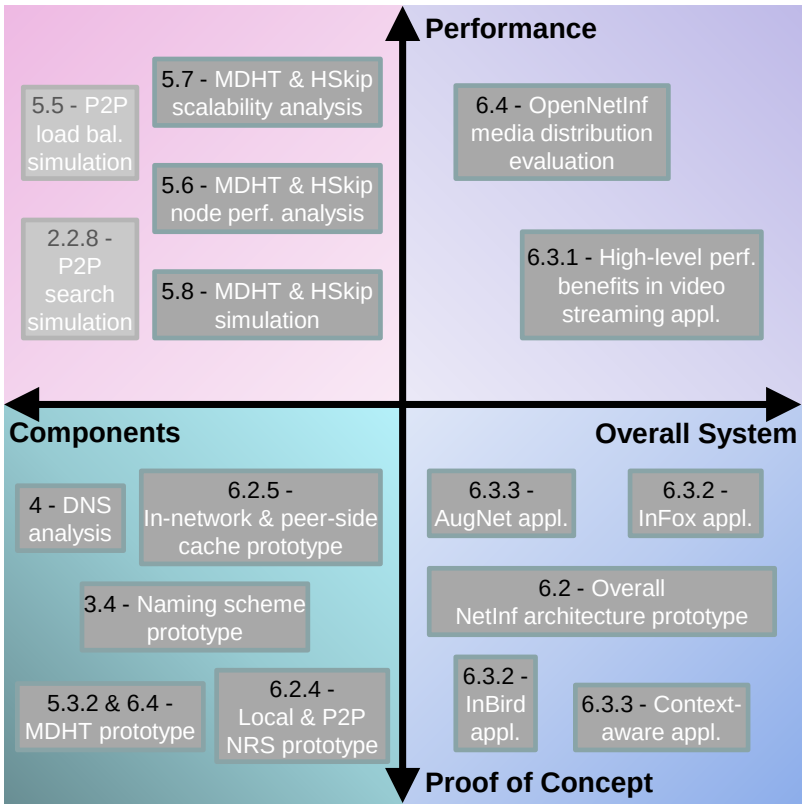

Figure 52: Evaluation overview. The numbers indicate the sections of this thesis where details can be found. Light boxes (number 2.2.8 and 5.5) indicate evaluations that are only summarized in this thesis.

For a global deployment, scalability of the NRS is especially critical for NetInf. The analysis and simulation results indicate that the proposed NetInf NRS solutions, specifically MDHT and HSkip, provide a scalable solution that can handle more than 10^{15} NDOs while keeping resolution latency (well) below 100 ms. These results are supported by the neighborhood effect as evaluated in this thesis. The combination of these results and my other feasibility and performance evaluations indicate that the Network of Information architecture is feasible on a large scale and should yield the promised advantages.

7.3 FUTURE WORK

Naturally, there are still interesting questions to look into and hard problems to address for the ICN ideas in general and the NetInf architecture specifically to become deployed and used on a wider scale:

NDO AGGREGATION The number of named data objects is vastly larger than the number of hosts in the current Internet, which means

that any ICN routing and name resolution system has a harder job compared to today's global IP routing and DNS name resolution. Although our NRS analysis and simulation results are promising, additional improvements of the NRS scalability might be required. These improvements can be achieved, e.g., via NDO aggregation (e.g., based on explicit aggregation [7]) that reduces the number of NDO entries to be stored in the NRS, improves cachability of NDO entries in the NRS, and reduces the number of resolution requests and consequently the load at NRS nodes. Besides name resolution, similar aggregation possibilities should also be investigated for NetInf's name-based routing approach.

ROUTING NetInf can support multiple routing protocols in parallel, e.g., routing protocols for edge domains and a routing protocol for the default-free zone. NetInf defines the guidelines and general conditions for these routing protocols, including protocol messages at the inter-domain level for registering new routing information and guidelines where to store routing state (i.e., in routers and/or in label stacks). However, more experiments are needed and more concrete (name-based) routing protocols have to be implemented in the future, with a focus on performance and scalability. Moreover, the integration of name-based routing and name resolution is interesting to investigate in more depth.

TRANSPORT PERFORMANCE NetInf's convergence layer allows for inter-connecting different types of networks into a single ICN. With underlays providing really different communication services (from uni-directional, opportunistic message forwarding to flow- and congestion-controlled higher-layer communication services; from delay-challenged to high-speed optical backbone networks), the interaction of NetInf transport functions with CL transport functions is interesting to investigate.

INTERACTIVE REAL-TIME COMMUNICATION The concepts of accessing named data objects and in-network caching in NetInf and ICN in general almost intuitively provide benefits for applications that are based on delivering requested named data objects. It is an interesting research question how far ICN can be taken to support different types of applications, especially those where the notion of *one request – one data object* does not hold or is not practical. Some of those applications such as voice over IP (VoIP) today employ a session abstraction and explicit session initiation transactions. Limits and extension possibilities for NetInf have to be investigated in this regard.

DYNAMIC DATA Currently, the focus of NetInf lies on static data such as large video files as the largest immediate benefits can be expected here. However, it is interesting to investigate how to apply the NetInf concepts to dynamic data such as frequently changing web pages. The NetInf secure naming concept does already work for dynamic data. However, consistent concepts for other aspects such as versioning, potential synchronization of distributed copies, etc. are needed to fully allow dynamic data to benefit from the NetInf advantages.

ACCESS CONTROL A global network where anybody has access to any published data object is not likely to prevail. Hence, the possibilities to apply access control concepts to NetInf and limit the publication scope of objects have to be investigated in more detail, e.g., leveraging NRS extensions. Likewise, encryption of NDO information can help to address this problem.

PRIVACY Requests for content are visible to the ICN network, resulting in a possibly worse privacy situation compared to today. On the other hand, it might not be possible to relate a request to a particular person. The privacy issues need to be investigated in more detail in order to understand the full consequences and to find means to mitigate them.

SECURITY An in-depth security analysis of the overall NetInf architecture and its components, which is not the focus of this thesis, is required in order to evaluate the relevant security goals and potential threads for NetInf. The security considerations discussed in this thesis, especially related to object naming, can serve as a starting point for a more thorough security analysis.

LEGAL ISSUES Ubiquitous caching probably does not sound too appealing for some content owners, who fear that their content can be illegally spread. The ongoing public copyright discussion can be helpful to find suitable solutions. Such solutions can potentially be based on a combination of new technical mechanisms and evolving laws and regulation, with the goal to provide a solution that is acceptable for all involved parties.

DEPLOYMENT The incentives for all involved players have to be clearly communicated to foster deployment. It can be supported by standardizing central ICN aspects (as we have already started, e.g., for our naming concept [28]). NetInf's ability to grow from the edges, i.e., supporting incremental deployment, should also simplify deployment.

BIBLIOGRAPHY

[1] B. Ager, W. Mühlbauer, G. Smaragdakis, and S. Uhlig, "Web content cartography," in *Internet Measurement Conference (IMC '11)*. ACM, November 2011. (Cited on pages xvi, 88, and 89.)

[2] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012. (Cited on pages 1, 17, 22, and 157.)

[3] C. Dannewitz, D. Kutscher, B. Ohlman, S. Farrell, B. Ahlgren, and H. Karl, "Network of information (NetInf) – An information-centric networking architecture," *Computer Communications*, vol. 36, no. 7, pp. 721–735, April 2013. (Cited on pages 1, 11, 17, 22, 23, and 157.)

[4] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking (draft)," in *Information-Centric Networking*, ser. Dagstuhl Seminar Proceedings, B. Ahlgren, H. Karl, D. Kutscher, B. Ohlman, S. Oueslati, and I. Solis, Eds., no. 10492. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Germany, 2011. (Cited on pages 1, 17, 22, and 157.)

[5] Cisco Systems, Inc., "Entering the zettabyte era," white paper, June 2011. (Cited on pages 1 and 140.)

[6] ——, "Cisco visual networking index: Global mobile data traffic forecast update," white paper, February 2012. (Cited on page 2.)

[7] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures," in *Proc. ACM SIGCOMM Workshop on Information-Centric Networking*. New York, NY, USA: ACM, 2011, pp. 1–6. (Cited on pages 6, 11, 24, 25, 46, 49, and 161.)

[8] W. Chai, N. Wang, I. Psaras, G. Pavlou, C. Wang, G. De Blas, F. Salguero, L. Liang, S. Spirou, A. Beben, and et al., "CURLING: Content-ubiquitous resolution and delivery infrastructure for next-generation services," *IEEE Communications Magazine*, vol. 49, no. 3, pp. 112–120, 2011. (Cited on pages 7 and 44.)

[9] K. Fall, "A delay-tolerant network architecture for challenged internets," in *SIGCOMM '03: Proc. Applications, Technologies, Architectures, and Protocols for Computer Communications*. New York, NY, USA: ACM Press, 2003, pp. 27–34. (Cited on page 10.)

[10] C. Perkins, "IP Mobility Support," RFC 2002 (Proposed Standard), Internet Engineering Task Force, Oct. 1996, obsoleted by RFC 3220, updated by RFC 2290. (Cited on pages 11 and 28.)

[11] D. R. Cheriton and M. Gritter, "TRIAD: A new next-generation Internet architecture," project web page: http://www-dsg.stanford.edu/triad/, July 2000. (Cited on pages 11 and 43.)

[12] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A data-oriented (and beyond) network architecture," in *Proc. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM '07)*. New York, NY, USA: ACM Press, 2007, pp. 181–192. (Cited on pages 11, 43, 66, and 154.)

[13] A. Ghodsi, T. Koponen, B. Raghavan, S. Shenker, A. Singla, and J. Wilcox, "Information-centric networking: Seeing the forest for the trees," in *Proc. 10th ACM Workshop on Hot Topics in Networks (HotNets-X)*. ACM Press, Nov. 2011. (Cited on pages 11 and 46.)

[14] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. L. Braynard, "Networking named content," in *Proc. 5th ACM Conf. Emerging Networking EXperiments and Technologies (ACM CoNEXT)*, Rome, Italy, December 2009. (Cited on pages 11 and 66.)

[15] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. D. Thornton, D. K. Smetters, B. Zhang, G. Tsudik, kc claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh, "Named data networking (NDN) project," PARC, Tech. Rep. NDN-0001, October 2010. (Cited on pages 11 and 43.)

[16] M. Ain, D. Trossen, P. Nikander, S. Tarkoma, K. Visala, K. Rimey, T. Burbridge, J. Rajahalme, J. Tuononen, P. Jokela, J. Kjällman, J. Ylitalo, J. Riihijärvi, B. Gajic, G. Xylomenos, P. Savolainen, and D. Lagutin, "D2.3 – Architecture definition, component descriptions, and requirements," Deliverable, PSIRP 7th FP EU-funded project, February 2009. (Cited on pages 11 and 44.)

[17] D. Trossen, G. Parisis, B. Gajic, J. Riihijärvi, P. Flegkas, P. Sarolahti, P. Jokela, X. Vasilakos, C. Tsilopoulos, S. Arianfar, and M. Reed, "D2.3 – Architecture definition, components descriptions and requirements," Deliverable, PURSUIT, 7th FP EU-funded project, October 2011. (Cited on page 11.)

[18] B. Ahlgren, M. D'Ambrosio, C. Dannewitz, A. Eriksson, J. Golic, B. Grönvall, D. Horne, A. Lindgren, O. Mämmelä, M. Marchisio, J. Mäkelä, S. Nechifor, B. Ohlman, K. Pentikousis, S. Randriamasy, T. Rautio, E. Renault, P. Seittenranta, O. Strandberg, P. Talaba, V. Vercellone, and D. Zeghlache, "Second netinf architecture description (D.6.2)," Deliverable, 4WARD 7th FP EU-funded project, January 2010. (Cited on page 11.)

[19] B. Ahlgren, M. D'Ambrosio, E. Davies, A. E. Eriksson, S. Farrell, B. Grönvall, C. Imbrenda, B. Kauffmann, G. Kunzmann, D. Kutscher, A. Lindgren, I. Marsh, L. Muscariello, B. Ohlman, K.-A. Persson, P. Pöyhönen, M. Shehada, D. Staehle, O. Strandberg, J. Tuononen, and V. Vercellone, "Content delivery and operations (D.B.2)," Deliverable, SAIL 7th FP EU-funded project, May 2012. (Cited on pages 11, 25, 35, 37, 40, and 46.)

[20] A. Eriksson and B. Ohlman, "Dynamic internetworking based on late locator construction," in *10th IEEE Global Internet Symposium*, May 2007. (Cited on pages 16 and 28.)

[21] C. Dannewitz, "NetInf: An information-centric design for the future Internet," in *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, Munich, Germany, May 2009. (Cited on pages 17 and 23.)

[22] B. Ahlgren, M. D'Ambrosio, C. Dannewitz, M. Marchisio, I. Marsh, B. Ohlman, K. Pentikousis, R. Rembarz, O. Strandberg, and V. Vercellone, "Design considerations for a network of information," in *Proc. Workshop on Re-Architecting the Internet (ReArch)*, Spain, December 2008. (Cited on pages 17 and 23.)

[23] C. Dannewitz, K. Pentikousis, R. Rembarz, E. Renault, O. Strandberg, and J. Ubillos, "Scenarios and research issues for a network of information," in *Proc. 4th Int. Mobile Multimedia Communications Conference*, Oulu, Finland, July 2008. (Cited on pages 17, 23, and 137.)

[24] C. Dannewitz, T. Biermann, M. Dräxler, and H. Karl, "Complex queries in P2P networks with resource-constrained devices," *Journal of Advances in Information Technology (JAIT) – Special Issue on Advances in P2P Technology*, vol. 02, no. 01, pp. 02–14, January 2011. (Cited on pages 17, 23, and 42.)

[25] T. Biermann, C. Dannewitz, and H. Karl, "An adaptive resource/performance trade-off for resolving complex queries in P2P networks," in *Proc. IEEE International Conference on Communications (ICC)*, Dresden, Germany, June 2009. (Cited on pages 17, 23, and 42.)

[26] ——, "Extended results on an adaptive resource/performance trade-off for resolving complex queries in P2P networks," University of Paderborn, Paderborn, Germany, Tech. Rep. TR-RI-08-294, October 2008. (Cited on pages 17, 23, and 42.)

[27] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure naming for a network of information," in *Proc. 13th IEEE Global Internet Symposium 2010 (in conjunction with IEEE INFOCOM)*, San Diego, USA, March 2010. (Cited on pages 18 and 55.)

[28] S. Farrell, D. Kutscher, C. Dannewitz, B. Ohlman, A. Keranen, and P. Hallam-Baker, "Naming Things with Hashes," RFC 6920 (Proposed Standard), Internet Engineering Task Force, Apr. 2013. (Cited on pages 18, 33, 41, 55, 65, 69, 141, 142, and 162.)

[29] H. Song, N. Zong, Y. Yang, and R. Alimi, "Decoupled application data enroute (DECADE) problem statement," Network Working Group Internet-Draft, October 2009. (Cited on pages 19 and 68.)

[30] B. Ohlman, O. Strandberg, C. Dannewitz, A. Lindgren, R. Maglione, and B. Ahlgren, "Requirements for accessing data in network storage," IETF Internet Draft draft-ohlman-decade-add-use-cases-reqs-02, October 2010. (Cited on pages 19, 55, and 68.)

[31] Y. Zhang, N. Zong, G. Camarillo, R. Yang, and V. Pascual, "Problem statement and requirements of peer-to-peer streaming," Internet Draft, February 2012. (Cited on pages 19 and 68.)

[32] C. Dannewitz, T. Rautio, O. Strandberg, and B. Ohlman, "Secure naming structure and p2p application interaction," IETF Internet-Draft draft-dannewitz-ppsp-secure-naming-02, March 2011. (Cited on pages 19, 55, and 68.)

[33] Z. Shelby, K. Hartke, C. Bormann, and B. Frank, "Constrained application protocol (CoAP)," IETF Internet draft draft-ietf-core-coap-09, March 2012. (Cited on pages 19 and 68.)

[34] C. Dannewitz, H. Karl, and A. Yadav, "Report on locality in DNS requests – Evaluation and impact on future Internet architectures," University of Paderborn, Paderborn, Germany, Tech. Rep. TR-RI-12-323, July 2012. (Cited on pages 19 and 71.)

[35] C. Dannewitz, M. D'Ambrosio, and V. Vercellone, "Hierarchical DHT-based name resolution for information-centric networks," *Computer Communications*, vol. 36, no. 7, pp. 736–749, April 2013. (Cited on pages 20 and 93.)

[36] M. D'Ambrosio, C. Dannewitz, H. Karl, and V. Vercellone, "MDHT: A hierarchical name resolution service for information-centric networks," in *Proc. ACM SIGCOMM Workshop on Information-centric Networking*. New York, NY, USA: ACM, August 2011, pp. 7–12. (Cited on pages 20 and 93.)

[37] D. Warneke and C. Dannewitz, "Load balancing in P2P networks: Using statistics to fight data and execution skew," *Journal of Advances in Information Technology (JAIT) – Special Issue on Advances in P2P Technology*, vol. 02, no. 01, pp. 40–49, 2011. (Cited on pages 21, 93, and 112.)

[38] ——, "Statistics-based ID management for load balancing in structured P2P networks," in *Proc. 34th IEEE Conference on Local Computer Networks (LCN)*. Zürich, Switzerland: IEEE, October 2009. (Cited on pages 21, 93, and 112.)

[39] C. Dannewitz, T. Biermann, M. Dräxler, F. Beister, and H. Karl, "Prototyping with the Future Internet Toolbox," in *Proc. 6th Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, May 2010. (Cited on pages 21 and 133.)

[40] T. Biermann, C. Dannewitz, and H. Karl, "FIT: Future Internet Toolbox," in *Proc. 6th Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, May 2010. (Cited on pages 21, 133, and 134.)

[41] ——, "FIT: Future Internet Toolbox — extended report," University of Paderborn, Paderborn, Germany, Tech. Rep. TR-RI-10-311, Feb. 2010. (Cited on pages 21 and 133.)

[42] C. Dannewitz, "Augmented Internet: An information-centric approach for real-world/Internet integration," in *Proc. Int. Workshop on the Network of the Future (in conjunction with IEEE ICC)*, June 2009. (Cited on pages 21, 133, and 146.)

[43] C. Dannewitz, H. Karl, and D. Warneke, "Service platform for real-world/Internet integration in mobile applications," in *Proc. 13. Mobilfunktagung*, Osnabrück, Germany, May 2008. (Cited on pages 21, 133, and 146.)

[44] C. Dannewitz, M. Herlich, and H. Karl, "OpenNetInf – Prototyping an information-centric network architecture," in *Proc. IEEE LCN – Workshop on Architectures, Services and Applications for the Next Generation Internet (WASA-NGI)*, October 2012. (Cited on pages 21 and 133.)

[45] C. Dannewitz and T. Biermann, "Prototyping a network of information," in *Demonstrations – IEEE Local Computer Networks (LCN)*, Zurich, Switzerland, 2009. (Cited on pages 21 and 133.)

[46] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden, "Tussle in cyberspace: Defining tomorrow's Internet," in *Proc. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. New York, NY, USA: ACM Press, 2002, pp. 347–356. (Cited on page 25.)

[47] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, K. Fall, and H. Weiss, "Delay-Tolerant Networking Architecture," RFC 4838 (Informational), Internet Engineering Task Force, Apr. 2007. (Cited on pages 25 and 46.)

[48] J. Seedorf and E. Burger, "Application-layer traffic optimization (ALTO) problem statement," RFC 5693, October 2009. (Cited on pages 38, 46, and 129.)

[49] H. Xie, A. Krishnamurthy, A. Silberschatz, and Y. R. Yang, "P4P: Explicit communications for cooperative control between P2P and network providers," P4PWG whitepaper, May 2007. (Cited on pages 38, 46, and 129.)

[50] A. Eriksson and B. Ohlman, "Scalable object-to-object communication over a dynamic global network," in *Proc. Future Network and MobileSummit 2010*, June 2010. (Cited on pages 39 and 52.)

[51] M. Cai, M. Frank, J. Chen, and P. Szekely, "MAAN: A multi-attribute addressable network for grid information services," in *Proc. Fourth International Workshop on Grid Computing (GRID '03)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 184. (Cited on page 42.)

[52] V. Jacobson, D. K. Smetters, J. D. Thornton, M. Plass, N. Briggs, and R. Braynard, "Networking named content," *Comm. ACM*, vol. 55, pp. 117–124, Jan. 2012. (Cited on page 43.)

[53] "PURSUIT – publish–subscribe Internet technology," http://www.fp7-pursuit.eu/. (Cited on page 44.)

[54] "Content-oriented networking: A new experience for content transfer (connect)," http://www.anr-connect.org/. (Cited on page 44.)

[55] S. Salsano, A. Detti, G. Tropea, N. B. Melazzi, L. Chiariglione, H. Castro, A.-C. Anadiotis, A. Mousas, C. Patrikakis, T. Huebner, M. Tanase, L. Corlan, P. Gkonis, J. Ribas, and D. Sequeira, "Convergence project – system architecture," EU project deliverable D3.2, July 2011. (Cited on page 44.)

[56] "Content Aware Searching retrieval and sTreaming (COAST)," http://www.coast-fp7.eu/. (Cited on page 44.)

[57] M. Xu, Z. Ming, D. Li, and C. Xia, "SIONA: A service and information oriented network architecture," Proc. AsiaFI Summer School, August 2011. (Cited on page 44.)

[58] "COntent Mediator architecture for content-aware nETworks (COMET)," http://www.comet-project.org. (Cited on page 44.)

[59] D. Han, A. Anand, F. Dogar, B. Li, H. Lim, M. Machado, A. Mukundan, W. Wu, A. Akella, D. G. Andersen, J. W. Byers, S. Seshan, and P. Steenkiste, "XIA: Efficient support for evolvable internetworking," in *Proc. 9th USENIX NSDI*, San Jose, CA, 2012. (Cited on page 44.)

[60] K. Katsaros, G. Xylomenos, and G. C. Polyzos, "MultiCache: An overlay architecture for information-centric networking," *Computer Networks*, vol. 55, no. 4, pp. 936–947, March 2010. (Cited on page 45.)

[61] A. Rowstron and P. Druschel, "Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems," in *Proc. IFIP/ACM Conference on Distributed Systems Platforms*, Nov. 2001, pp. 329–350. (Cited on pages 45 and 129.)

[62] M. Castro, P. Druschel, A. M. Kermarrec, and A. I. T. Rowstron, "Scribe: A large-scale and decentralized application-level multicast infrastructure," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 8, pp. 1489–1499, 2002. (Cited on page 45.)

[63] S. Paul, "Postcards from the edge: A cache and forward architecture for the future Internet," in *2nd COST-NSF Workshop on Future Internet (NeXtWorking)*, Berlin, April 2007. (Cited on page 45.)

[64] L. Dong, H. Liu, Y. Zhang, S. Paul, and D. Raychaudhuri, "On the cache-and-forward network architecture," in *Proc. IEEE Conference on Communications (ICC'09)*. Piscataway, NJ, USA: IEEE Press, 2009, pp. 2119–2123. (Cited on page 45.)

[65] A. Carzaniga and A. L. Wolf, "Content-based networking: A new communication infrastructure," in *NSF Workshop on an Infrastructure for Mobile and Wireless Systems*, ser. Lecture Notes in Computer Science, no. 2538. Scottsdale, Arizona: Springer-Verlag, October 2001, pp. 59–68. (Cited on page 45.)

[66] A. Carzaniga, M. Papalini, and A. L. Wolf, "Content-based publish/subscribe networking and information-centric networking," in *Proc. ACM SIGCOMM Workshop on Information-Centric Networking (ICN-2011)*, Toronto, Canada, Aug. 2011, pp. 56–61. (Cited on page 45.)

[67] A. Carzaniga, "Architectures for an event notification service scalable to wide-area networks," Ph.D. dissertation, Politecnico di Milano, Milano, Italy, Dec. 1998. (Cited on pages 46 and 140.)

[68] J. Greifenberg and D. Kutscher, "Efficient publish/subscribe-based multicast for opportunistic networking with self-organized resource utilization," in *Proc. First IEEE International Workshop on Opportunistic Networking*, 2008. (Cited on page 46.)

[69] J. Su, J. Scott, P. Hui, J. Crowcroft, E. De Lara, C. Diot, A. Goel, M. H. Lim, and E. Upton, "Haggle: Seamless networking for mobile applications," in *Proc. Conference on Ubiquitous Computing (UbiComp '07)*. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 391–408. (Cited on page 46.)

[70] D. Trossen and A. Kostopoulos, "Exploring the tussle space for information-centric networking," in *Proc.- 39th Research Conference on Communication, Information and Internet Policy (TPRC)*, Arlington, Sep. 2011. (Cited on page 46.)

[71] D. Trossen, M. Särelä, and K. R. Sollins, "Arguments for an information-centric internetworking architecture," *Computer Communication Review*, vol. 40, no. 2, pp. 26–33, 2010. (Cited on page 46.)

[72] Y. Wang, K. Lee, B. Venkataraman, R. L. Shamanna, I. Rhee, and S. Yang, "Advertising cached contents in the control plane: Necessity and feasibility," in *Proc. Workshop on Emerging Design Choices in Name-Oriented Networking (in conjunction with IEEE INFOCOM)*, Orlando, USA, March 2012. (Cited on page 46.)

[73] I. Psaras, R. G. Clegg, R. Landa, W. K. Chai, and G. Pavlou, "Modelling and evaluation of CCN-caching trees," in *Proc. 10th Conference on Networking – Volume Part I (NETWORKING'11)*. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 78–91. (Cited on page 46.)

[74] L. Muscariello, G. Carofiglio, and M. Gallo, "Bandwidth and storage sharing performance in information centric networking," in *Proc. ACM SIGCOMM Workshop on Information-Centric Networking (ICN '11)*. New York, NY, USA: ACM, 2011, pp. 26–31. (Cited on page 46.)

[75] G. Carofiglio, M. Gallo, L. Muscariello, and D. Perino, "Modeling data transfer in content-centric networking," in *Proc. Teletraffic Congress (ITC '11)*. ITCP, 2011, pp. 111–118. (Cited on page 46.)

[76] J. Rajahalme, M. Särelä, P. Nikander, and S. Tarkoma, "Incentive-compatible caching and peering in data-oriented net-

works," in *Proc. ACM CoNEXT Conference – ReArch Workshop*, December 2008. (Cited on page 46.)

[77] G. Carofiglio, M. Gallo, and L. Muscariello, "ICP: Design and evaluation of an interest control protocol for content-centric networking," in *Proc. IEEE INFOCOM Workshop on Emerging Design Choices in Name-Oriented Networking*, Orlando, USA, March 2012. (Cited on page 46.)

[78] S. Oueslati, J. Roberts, and N. Sbihi, "Flow-aware traffic control for a content-centric network," in *Proc. IEEE INFOCOM*, 2012. (Cited on page 46.)

[79] J. Rajahal, M. Särelä, K. Visala, and J. Riihijärvi, "On name-based inter-domain routing," *Computer Networks*, vol. 55, pp. 975–986, March 2011. (Cited on page 46.)

[80] S. DiBenedetto, C. Papadopoulos, and D. Massey, "Routing policies in named data networking," in *Proc. ACM SIGCOMM Workshop on Information-Centric Networking (ICN '11)*. New York, NY, USA: ACM, 2011, pp. 38–43. (Cited on page 46.)

[81] M. Baugher, B. Davie, A. Narayanan, and D. Oran, "Self-verifying names for read-only named data," in *Proc. Workshop on Emerging Design Choices in Name-Oriented Networking*, 2012. (Cited on page 46.)

[82] Privacy and Security Working Group, "Identity in an information-centric Internet," White paper, CFP Privacy and Security Working Group, MIT, April 2008. (Cited on page 46.)

[83] H. S. Jeon, I. S. Choi, B. J. Lee, and H. Y. Song, "A closer look at content-centric internet research projects," in *Proc. Advanced Communication Technology (ICACT)*, 2012, pp. 698–702. (Cited on page 46.)

[84] A. Baid, T. Vu, and D. Raychaudhuri, "Comparing alternative approaches for networking of named objects in the future Internet," in *Proc. Computer Communications – NOMEN workshop*, March 2012. (Cited on page 46.)

[85] J. Choi, J. Han, E. Cho, T. Kwon, and Y. Choi, "A survey on content-oriented networking for efficient content delivery," *IEEE Communications Magazine*, vol. 49, pp. 121–127, March 2011. (Cited on page 46.)

[86] P. Jokela, A. Zahemszky, C. Esteve Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-Networking," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 195–206, 2009. (Cited on pages 48 and 154.)

[87] M. D'Ambrosio, P. Fasano, M. Marchisio, V. Vercellone, and M. Ullio, "Providing data dissemination services in the future Internet," in *Proc. World Telecommunications Congress (WTC'08) (in conjunction with IEEE Globecom)*, New Orleans, LA, USA, December 2008. (Cited on pages 53 and 154.)

[88] N. Paskin, "Digital object identifier (DOI®) system," in *Encyclopedia of Library and Information Sciences*, 3rd ed. Taylor & Francis, 2010. (Cited on pages 56 and 67.)

[89] J. Katz and Y. Lindell, *Introduction to Modern Cryptography: Principles and Protocols*, ser. CRC Cryptography and Network Security Series, 1, Ed. Chapman & Hall, 2007. (Cited on page 58.)

[90] P. Hallam-Baker, R. Stradling, S. Farrell, D. Kutscher, and B. Ohlman, "The named information (ni) URI scheme: Parameters," IETF Internet-Draft draft-hallambaker-decade-ni-params-02, April 2012. (Cited on pages 65 and 69.)

[91] M. Nottingham and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)," RFC 5785 (Proposed Standard), Internet Engineering Task Force, Apr. 2010. (Cited on pages 66 and 142.)

[92] R. L. Rivest and B. Lampson, "SDSI – A simple distributed security infrastructure," MIT, Tech. Rep., 1996. (Cited on page 66.)

[93] D. Mazières and M. F. Kaashoek, "Escaping the evils of centralized control with self-certifying pathnames," in *Proc. SIGOPS European Workshop*, 1998. (Cited on page 67.)

[94] C. Ellison, B. Frantz, B. Lampson, R. Rivest, B. Thomas, and T. Ylonen, "SPKI Certificate Theory," RFC 2693, IETF, Sept. 1999. (Cited on page 67.)

[95] M. Blaze, J. Feigenbaum, and A. D. Keromytis, "Keynote: Trust management for public-key infrastructures," in *Proc. Security Protocols International Workshop*, vol. 1550. Cambridge, England: Springer LNCS, 1998, pp. 59–63. (Cited on page 67.)

[96] S. Sun, L. Lannom, and B. Boesch, "IETF RFC 3650, handle system overview," November 2003. (Cited on page 67.)

[97] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," RFC 4423 (Informational), Internet Engineering Task Force, May 2006. (Cited on page 67.)

[98] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana, "Internet indirection infrastructure," in *Proc. ACM SIGCOMM*, US, August 2002. (Cited on page 67.)

[99] B. Ahlgren, J. Arkko, L. Eggert, and J. Rajahalme, "A node identity internetworking architecture," in *Proc. 9th IEEE Global Internet Symposium (in conjunction with IEEE INFOCOM)*, Spain, April 2006. (Cited on page 67.)

[100] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish, "A layered naming architecture for the Internet," in *Proc. SIGCOMM*, Oregon, USA, 2004. (Cited on page 67.)

[101] B. Zdrnja, N. Brownlee, and D. Wessels, "Passive monitoring of DNS anomalies," in *Proc. Detection of Intrusions, Malware, and Vulnerability Assessment (DIMVA)*, Lucerne, Switzerland, July 2007, pp. 129–139. (Cited on page 78.)

[102] T. McGregor, S. Alcock, and D. Karrenberg, "The RIPE NCC internet measurement data repository," in *Proc. 11th International Conference on Passive and Active Measurement*, ser. PAM'10. Berlin, Heidelberg: Springer-Verlag, 2010, pp. 111–120. (Cited on page 87.)

[103] S. Castro, M. Zhang, W. John, D. Wessels, and k. claffy, "Understanding and preparing for DNS evolution ," in *Traffic Monitoring and Analysis Workshop (TMA)*, Zurich, Switzerland, April 2010, pp. 1–6. (Cited on page 87.)

[104] S. Castro, D. Wessels, M. Fomenkov, and k. claffy, "A Day at the Root of the Internet," *ACM SIGCOMM Computer Communication Review (CCR)*, no. 5, pp. 41–46, Oct 2008. (Cited on page 87.)

[105] J. Jung, E. Sit, H. Balakrishnan, and R. Morris, "DNS performance and the effectiveness of caching," in *Proc. 1st ACM SIGCOMM Workshop on Internet Measurement (IMW)*. New York, NY, USA: ACM Press, 2001, pp. 153–167. (Cited on page 87.)

[106] B. Ager, F. Schneider, J. Kim, and A. Feldmann, "Revisiting cacheability in times of user generated content," in *Proc. INFOCOM IEEE Conference on Computer Communications Workshops*. New York, NY, USA: IEEE, March 2010, pp. 1–6. (Cited on page 87.)

[107] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, "Measurement, modeling, and analysis of a peer-to-peer file-sharing workload," *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5, pp. 314–329, December 2003. (Cited on page 87.)

[108] P. McDaniel and S. Jamin, "A scalable key distribution hierarchy," University of Michigan. Department of Electrical Engineering and Computer Science, Tech. Rep., 1998. (Cited on pages 87 and 89.)

[109] N. J. A. Harvey, M. B. Jones, S. Saroiu, M. Theimer, and A. Wolman, "SkipNet: A scalable overlay network with practical locality properties," in *Proc.USENIX Symposium on Internet Technologies and Systems (USITS)*. Berkeley, CA, USA: USENIX Association, 2003, p. 9. (Cited on pages 94, 104, 107, and 130.)

[110] F. Consulting, "ecommerce web site performance today – an updated look at consumer reaction to a poor online shopping experience," online, September 2009. (Cited on page 94.)

[111] J. Gantz and D. Reinsel, "Extracting value from chaos," White paper, June 2011. (Cited on page 95.)

[112] I. Stoica, R. Morris, D. Liben-Nowell, D. Karger, M. Kaashoek, F. Dabek, and H. Balakrishnan, "Chord: A scalable peer-to-peer lookup protocol for Internet applications," *IEEE/ACM Transactions on Networking*, vol. 11, no. 1, pp. 17–32, February 2003. (Cited on pages 102, 118, and 129.)

[113] G. S. Manku, "Balanced binary trees for ID management and load balance in distributed hash tables," in *Proc. Twenty-Third Annual ACM Symposium on Principles of Distributed Computing (PODC '04)*. New York, NY, USA: ACM, 2004, pp. 197–205. (Cited on page 111.)

[114] P. Ganesan, B. Yang, and H. Garcia-Molina, "One torus to rule them all: Multi-dimensional queries in P2P systems," in *Proc. 7th International Workshop on the Web and Databases (WebDB '04)*. New York, NY, USA: ACM, 2004, pp. 19–24. (Cited on pages 111 and 112.)

[115] L. Breslau, P. Cao, L. Fan, G. Phillips, and S. Shenker, "Web caching and Zipf-like distributions: Evidence and implications," in *Proc. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM '99)*, March 1999, pp. 126–134. (Cited on page 112.)

[116] S. Zhao, D. Stutzbach, and R. Rejaie, "Characterizing files in the modern Gnutella network: A measurement study," in *Proc. SPIE/ACM Multimedia Computing and Networking*, San Jose, CA, January 2006. (Cited on page 112.)

[117] Texas Memory Systems, Web source: http://www.ramsan.com, last checked: June 2012. (Cited on page 113.)

[118] The Measurement Factory, "DNS survey," Web source: http://dns.measurement-factory.com/surveys/200810.html, October 2008, last checked: June 2012. (Cited on page 113.)

[119] H. Iinou, M. Zushi, H. Nishida, and K. Sato, "DNS query traffic increase on caching DNS resolvers," DNS-OARC Workshop, Denver, October 2010. (Cited on page 113.)

[120] M. Steiner and E. W. Biersack, "Where is my peer? Evaluation of the Vivaldi network coordinate system in Azureus," in *Proc. 8th IFIP-TC 6 Networking Conference*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 145–156. (Cited on page 117.)

[121] S. Dröge, "Design and evaluation of a scalable information dissemination layer for information-centric networks," Master's thesis, University of Paderborn, April 2011, supervised by C. Dannewitz, H. Karl. (Cited on page 122.)

[122] J. Cho and H. Garcia-Molina, "Estimating frequency of change," *ACM Transactions on Internet Technology (TOIT)*, vol. 3, no. 3, pp. 256–290, August 2003. (Cited on page 123.)

[123] A. Dandoush, S. Alouf, and P. Nain, "Lifetime and availability of data stored on a P2P system: Evaluation of recovery schemes," INRIA, Research Report RR-7170, January 2010. (Cited on page 123.)

[124] L. O. Walters, "A Web browsing workload model for simulation," Ph.D. dissertation, University of Cape Town, May 2004. (Cited on page 124.)

[125] A. Williams, M. Arlitt, C. Williamson, and K. Barker, *Web Workload Characterization: Ten Years Later*. Springer, 2005, ch. 1, pp. 3–22. (Cited on page 124.)

[126] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz, "Tapestry: A resilient global-scale overlay for service deployment," *IEEE Journal on Selected Areas in Communications*, vol. 22, no. 1, pp. 41–53, Jan 2004. (Cited on page 129.)

[127] S. Ratnasamy, P. Francis, M. Handley, R. M. Karp, and S. Shenker, "A scalable content-addressable network," in *Proc. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. San Diego, California, United States: ACM Press, 2001, pp. 161–172. (Cited on page 129.)

[128] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the XOR metric," in *Proc. Workshop on Peer-to-peer Systems*, ser. LNCS, MIT. London, UK: Springer, March 2002, pp. 53–65. (Cited on page 129.)

[129] V. Ramasubramanian and E. G. Sirer, "The design and implementation of a next generation name service for the Internet,"

in *Proc. Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. New York, NY, USA: ACM, 2004, pp. 331–342. (Cited on page 129.)

[130] R. Cox, A. Muthitacharoen, and R. Morris, "Serving DNS using a peer-to-peer lookup service," in *Proc. Workshop on Peer-to-Peer Systems*, ser. LNCS, MIT. London, UK: Springer Verlag, March 2002, pp. 155–165. (Cited on page 129.)

[131] M. Walfish, H. Balakrishnan, and S. Shenker, "Untangling the Web from DNS," in *Proc. Symposium on Networked Systems Design and Implementation (NSDI)*. Berkeley, CA, USA: USENIX Association, 2004. (Cited on page 129.)

[132] L. Mathy and L. Iannone, "LISP-DHT: Towards a DHT to map identifiers onto locators," in *Proc. Workshop Re-Architecting the Internet (ReArch) (in conjunction with CoNEXT)*, December 2008. (Cited on page 129.)

[133] H. A. Alzoubi, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast CDNs revisited," in *Proc. Conference on World Wide Web*, ser. WWW '08. New York, NY, USA: ACM, 2008, pp. 277–286. (Cited on page 129.)

[134] Z. Al-Qudah, S. Lee, M. Rabinovich, O. Spatscheck, and J. Van der Merwe, "Anycast-aware transport for content delivery networks," in *Proc. Conference on World Wide Web*, ser. WWW '09. New York, NY, USA: ACM, 2009, pp. 301–310. (Cited on page 129.)

[135] M. Artigas, P. Lopez, and A. Skarmeta, "A comparative study of hierarchical DHT systems," in *Proc. IEEE Local Computer Networks (LCN)*, October 2007, pp. 325 –333. (Cited on page 130.)

[136] M. S. Artigas, P. G. Lopez, J. P. Ahullo, and A. F. G. Skarmeta, "Cyclone: A novel design schema for hierarchical DHTs," in *Proc. Peer-to-Peer Computing*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 49–56. (Cited on page 130.)

[137] L. Garces-Erice, E. W. Biersack, P. Felber, K. W. Ross, and G. Urvoy-Keller, "Hierarchical peer-to-peer systems," *Parallel Processing Letters (PPL)*, vol. 13, no. 4, pp. 643–657, December 2003. (Cited on page 130.)

[138] A. Mislove and P. Druschel, "Providing administrative control and autonomy in peer-to-peer overlays," in *Proc. 3rd Workshop on Peer-to-Peer Systems (IPTPS)*, February 2004. (Cited on page 130.)

[139] P. Ganesan, K. Gummadi, and H. Garcia-Molina, "Canon in G major: Designing DHTs with hierarchical structure," in *Proc.*

Distributed Computing Systems (ICDCS). Washington, DC, USA: IEEE Computer Society, 2004, pp. 263–272. (Cited on page 130.)

[140] M. Xu, S. Zhou, and J. Guan, "A new and effective hierarchical overlay structure for Peer-to-Peer networks," *Computer Communications*, vol. 34, no. 7, pp. 862–874, May 2011. (Cited on page 130.)

[141] A. T. Mýzrak, Y. Cheng, V. Kumar, and S. Savage, "Structured superpeers: Leveraging heterogeneity to provide constant-time lookup," in *Proc. 3rd IEEE Workshop on Internet Applications (WIAPP)*. Washington, DC, USA: IEEE Computer Society, 2003, p. 104. (Cited on page 130.)

[142] V. Ramasubramanian and E. G. Sirer, "Beehive: O(1) lookup performance for power-law query distributions in peer-to-peer overlays," in *Proc. Networked System Design and Implementation (NSDI)*. San Francisco, CA, USA: USENIX Association, March 2004. (Cited on page 130.)

[143] M. Soellner, P. Schefczik, P. Bertin, G. Wei, X. Zhang, T.-M.-T. Nguyen, J. Mäkelä, T. Rautio, O. Mämmelä, S. Pérez, A. Eriksson, A.-M. Biraghi, C. Foley, M. P. de Leon, C. Dannewitz, T. Biermann, and M. Marchisio, "Mobility in the future Internet: the 4WARD innovations," 2nd Future Internet Cluster Workshop, June 2010. (Cited on pages 133 and 141.)

[144] R. T. Fielding, "Architectural styles and the design of network-based software architectures," Ph.D. dissertation, University of California, Irvine, 2000. (Cited on page 136.)

[145] K. Scott and S. Burleigh, "Bundle Protocol Specification," RFC 5050 (Experimental), Internet Engineering Task Force, Nov. 2007. (Cited on page 139.)

[146] C. Dannewitz, M. Herlich, E. Bauer, M. Becker, F. Beister, N. Dertmann, R. Hrestic, M. Kionka, M. Mohr, M. Mühe, D. Murali, F. Steffen, S. Stey, E. Unruh, Q. Wang, and S. Weber, "OpenNetInf documentation – Design and implementation," University of Paderborn, Tech. Rep. TR-RI-11-314, September 2011. (Cited on pages 140, 146, and 156.)

[147] Wikitude GmbH, "Wikitude AR Guide," Web source: http://www.wikitude.com/, 2008, last checked: June 2012. (Cited on page 146.)

[148] M. Braun and R. Spring, "Enkin," Web source: http://enkinblog.blogspot.de/, 2008, last checked: June 2012. (Cited on page 146.)

[149] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: Voice-over content-centric networks," in *Proc. Workshop on Re-Architecting the Internet (ReArch)*, Rome, Italy, 2009. (Cited on page 154.)

[150] R. Aguero, B. Ahlgren, P. A. Aranda, T. Begin, A. E. Eriksson, S. Farrel, P. Goncalves, C. Imbrenda, M. Keller, D. Kutcher, A. Lindgren, O. Mehani, B. Melander, B. Ohlman, S. P. Sanchez, H. Puthalath, P. Pöyhönen, S. S. Lor, P. Schefczik, F. Schneider, A. Sefidcon, A. Sharma, O. Strandberg, L. Suciu, and A. Udugama, "(D.A.9) Description of overall prototyping use cases, scenarios and integration points," Deliverable, SAIL 7th FP EU-funded project, June 2012. (Cited on pages 154 and 155.)

[151] B. Ahlgren, B. Ohlman, E. Axelsson, and L. Brown, "Experiments with subversion over OpenNetInf and CCNx," in *Proc. Swedish National Computer Networking Workshop (SNCNW)*, Linköping, Sweden, June 2011. (Cited on page 154.)