Long-Term Location-Independent Research Data Dissemination Using Persistent Identifiers

Dissertation

zur Erlangung des Doktorgrades
„Doktor rerum naturalium"
der Mathematisch-Naturwissenschaftlichen Fakultäten
der Georg-August-Universität zu Göttingen

im PhD Programme in Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Oliver Wannenwetsch (geb. Schmitt)
aus Stuttgart

Göttingen, 2016

Betreuungsausschuss: Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung
Göttingen mbH (GWDG),
Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Jens Grabowski
Institut für Informatik
Georg-August-Universität Göttingen

Dr. Lena Wiese
Institut für Informatik
Georg-August-Universität Göttingen

Prüfungskommission:
Referent: Prof. Dr. Ramin Yahyapour
Gesellschaft für wissenschaftliche Datenverarbeitung
Göttingen mbH (GWDG),
Institut für Informatik
Georg-August-Universität Göttingen

Korreferenten: Prof. Dr. Jens Grabowski
Institut für Informatik
Georg-August-Universität Göttingen

Weitere Mitglieder   Dr. Lena Wiese
der Prüfungskommission: Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Xiaoming Fu
Institut für Informatik
Georg-August-Universität Göttingen

Prof. Dr. Caroline Sporleder
Institut für Informatik
Georg-August-Universität Göttingen

Jun.-Prof. Dr. Marcus Baum
Institut für Informatik
Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 11. Januar 2017

Abstract

Research data occurs in all scientific experiments, computer simulations, observations or as a derivation from other datasets, literature or publications. As a subset of the general concept of digital data, it is classified through its distinct state and its origin. Enriched with descriptive metadata, research data serves as a foundation for discoveries and publishing results in various formats. For citing and linking specific research datasets and publications, unique and persistent identification is necessary. Today, this is realized by Persistent Identifier (PID) systems that provide stable identification for digital entities and an optional annotation by descriptive metadata. Moreover, PID systems abstract the current network location of data in order to anticipate changes in its network location, owed to alternating Uniform Resource Locators (URL) on the World Wide Web (WWW). Applying these concepts, PID systems have tagged billions of research datasets and publications over the past 20 years.

On these foundations, the Handle PID system, known from the Digital Object Identifier (DOI) system, provides reliable access to digital publications and research data to the whole scientific community. While the architecture of the Handle system itself, which depends on fixed network locations, was designed with farsightedness, additional end-user services for PID resolution and management have introduced critical weak spots that can be discovered by comprehensively reviewing the current state-of-the-art.

This thesis focuses on the adaption of location-independent network paradigms which have shown encouraging results when applied to several problems in the domain of decentralized network infrastructures in PID systems. Our first approaches aim at evolving the Handle system design into a self-adjusting system for all major infrastructure services that does not depend on fixed network locations. We tackle this by incorporating strategies and techniques from location-independent network paradigms originating from the current research branch of Named Data Networking (NDN). By this, major weak spots can be eliminated in the Handle PID system and it becomes robust against core infrastructure outages, sudden network topology changes, packet loss and heavy load situations.

The second goal of the thesis is the integration of next generation data dissemination technologies based on location-independent network paradigms into the domain of persistent identifier systems. Therefore, we propose to employ the Handle system for citing research datasets which are disseminated by location-independent technologies based on BitTorrent and NDN. To tackle the trust challenges of dynamic data locations, we create a novel approach for trusted data dissemination in location-independent networks that ensures the authenticity of data as well as the attribution to data issuers. This is done by incorporating the foundations of the Handle PID system and a further format for exchanging complex access information in PIDs.

Acknowledgements

Contents

Curriculum Vitae   **313**

Chapter 1

Introduction

Research data have become more and more important over the past decades. They have grown massively in volume, complexity and importance for all scientific disciplines [1]. As a result, they have become a strategic resource for conducting state-of-the-art research and a burden for curating and preserving them for future use. To maximize the advantage of research data over the time, "long-term" research data access is a key principle for all organizations supporting scientific data activities like data centers, information technology providers and libraries. It is necessary for all data-driven science disciplines for justifying results, sharing irreplaceable data and empowering scientific processes [2]. If research data can be reused, it has the potential to accelerate research work and one can take advantage of former investments in science [3]. This is imminent important for the near future scientific endeavors, where existing research data collections are the foundations for scientific big data and machine learning applications. Hence, the research data management community is aiming at improving data dissemination through enhanced data access and distribution mechanisms [4].

In a more detailed view, the duration of "long-term" ranges from the minimal time span of ten years for justifying results in order to comply with the rules of good scientific practice up to infinite time spans, when data is irreplaceable, as it is the case for observations of the nature [5]. To prepare research data that is or was subject of scientific work and publications for long-term access, a curation process is applied on the data. Within this curation process, the relevant data sets are selected and augmented with descriptive data providing provenience information (metadata).

While data curation and preservation are the necessary foundation, long-term accessibility is necessary to disseminate the data to every interested party. With today's globalized research communities, research data dissemination today equals data accessible from all over the planet using the Internet. Like almost all networks used today for data exchange, the Internet depends on location-dependent data access principles that require the knowledge of the network location for data access. Hence, changes in the network and data organization

makes long-term data access a challenge with a frequently changing data pointer (Uniform Resource Locator (URL)). To overcome these difficulties, Persistent Identifiers (PIDs) have been created by the research data management community to replace changing data pointers with fixed identifiers that are resolved against changing data locations [6].

While the introduction of PIDs has simplified the research data access for scientists accessing PID-tagged publications and data, they introduced new challenges and bottle necks for data curators, publishers and PID infrastructure providers [7]. In this thesis, we investigate the challenges for PID systems and research data dissemination through PIDs generated by today's location-dependent network infrastructure. We provide contributions and solutions to these challenges by transferring and adapting principles from location-independent networks. By this, we remove major obstacles in long-term data dissemination through PID and provide solutions for enhancing the Handle PID system into more efficient and robust long-term data access and identification system.

1.1 Motivation

To enable ubiquitous research data dissemination, research data is attached to the Internet or large community networks through repositories [8]. This *online* dissemination of research data that includes the (processed) data and its associated metadata is done via location-based network technology which is the foundation of almost all Wide Area Network (WAN) technologies.

Sharing research data through networks imposes the challenge to offer data intact, genuine and citable [9] [10]. The first challenge is implied by the design of today's networks that use the *host-to-host principle*, a location-based mechanism for accessing computers or nodes located at the ends of the network. In this type of network, every computer or node owns at least one Internet Protocol (IP) address, in order to be contacted by other network parties. The design of the network is based on the idea to leave intelligent components at the end of the wires, i.e., at the host system side, and to build the network on simple components that only forward packets to all sides. This allows an independent upgrade of network components without further impact on host systems, as long as logic data paths and transmission mechanisms remain compatible [11]. These principle allows scaling out IP networks to the size of the largest network on earth – the Internet. In this network type that was invented in the late 1960s / early 1970s the question *"where is my data located?"* has to be answered every time, when data access is needed. For long-term data access, the challenge of research data localization needs to be solved equivalently, meaning if research data is moved from one host to the other, the question of data localization has to be answered without additional knowledge.

A second challenge arising from the host-to-host principle is that data access has to come through a narrow waist of the IP network [11]. This means that every data access has to be answered by a single host and that there is no caching involved. To improve the

availability of data, multiple approaches have been invented to mitigate the challenges of the host-to-host principle. They often combine multiple distributed data sources that form a smart data distribution network. One very prominent example is a Content Distribution Network (CDN) that utilizes multiple HTTP-servers on the Internet to distribute files, to serve content and to host popular websites [12]. CDNs are very common and form the foundation of large operators such as Akamai, Inc. and Alphabet, Inc. that serve the content of the most-popular websites [13]. A drawback of the CDN network design is that it requires planning and the prediction of content hot spots caused by high demand. Furthermore, a high investment is needed to run own CDN infrastructure or to rent existing CDN capacity with large data volumes. Hence, running a CDN requires efforts, investments, additional knowledge and infrastructure. Therefore it is no practical option for research data repository owners, who can only spend limited resources on data dissemination on the Internet [14].

But besides the attempts of improving the data availability through centrally operated infrastructure, other attempts have been made in the open source and research data management community. A common approach is to create an overlay network that operates on top of the location-dependent IP network. The design of the overlay network is intended to work around or solve limitations of the location-based IP network. These overlay networks are built on location-dependent principles and support multi-host sourcing like GridFTP [15] or even provide location-independent data access through content access like BitTorrent [16]. Another option is to replace the host-to-host principle and the location-boundedness with a new approach that can be found in Information Centric Network (ICN) with its current realization of Named Data Networking (NDN) [11]. Location-independent research data dissemination has been recently introduced into the highly data intensive science disciplines of climate research and high energy physics, but does currently not reflect aspects of long-term data access [17] [18].

For the research data management community granting long-term access to research data is difficult, because network topologies change frequently due to technical evolutions, organizational restructuring or growth [2]. By this, the location of hosts and therefore data also changes which has a significant impact on making data accessible and citable under the same URL for months, years or even decades. As this impact has been observed quite early, several technologies and implementations have been developed to overcome this problem. Domain Name System (DNS) is a mechanism on top of the raw IP addresses that offers a human-readable mapping of names to network entities [19]. But the relationship of IP-addresses and DNS names is fragile and transports problems of domain management to the area of long-term data dissemination [2]. As DNS does not solve the problem of long-term data access in a changing network environment, entity identification systems have been created that tag data on changing locations with a fixed identifier that is long-living and durable through technical measures, organizational rules and independent namespaces.

Nevertheless, almost all significant PID systems still rely on location-dependent network technology and thus share also the disadvantages of IP technology [20]. Furthermore, the most common PID systems do not support location-independent data access technology as resolution targets. As a result, PID systems have to include location-independent technology on the one side and must also enable location-independent data dissemination

through overlay and information-centric network approaches on the other side. Describing, formulating and evaluating those two aspects in order to advance PID technology with location-independent access principles are the goals of this thesis. By this, research data dissemination can be organized decentralized and data access becomes more robust, as data movement does not have an impact on long-term access. Research data can be disseminated through durable PIDs as long as the data is available online and PIDs can be resolved, without central infrastructure as long as a party is resolving them on the network at any location. These decentralized location-independent approaches presented in this thesis allow to reduce the efforts for research data dissemination and to integrate the users of the PID and repository infrastructure into the operation of the platform itself.

1.2 Scope of Thesis

In this thesis, two approaches for location-independent research data dissemination are presented. In the first approach, the central key-concept of PID is decoupled from the necessity of location-based operation. In the second approach, location-independent data dissemination is realized using persistent identifiers as permanent access media. Our considerations base on the assumption that research data is static and consists of a set of aggregated data that is stored jointly with its metadata. Additionally, we consider the assumption that research data management has a very slow change momentum, meaning that existing principles and infrastructures can only be renewed over time, as billions of data sets exists on infrastructure operated by entities all over the planet. Hence, green-field data management approaches and fundamental changes for research data management have no impact on practical system operation. Therefore, as a third assumption, conceptual improvements of research data management have to be incremental and to respect existing principles at a maximum in order to create an impact on practitioners' side.

As introduced in the motivation section, a major challenge for research data dissemination is caused by the location-dependent design principles of today's networks. Thus, we aim on improving long-term research data dissemination through PID using location-independent access techniques, while respecting the assumptions on the changing nature of research data management. This leads us to the major hypothesis that location-independent data access is beneficial for PID systems and provides operational robustness with improved research data dissemination through PID usage. For evaluating this hypothesis, we concentrate on the following research questions concerning the improvement of research data dissemination with location-independent data access through PID:

RQ 1: Can persistent identifier systems benefit in performance and robustness by integrating foundations of location-independent data access?

This leads to the following detailed sub-questions, which will be answered in the thesis:

RQ 1.1: Which requirements exists for extending a PID system towards location-independent data access and operation?

RQ 1.2: How to provide the necessary end-to-end connection principle required by PID systems within a NDN network?

RQ 1.3: How to assure operational and semantic interoperability between location-dependent and location-independent PID systems?

Additionally, a second research question particularly focused on the aspect of improved research data dissemination with location-independent data access is formulated:

RQ 2: Can persistent identifiers become a possible base for improving research data dissemination through location-independent data access?

Again, we split this up into detailed research questions, which are also subject of this thesis:

RQ 2.1: How to construct a persistent identifier model that enables data dissemination for location-independent data access in conjunction with classic location-based data access?

RQ 2.2: Does the operational and architectural integration of this model not impair existing PID systems?

RQ 2.3: Can PIDs help to safeguard trusted data access in location-independent networks?

1.3 Goals and Contributions

Before answering the research questions in this thesis, let us enumerate our contributions to the current state-of-the-art in research data dissemination using persistent identifiers. In contrast to existing work on PID and research data dissemination, the contributions we provide in this thesis do not propose green-field approaches or require major modification to PID systems, but they rather adapt and embrace the slow change momentum of research data management with its billions of PIDs and datasets stored in digital repositories all over the planet. For this, we first focus on conceptualizing, shifting and operating PID systems on location-independent network environments facilitating NDN with our first contribution:

Contribution 1: Location-Independent Persistent Identification of Research Data
We provide a generalized approach for operating a persistent identifier system in a location-independent network infrastructure. By this, we provide a concept of instant PID access without prior knowledge on PID infrastructure locations and we provide better outscaling capabilities combined with an improved infrastructure robustness. Chapter 5 is presenting this contribution.

This contribution consists of following subcontributions:

Contribution 1.1: Instant Handle PID Access Without A Priori Knowledge on PID Infrastructure Locations Using Converged Name Spaces in NDN

For creating, maintaining and resolving PIDs without a priori knowledge of PID infrastructure network locations, we propose a converged namespace model that allows instant access to PIDs in location-independent networks. For this, we introduce a converged name space model for Handle PIDs in NDN networks that augments the Handle PID naming scheme to a first-class global-routable NDN data naming scheme. This contribution can be found in Section 5.3.2.

Contribution 1.2: Selective End-to-End Communication in NDN networks using Interest-based Data Push for PID Management

For maintaining PIDs in distributed systems, access to specific systems in the network is necessary. As NDN is based on a content-centric approach, designed for accessing data from sources using a publish-retrieve cycle, pushing data similar to location-dependent networks from a source to a client requires multiple network round trips. To avoid this, we propose a new polling-free approach for NDN that allows spontaneous data transfers (data pushs) without further preparation from one NDN network node to another, by facilitating NDN interests for data transport. By using our data push techniques, we are able to access specific PID data sources spontaneously within the NDN network. Thus, we can offer all location-based use cases of the Handle PID system within the NDN domain. The results are presented in Section 5.3.3.1.

Contribution 1.3: Location-Independent Multi-Source PID Resolution Using NDN networks

In order to improve the reaction time for PID access, we propose an approach, which benefits from the advantages of NDN. By this, we can create a more robust PID resolution and an improved distribution of administrative metadata within the Handle system. The contribution is contained in Section 5.3.3.2.

Contribution 1.4: Protocol Selection and Data Exchange for NDN Interoperability using Application Protocol Information

As pointed out in the introduction, research data dissemination has a very slow change momentum. Thus, an interoperability between location-dependent and location-independent approach in NDN is inevitable. For this, we present an approach to routing PID Handle-based data between both network domains using a gateway principle. To tackle this, we present a new method for selecting the best suitable protocol for data forwarding based on application protocol metadata. Our contribution fill the gap between the different transport paradigms (packet-oriented vs. content-oriented) for the Handle PID domain. The contribution is shown in Section 5.3.4.

Our further contributions focus on disseminating research data through location-independent access technology using PIDs. For this, we propose as second major contribution:

Contribution 2: Location-Independent Research Data Dissemination Through PIDs
We provide a generalized approach for accessing research data through a location-independent network using Handle PIDs. Our contribution that facilitates the embedding of location-independent access information for BitTorrent and NDN into PIDs removes the need for adjusting the resolution target stored within the PID. By this, a major limitation in the scaleout behavior of the Handle PID system can be overcomed and the creation of maintainance-free PIDs is possible for the first time. The complete contribution is content of Chapter 6.

This contribution consists of following subcontributions:

Contribution 2.1: Extension of the Magnet URI Scheme as a NDN Data Name Container for Storing and Exchanging Access Information with Complex Metadata
As network-locations cannot be used as assurance for trustworthy content in NDN, additional metadata like checksums and digital signatures are important additions in the access information. With the current state-of-the-art, there is no container standard for exchanging complex access information for content stored in NDN networks, beside simple Uniform Resource Identifier (URI) notations limited to NDN data names. For adding complex metadata and access assurance information into NDN access information, we propose an extended Magnet Link URI scheme as a standardized container format for exchanging and storing NDN access information. This contribution is shown in Section 6.3.1.

Contribution 2.2: Embedding Location-Independent Access Information Into PID
The current Handle PID system is limited to provide and resolve location-dependent access information based on URLs. To release this barrier, we provide an approach based on Magnet Links to embed location-independent access information into Handle PIDs. In contrast to existing work on alternative resolution targets for PID, we investigate the impact of PID resolution behavior for extended resolution target data. By this, we can show that our approach has no significant impact on the scaling behavior of the Handle PID system. Section 6.3.3 provides the contribution.

Contribution 2.3: Accessing Research Data Through PIDs Containing Location-Independent Resolution Targets
Based on the previous contributions for location-independent resolution targets in PIDs, we propose an improved PID resolution scheme for accessing research data through PIDs containing location-independent access information. We presented this contribution in Section 6.3.6.

1.4 Impact

During the work of this thesis, following peer-reviewed conference proceeding have been published that contain some intermediate results on transferring location-independent approaches in the domain of PID:

O. Schmitt (Wannenwetsch), T. A. Majchrzak, S. Bingert, "Experimental realization of a Persistent Identifier Infrastructure stack for Named Data Networking", Proceedings of the 15th IEEE International Conference on Architecture and Storage (NAS2015), Boston, USA, Aug. 2015. DOI: 10.1109/NAS.2015.7255207
URL: http://ieeexplore.ieee.org/document/7255207/

O. Wannenwetsch, T. A. Majchrzak, "On Constructing Persistent Identifiers with Persistent Resolution Targets", Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS2016), pp. 1031-1040, Gdańsk, Poland, Sep. 2016. DOI: 10.15439/2016F87
URL: http://ieeexplore.ieee.org/document/7733372/

Furthermore, the author has contributed to following peer-reviewed journal articles in the field of research data management:

T. Kálmán, D. Tonne, **O. Schmitt (Wannenwetsch)**, "Sustainable Preservation for the Arts and Humanities", New Review of Information Networking, pp. 123-136, Vol. 20, Issue 1-2, 2015. DOI: 10.1080/13614576.2015.1114831
URL: http://dl.acm.org/citation.cfm?id=2859726.2859740

O. Schmitt (Wannenwetsch), T. A. Majchrzak, "Document-Based Databases for Medical Information Systems and Crisis Management", International Journal of Information Systems for Crisis Response and Management (IJISCRAM), pp. 63-80, Vol. 3, Issue 4, 2013. DOI: 10.4018/ijiscram.2013070104
URL: http://www.igi-global.com/article/document-based-databases-for-medical-information-systems-and-crisis-management/96922

The author has contributed to following peer-reviewed conference proceedings in the field of research data management:

H. Kusch, **O. Schmitt (Wannenwetsch)**, B. Marzec, S. Y. Nussbeck, „Datenorganisation eines klinischen Sonderforschungsbereiches in einer integrierten, langfristig verfügbaren Forschungsdatenplattform", Proceedings der 60. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (GMDS), Krefeld, Sep. 2015. DOI: 10.3205/15gmds104
URL: http://www.egms.de/static/de/meetings/gmds2015/15gmds104.shtml

O. Schmitt (Wannenwetsch), P. Weil, P. Wieder, S. Y. Nussbeck, „Integrierte Portalumgebung und verteilte Echtzeitsuche für medizinische Langzeitarchivierung", Proceedings der 59. Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (GMDS), Göttingen, Sep. 2014. DOI: 10.3205/14gmds014
URL: http://www.egms.de/static/de/meetings/gmds2014/14gmds014.shtml

Additionally, the author has contributed to following peer-reviewed conference proceedings outside the scope of this thesis:

N. Campos-López, **O. Wannenwetsch**, "The PERICLES Process Compiler: Linking BPMN Processes into Complex Workflows for Model-Driven Preservation in Evolving Ecosystems", Proceedings of the 12th International Conference on Web Information Systems and Technologies (WEBIST2016), pp. 76-83, Rome, Italy, Apr. 2016. DOI: 10.5220/0005759800760083
URL: http://www.scitepress.org/DigitalLibrary/PublicationsDetail.aspx ?ID=E24vyfHKFm8%3d&t=1

The author has supervised two master theses related inside and outside the scope of the provided research work:

A. Wildschütz, „Transformation einer Persistent Identifier Infrastruktur zum Management globaler Namensräume in Named Data Networking", Master Thesis, Institute of Computer Science, University of Göttingen, 2016.

M. Hellkamp, "A hierarchical File-System for the CDSTAR Object Storage Interface", Master Thesis, Institute of Computer Science, University of Göttingen, 2015.

1.5 Structure of Thesis

This thesis is structured as follows. We start with the foundations in Chapter 2, where we introduce the field of research of the thesis and its terminology. The terminology covers the area of research data management (cf. Section 2.1), digital research repositories (cf. Section 2.2), persistent identifiers (cf. Section 2.3), and its realization in the Handle persistent identifier system (cf. Section 2.4), as well as the Magnet Link scheme (cf. Section 2.5).

In Chapter 3, we motivate the thesis' contributions by pointing out the problem statements in the context of research data dissemination in location-dependent networks in conjunction with persistent identifiers. For this, we first look at the challenges introduced by the location-dependent network principles and its impact on the infrastructure of Handle PID systems.

In Chapter 4, we refer to scientific work that is related to this thesis, in order to put our contributions into a broader research context. This chapter is divided into existing work on research data dissemination facilitating overlay networks (cf. Section 4.1), and Named Data networking (cf. Section 4.2). Furthermore, it refers to previous work on persistent identifiers in named data networks (cf. Section 4.3), archive access naming schemes derived from

Magnet Links (cf. Section 4.4) and running location-dependent legacy network application in NDN networks (cf. Section 4.5). We conclude Chapter 4 with a description of the research delta provided in this thesis.

Chapter 5 contains the details of the first approach on location-independent persistent identifiers in this thesis. First, we explain the foundations of the paradigm shift from the location-bound PID as-is situation to new location-independent possibilities in Section 5.1. Then, we list the benefits of our approach in Section 5.2 and continue with a detailed explanation of it in Section 5.3, where we present the PID and NDN name space conversion (cf. Section 5.3.2), the different location-independent access models (cf. Section 5.3.3) and an interoperability model for an interaction of our PID approach using the existing location-dependent Handle PID foundations (cf. Section 5.3.4). We validate our approach with an implementation and an evaluation. The implementation of our access model approaches is provided in Section 5.4. In Section 5.5, we verify our contributions employing real-world PID resolution data in a simulated NDN test bed that we describe in Section 5.5.1. The processing of the resolution data sets is described in Section 5.5.2. Then, we compare the different access models with the current state-of-the-art Handle PID system for normal and faulty network conditions in Section 5.5.3 and Section 5.5.4.

In the following Chapter 6, we explain our approach on accessing location-independent research data through PIDs. For this, we point out first the impact of PID distributed maintenance efforts caused by the current principles of all PID systems linking to location-dependent resolution targets (URLs) (cf. Section 6.2). Then, we explain our approach in detail on creating maintenance-free PIDs that contain location-independent resolution targets which are persistent by design and do not require current adjustments like URLs (cf. Section 6.3). To explain our method in detail, we first start with the approach of extending the Magnet URI scheme to augment it into a container format for storing and transporting NDN access information (cf. Section 6.3.1). By this, our extended Magnet URI scheme allows transporting location-independent access information within Handle PIDs. We explain the embedding of Magnet Links into Handle PIDs in the subsequent Section 6.3.3. As this has particular impact on the data access service chain that is an essential part of a (Handle) PID system, we investigate the impact of our approach on the PID resolution (cf. Section 6.3.4). Afterwards, we point out the approach on maintaining and creating PIDs following our approach (cf. Section 6.3.5). As almost all PID resolutions conducted through the Internet by end-users are using state-of-the-art location-based network technology, we look at the data access using PID from location-based networks in Section 6.3.6. To evaluate our feasibility of our approach and to assess the impact on the Handle PID infrastructure and its operation, we provide an implementation of our approach in Section 6.4. In this section, we first look at the implementation at the server side (cf. Section 6.4.1) and afterwards shift the focus to the client side (cf. Section 6.4.2). Finally, we evaluate the approach and its materialization in the implementation in an own evaluation section (cf. Section 6.5).

In Chapter 7, we discuss the result of the approaches and its evaluations. For this, we answer the research questions and highlight our contributions in Section 7.1 and 7.3. Afterwards, we point out the limitations of our approaches in Section 7.2 and 7.4. Finally, in Chapter 8, we conclude the thesis and provide an outlook on potential future work.

Chapter 2

Foundations

In this chapter, we present the foundations of this thesis which cover basic concepts, terminologies and definitions. We start by introducing the principles of research data management and digital repositories. Then, the central idea of persistent identifiers is introduced. Finally, the design principles behind content-driven location-independent network technologies are explained with all essential details which are necessary to follow the contributions of this thesis.

2.1 Research Data Management

In order to make research data ready for use in new research processes and to use digital data sets for scientific result verification, different steps such as data selection, preparation, metadata annotation and publishing are necessary for long-term access [21]. Let use first introduce the definition of research data management used at the University Libraries of Boston, in order to understand these steps better [22]:

Definition 2.1 (Research Data Management) *"Research data is data that is collected, observed, or created, for purposes of analysis to produce original research results. […] Research data can be generated for different purposes and through different processes, and can be divided into different categories:"*

1. *"**Observational:** data captured in real-time, usually irreplaceable. For example, sensor data, survey data, sample data, neurological images."*
2. *"**Experimental:** data from lab equipment, often reproducible, but can be expensive. For example, gene sequences, chromatograms, toroid magnetic field data."*
3. *"**Simulation:** data generated from test models where model and metadata are more important than output data. For example, climate models, economic models."*

4. *"**Derived or Compiled:** data is reproducible but expensive. For example, text and data mining, compiled database, 3D models."*
5. *"**Reference or Canonical:** a (static or organic) conglomeration or collection of smaller (peer-reviewed) datasets, most probably published and curated. For example, gene sequence data banks, chemical structures, or spatial data portals."*

The definition of *curation* given by [4] emphasizes the level-approach that is followed here to focus on the scientific activities performed in this thesis. The distinction of research data management levels is depicted in Figure 2.1.

Definition 2.2 (Curation) *"The activity of managing and promoting the use of data from its point of creation, to ensure it is fit for contemporary purpose, and available for discovery and re-use. For dynamic datasets this may mean continuous enrichment or updating to keep it fit for purpose. Higher levels of curation will also involve maintaining links with annotation and other published materials."*

Furthermore, the data needs to be intact without any physical or semantic damage which allows a successful reconstruction of the data sets. Curation of data means that the bits are *preserved* and the minimal requirements of good scientific practice are met regarding the archiving of data sets. Still, this lowest standards needs periodical activities, i.e. by verifying the physical intact state of data and the encoded information [4] (cf. Figure 2.1).

Definition 2.3 (Archiving) *"Archiving is a curation activity which ensures that data is properly stored, selected and can be accessed and that its logical and physical integrity is maintained over time, including security and authenticity."*

In this case of archiving the data sets will be opened to be usable by other persons or entities, such as research institutes. In this state, the digital research information is *preserved* and can be integrated into the scientific practice and be can *cited* in publications. The notation of preservation used in this thesis is defined by [4] as follows:

Definition 2.4 (Preservation) *"Preservation is an activity within archiving in which specific items of data are maintained over time so that they can still be accessed and understood through changes in technology."*

Furthermore, the preserved data is reusable and can be subject of verifying results and feed new research processes. To reach this state, research data has to be augmented with *metadata* that describes the data sets, its authors, the origin and gives information how to interpret the data. For citing it, a unique data identification has to be added that should exist for an infinite time space or at least for a time span, when research data has to be reusable and citable. In this setting, PIDs are used as a concept for creating and maintaining long-living identifiers for digital entities.

In the context of research data management, we further need to introduce the terms of *Metadata* and *Meta-Metadata*. It is important that the definition of metadata has two distinct meanings in these contexts. The first meaning of metadata is the more natural one, where metadata describes scientific data sets, i.e., through adding author information. This is given in Definition 2.5. The second meaning of metadata aims at describing data by a (data) model [23], i.e., through adding a revision number to a data scheme [24]. We introduce this meaning in Definition 2.6 and refer to it as *meta-metadata*.

Definition 2.5 (Metadata) *"There is a large amount of information which describe the data in statistical and scientific database that are kept in an ad hoc fashion in log books, text files, or even hand-written notes. Examples are the failure logs of devices, the date and method used in generating a new analyzed data set, the identity of people who generated the data sets, the description of materials that were encoded in the database, the data units used, etc. Such information, which is referred to collectively as metadata, can be quite complex and is just as important as the database itself for analysis purposes. In addition, such metadata are particularly important for archival purposes." [25]*

Definition 2.6 (Meta-Metadata) *"The metadata capturing the physical characteristics or structure of the data and the metadata associated with the logical interpretation of the data. The physical [Meta-] metadata allows us to decode the raw bits into integers, reals, and other structures." [26]*

After introducing the (meta-) metadata concept, we can now have a look at the next levels defined by Lord et al. [4]. The highest levels that can be achieved in (digital) data curation are *integration* and *preservation* which use citable resp. reusable data (cf. Figure 2.1). Integrated data is available with a full set of metadata, long-term-valid identifiers and a machine-readable interface (e.g. a web services) that allow an easy connection of data consuming services. Additionally, the metadata use a community-acknowledged or broadly used metadata scheme that enables interoperability in data usage among a large group of information systems. This is the level where *research data curation* takes place as it is defined in [27]:

Definition 2.7 (Research Data Curation) *A set of activities that aims at the point where the research is finished [...] some results are available for public viewing. Characterized by fewer items, more metadata, statically derived.*

As the data consumers are interested in the integrity of data and its accessibility the **content** is most important and not the place where data is stored or how data is transferred between source and requester, as long as it works fast, reliable and secure. This fact is condensed in the **what principle**, where only content matters and network transmission and connectivity is a necessary circumstance for content dissemination of digital information [11]. Hence, accessibility of curated research data can be condensed into three major challenges according to the literature:

1. Intact, discoverable an accessible content is the foundation for long-term data preservation.
2. Data accessibility is realized through connecting archived data to networks.
3. Data localization is a major challenge introduced by today's location-dependent networks.

In addition, the presentation level is the result of an information system which uses integrated data sets and offers interfaces to provide an instant presentation of research data that make data sets visible and browsable for interested parties. In this level research data is often highly aggregated to certain aspects, categories or dimensions (e.g. correlations between time, space and observations). Figure 2.1 summarizes the different levels of digital curations.

| presentation | visible, instant presentation |

| integrated | new research tools, new methodology |

| citable, reusable | content preservation, publication, integrated into scientific practice |

| archived, content accessible | bit preservation, minimal requirements for good scientific practice |
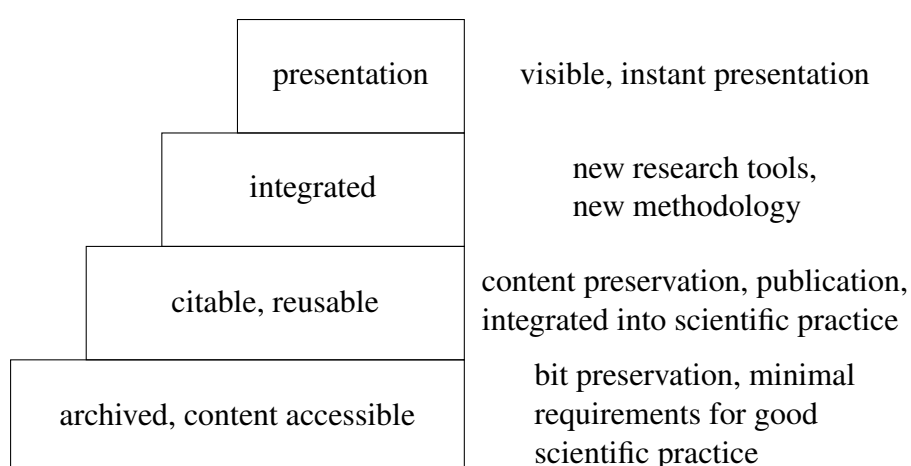
Figure 2.1: Levels of Digital Data Curation (adapted from [21])

2.2 Digital Data Repositories

In order to perform research data curation, information system engineers and researchers have to create specialized information systems that are able to store research data jointly with the respective metadata. These information systems are defined as *Digital Research Repositories* by Heery and Anderson [28] and provide a minimum set of basic operations such as *put, get, search* combined with an access control mechanism. With these basic operations, they provide two major groups of functionality. The first group of functionality is centered towards research data curation. For this, data repository software provides mechanisms to select, manage and verify research data in order to provide the basic levels of bit and content preservation. Furthermore, as part of the curation activities, they allow to assign and manage metadata sets. To make data accessible on a global layer, digital repository software allows the assignment of a persistent identifier to curated data sets, too. The second group of functionality is centered towards the accessibility and presentation of

curated research data. With the basic operations presented before, the repository software expose curated research data and its metadata over a network and often provides specialized presentation layers and search catalogues for browsing and selecting relevant data sets.

To assess the impact of the approaches presented in this thesis on research data curation, we have to find measurements that reflect the current importance of research data management on an international level. Unfortunately, it is very hard to estimate an overall amount of research data that is tagged with PIDs. Therefore, we only highlight the growing importance of research data and digital repositories by the number of public known repositories. The re3data project, an EU-founded project to explore, count and consult digital repositories and digital infrastructure for digital curation emphasizes the growing importance of digital research data. Between August 2012 and August 2014, digital repositories have been indexed by the re3data project [29]. As we can see from Figure 2.2, more than 1000 digital repositories have been indexed in the European Union, leading to the fact that each member country runs dozens of digital repositories for research data as national efforts. Hence, if we assume a four digit number of digital data sets per repository, we can estimate billions of data sets that are subject of data curation and PID assignment.
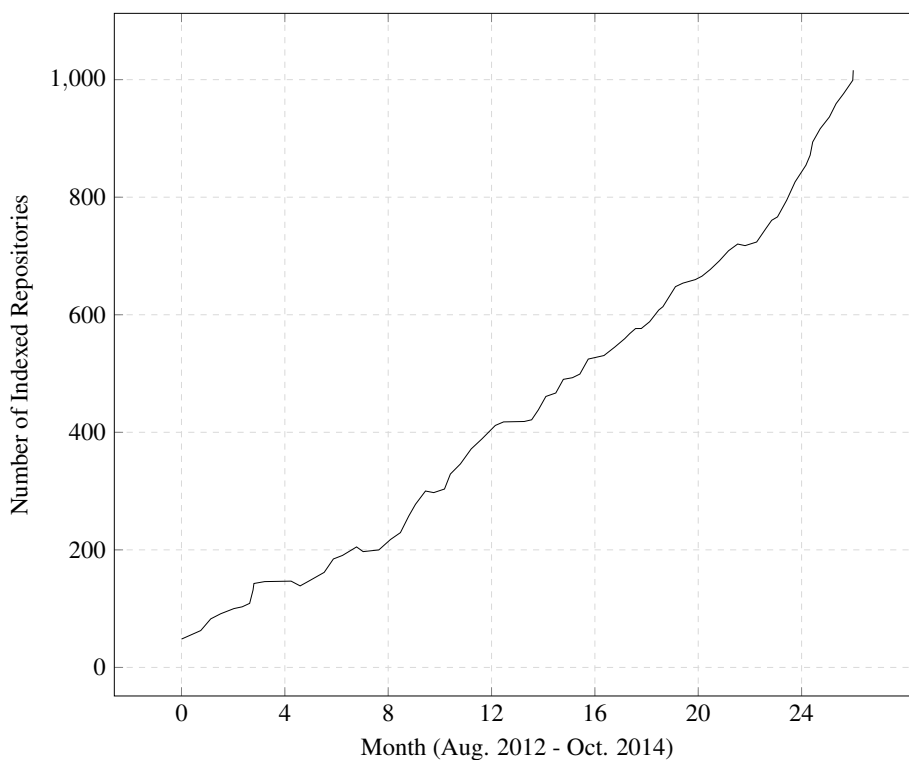
Figure 2.2: Indexed research data repositories by re3data.org [29]

2.3 Persistent Identifiers

A key concept used in this thesis is the Persistent Identifier (PID). Therefore, we introduce the technical term by the definition of E. Tonkin that is widely used in literature:

Definition 2.8 (Persistent Identifier (PID)) *"Persistent identifiers are simply maintainable identifiers that allow us to refer to a digital object – a file or set of files, such as an e-print (article, paper or report), an image or an installation file for a piece of software. [...] unlike a simple hyperlink, persistent identifiers are supposed to continue to provide access to the resource, even when it moves to other servers or even to other organizations."* [20]

Most important for PIDs are the terms *identification* and *persistency*. *Identification* means to connect the identity of a digital object to a respective technical identifier. This technical identifier is living in an own namespace and secured against unintended changes. In the majority of PID systems the validity of technical identifiers follows a defined life cycle, which is embedded into the operational model of a *PID operator* and the connected policies and rules. The term PID operator can refer to an organization that is operating the infrastructure for running a PID service (with servers, networks and all other necessary items) or to an organization which is creating and maintaining the PIDs. In practical applications PID operators are fulfilling both functions simultaneously or execute certain functions in behalf or digital repository owners. The form and semantic of the identifier is also part of the policies and rules of the PID operator. Often an opaque identifier (dumb number) is chosen that has no deeper semantic except of organizational identifier segmentation. The principle of *identification* requires a uniqueness of the identifier in the respective namespace. By maintaining an own namespace that is free of overlapping in a common domain of data curation, PIDs ensure the validity of their identifiers over changes in technology, network topology and organizational changes [6].

But in order to make a persistent identification scheme a persistent identifier, general assertions of the PID concept have to be kept at all time to assure its durability. These assertions can be found, in the fundamentals of the Document Object Identifier (DOI) system, the most common PID system used in academia, publications and libraries today [30] [31]. The foundations of the DOI system require the following measures that are accompanied by organization and social measures [32] [33]:

1. *"A syntax specification, defining the construction of a (PID) name [...]."*
2. *"A resolution component, providing the mechanism to resolve the (PID) name to data specified by the registrant."*
3. *"A metadata component, defining an extensible model for associating descriptive and other elements of data with the (PID) name."*
4. *"A social infrastructure [...]."*

The concept of *persistency* has three meanings for PIDs. The first one is the persistency of the PID as a digital object itself, which means that PIDs do exist infinitely at best and follow

a controlled life-cycle with distinct state from creation to the end of their lifetime. Within this life-cycle, all PIDs have to stay in a defined state and are not allowed to vanish. PIDs which reach the end of their life-cycle continue to exist but are assigned to a semantic state of invalidity. This invalidity can be realized by a tombstone website, if the PID is resolved via Hypertext Transfer Protocol (HTTP). The second meaning refers to the identity of the PID. The identifier also has a certain life-cycle and should always be in a *defined state*. A defined state in this context means that the PID operator has to check and adjust the identifier regularly to reflect the actual real condition of the PID. Hence, an undefined state refers to a PID that is identifying or referring a missing or invalid resource, without stating this fact. PIDs in an undefined state appear broken to users and may result in unresolvable identifiers or data referencing to resources that have been deleted or moved to an unknown location. This is particularly important for updating an identifier and is implemented differently in PID systems. Often a forward from the old identifier to the new identifier is used. The third meaning refers to the persistency of PID data. All PIDs that are subject of Definition 2.8 contain at least a resolution target as associated data record. This associated data has to be updated as often as the operational rules and guidelines are demanding it and is within inside the PID system. The resolving target, often stated as *target URL*, has to be updated in order to point to the current valid location of the digital object. If digital objects are moved, the resolving target has to be updated, too. Some PID and/or repository operators are checking the validity of PID data and maintain updating processes for the PID and associated data.

Figure 2.3 illustrates the principle behind PIDs in the context of linking scientific publications to digital objects. The scientific publication on the left side contains a PID, which is in this case a DOI. This PID is used to link the publication to a specific digital object. The assignment of the PID to the digital object has been done independently from the creation of the publication. If a reader of the publication is interested in the digital object cited in the publication, he or she can use a PID infrastructure to resolve the opaque identifier into a data pointer (in most systems a URL). With the data pointer, the digital object can be accessed over network. If the network location changes over time, the PID is adjusted from the past to the *current URL*. This process is repeated, every time the data object is moved to a new place. Hence, the next iteration will move the data pointer from the *current URL* to the *future URL*.
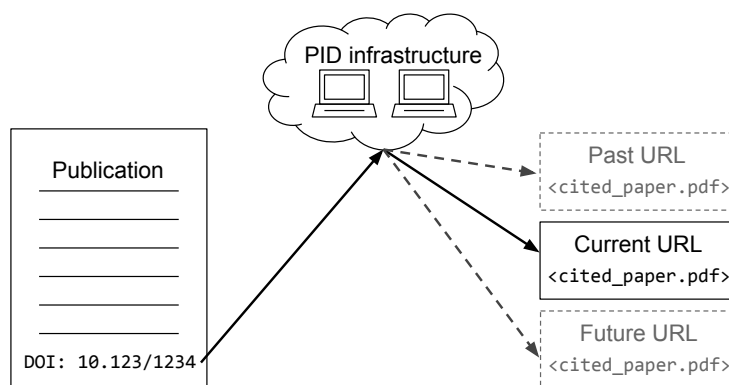
Figure 2.3: Working Principle of PID systems

2.4 Handle System

After describing PID in general, we now want to introduce the Handle PID System. It is used in this thesis to verify our approaches with real-world infrastructure and to implement the suggested approach in a prototypical system for evaluation. The thesis uses productive Handle PID systems hosted at GWDG in the area of European Persistent Identifier Consortium (EPIC) to gather strictly anonymized metadata [34], perform measurements and draw samples from large PID populations hosted by GWDG customers and associated PID-data centers. Let us now first have a look at the system foundations of Handle. The Handle System, initially released in 1994, consists of a set of specifications that allow the creation of a distributed information system infrastructure for tagging and resolving PIDs on digital objects. It realizes PIDs by building up a distributed key-value-like data base system that stores entries with a distinct name that resolvable in a global namespace. Those values can be grouped together into sets, called *Handles*. In the terms of the Handle system, a PID consists of a named Handle that contains multiple Handle values, where one value is a target URL. The Handle value with the target URL is needed to resolve the PID for accessing digital objects linked by the PID (cf. Figure 2.3). Through maintaining the link between the identifier and the digital object, it offers the possibility to have long-living data names that overcome the problem of changing network locations. The Handle specifications include protocols and mechanisms for setting up a distributed system set that stores, manages and resolves PIDs [35]. The system itself is described by the specifications that are available as Request for Comments (RFC) from the Internet Engineering Task Force:

- RFC 3650: Handle System Overview [36]
- RFC 3651: Handle System Namespace and Service Definition [37]
- RFC 3652: Handle System Protocol (ver 2.1) Specification [38]

RFC 3650 describes the fundamentals of the Handle System and gives an explanation of the system architecture. RFC 3651 depicts the definition of the namespace and the roles of the infrastructure parts. RFC 3652 explains in detail the native Handle Protocol, the transport of Handle messages via User Datagram Protocol (UDP) and Transmission Control Protocol (TCP), as well as the management of sessions for user authentication and data transport encryption.

The core system has been designed as a very robust and resilient infrastructure with no single point of failure. By design, it is scalable and allows the inclusion of additional servers at any time to tackle high load situations and to provide failover capabilities. Furthermore, it includes cryptographic trust mechanisms that secure PID resolution and maintenance and it integrates everything into a open, well-documented system that provides transparent abstraction to the user about the architectural and technical details [39]. The International DOI Foundation (IDF) is operating the DOI System on base of the Handle System and introduces own guidelines and policies for usage and operation [33]. Thus, technical properties for the Handle system can be transferred to DOI system and vice versa.

The Handle system and the DOI system are integrated in the same namespace and part of the same worldwide infrastructure. As a result, servers operated for DOI by the IDF are able to resolve Handle PIDs and Handle resolvers linked to the global Handle system are able to resolve DOIs. In Table 2.1, the main properties of the Handle System are listed.

Property	Availability	Source
unique identifier	yes, if connected to a central infrastructure	[36]
user base	1000 servers in 75 countries, DOI >100 mio handles with 12k registrants	[30] [31]
standardization	yes, by IETF	[36] [37] [38]
metadata support	yes	[36]

Table 2.1: Properties of the Handle System

For assigning PID identifiers that are resolvable and manageable in a distributed manner, the Handle System features a namespace, which is split into a global and a local part. Figure 2.4 gives an overview on the naming scheme in Handle. The global name part, identified by a *Handle Prefix*, in literature also called *Naming Authority (NA)*, splits the global namespace into local sub-namespaces. Local sub-namespaces can be grouped together in the prefix, by introducing a dot as separator (American Standard Code for Information Interchange (ASCII) code 0x2E). This grouping is performed in the DOI system, where all prefixes start with a Directory Indicator (DI) "10." [40]. But also new Handle prefixes assigned by Data Oriented Network Architecture (DONA), as consortium for global PID infrastructure operation assigns use a grouping policy like DOI using a dot separator. The local sub-namespaces that are identified through a *Handle Suffix* can assign own IDs for their local entities. Local administrators have to make sure that local assignments are in line with the Handle policy PID requirements. A full Handle PID ID consists of a concatenation of the different identifying parts. Thus, the Handle PID qualifies itself as a *unique indentifier* if the Handle System installation is attached to the central infrastructure in order to assure an overlap free assignment of PID identifiers.

```
10.1145/973097.973100
```

        Naming   Unique
       Authority   Local Name
      (NA) (Prefix)   (Suffix)
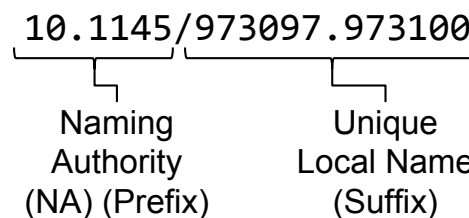
Figure 2.4: Handle Naming Scheme

In order to explain the approaches presented in this thesis, a short introduction into the construction of a Handle is needed. As defined above, a Handle is a set of values that is accessible trough an identifier. In Figure 2.5, the schematics of a Handle is depicted. A Handle consists of a set of typed, indexed key-value pairs that store the data record-by-record [36] [37]. The data is stored as hierarchical records enumerated by an index. The key-value pairs store sequences of binary octets that are terminated by a 4-byte unsigned int length marker. Strings are represented as UTF-8 encoded strings.

```
<index>        : 3
  <index>        : 2
    <index>        : 1

    <type>         : URL
    <data>         : https://gwdg.de/...
    <TTL>          : {Relative: 24 hours}
    <permission>   : PUBLIC_READ, ADMIN_WRITE
    <timestamp>    : 927364617838
```
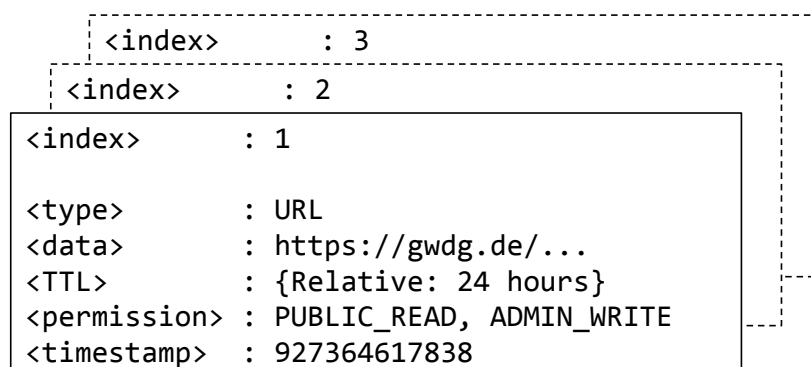
Figure 2.5: Handle With Its Associated Values (based on [37])

The separation into global and local namespaces in Handle is also extended to the architecture of the system (cf. Figure 2.6). All assignments of prefixes are stored in global available Handle servers on the Internet, forming the Global Handle Registry (GHR). To share the responsibility of prefix assignments, the GHR is operated as Multi Primary Administrator (MPA) GHR [41]. This means that the GHR is operated as a distributed system by different organizations, in order to prevent an organizational overweight of a single infrastructure operator. DONA is coordinating the operation of the MPA GHR. DONA is governed by international stakeholders and experts and operates in cooperation with the International Telecommunication Union (ITU), the United Nations' specialized agency for information and communication technologies. Each GHR is operated by an own organization. Corporation for National Research Initiatives (CNRI) is an MPA GHR operator located in the US, the GWDG is located in the EU and operates the MPA GHR in behalf of EPIC, while the Chinese Handle Coalition (CHC) is operating a MPA GHR in China. The GHR is replicated to multiple sites, in order to prevent data loss in case of data center disasters. Moreover, the redundancy is used for load distribution and maximizing uptime in case of maintenance or disaster. The GHR database stores the network locations (IP-addresses) of all official Local Handle System (LHS). The LHS is responsible for all Handle PIDs that belong to a certain Handle prefix (local sub-namespace) and are under local administration of PID operator or PID service providers. Additionally, the verification information based on cryptographic certificates is stored in the GHR to make LHS responsibility for a certain prefix verifiable publicly. The LHS stores the actual PID information in a local Handle server systems with an attached database system. Often

operators of LHS cooperate together by mirroring their LHS site information in order to prevent data loss or provide higher query and update capacities through load sharing [42].
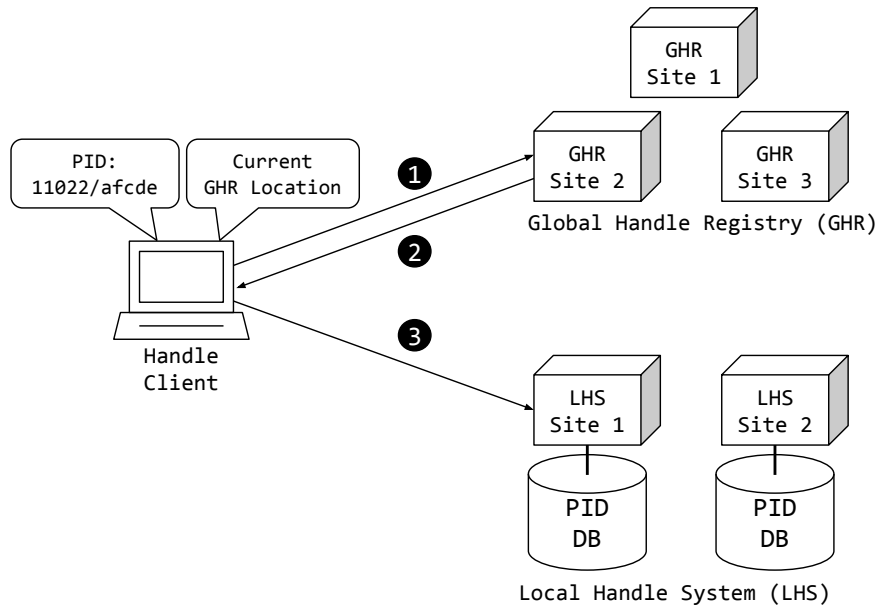
Figure 2.6: Handle System Architecture with Client Interaction

In Figure 2.6, the process of client interaction is depicted. For the interaction with the Handle System, a client can use the native location-dependent Handle protocol that uses port 2641 for UDP and TCP transport [38]. Before resolving, accessing or changing the PID, the client needs the full PID and the current network location of the GHR. This a priory bootstrap information is needed within the Handle system to start interaction. When using the Handle software, bootstrapping information are included as static data set in the Handle software bundle. Then, the client contacts the GHR in step ❶ in order to obtain the location of the LHS and the optional verification information, to assure that the LHS has been authorized to answer requests for the local sub-namespace. After responding with the LHS access information in step ❷, the client contacts the primary LHS site and sends its request for processing the Handle in step ❸.
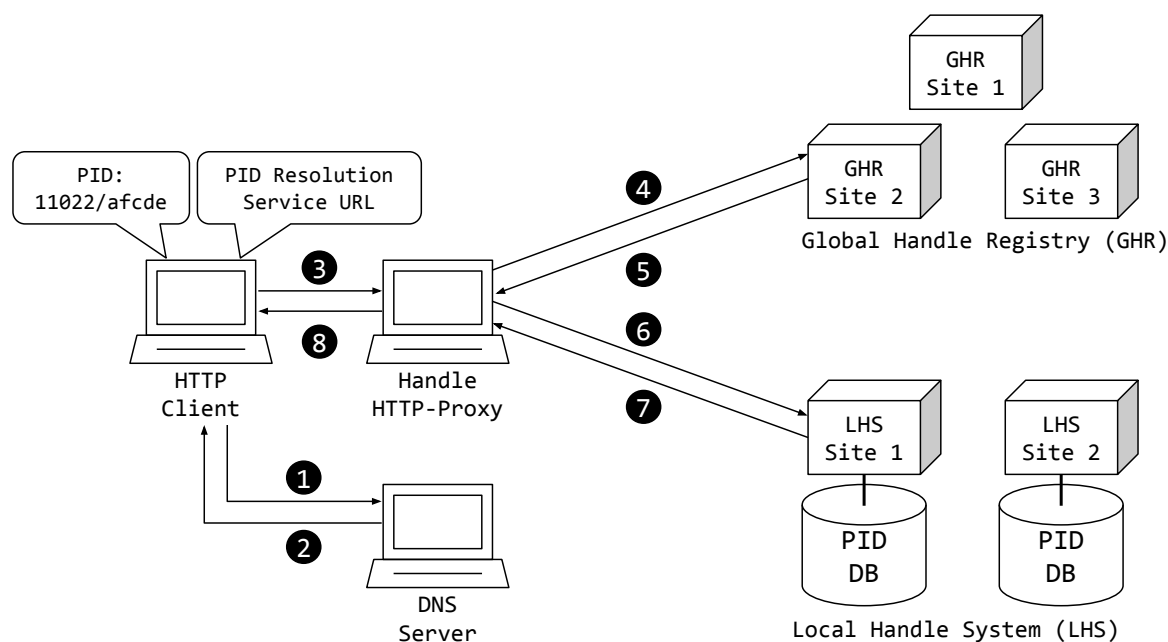
Figure 2.7: Handle System Architecture with Client Interaction Using a Handle HTTP-Proxy

Although, the Handle system has been designed as a distributed PID system with almost no external dependencies at its core like DNS, it comes with an entry barrier for application integration. This entry barrier is the adaption of the complex native Handle protocol. To make the adaption of Handle-based PID systems easier and to support PID resolution from the web browser, HTTP-based proxy systems have been added to the Handle infrastructure. The HTTP-based proxy system allows resolving Handle PIDs into target URLs using HTTP forwarding. This feature is a real-world example of the switching between location-independent persistent ID spaces and location-based resolution targets as shown in Chapter 3. In Figure 2.7, the working principle is depicted. As we can see, the working principle is identical to Figure 2.6, but two new parties have been added. The *Handle HTTP-Proxy* acts as an intermediary between the location-dependent network of the user and the Handle space. Hence, the proxy has two interfaces, a HTTP-interface for the HTTP-client communication and a Handle interface for the Handle communication to the GHR and LHS servers. In this scenario, a *DNS Server* is needed, because a user accesses the Handle HTTP-proxy using a URL with a DNS domain. As stated in Chapter 3, the DOI and Handle operators are offering official HTTP-proxies at the domains dx.doi.org or hdl.handle.net. In order to understand the HTTP-proxy, we have a look at the HTTP-client interaction in Figure 2.7. In this figure, we look at each step of the PID resolution using a Handle proxy server via HTTP. In step ❶, HTTP-client, e.g., a web browser, uses the central resolution service URL hdl.handle.net to resolve its PID 11022/afcde. In this step, the domain hdl.handle.net is resolved into the IP-address of the Handle HTTP-proxy. The resolution process contains a round robin selection that returns one IP-address out of five other addresses linked to the other five official proxy

servers (cf. Figure 3.2 and Chapter 3). After delivering the IP-address to the client in step ❷, the client connects to the HTTP-proxy and passes the PID as resource in the `GET` command of the HTTP request [43] (step ❸). Then, the HTTP-proxy contacts the GHR in step ❹, in order to obtain the location of the LHS and the optional verification information, to assure that the LHS has been authorized to answer requests for the local sub-namespace. After responding with the LHS access information in step ❺, the HTTP-proxy contacts the primary LHS site and sends its request for processing the Handle in step ❻. Now, after the resolution of the PID, the Handle record with the target URL is responded to the HTTP-proxy (step ❼). After obtaining the target URL, the HTTP-proxy integrates the target URL into a HTTP redirection that is sent as a response to the HTTP client (step ❽). The client parses the response and follows the redirection URL and as a result, the web browser display the resource that has been the result of the PID resolution. This process is also identical for reading any other Handle values or administrative information from the Handle system using the official HTTP-proxies.

In addition to Figure 2.7, we provide in Figure 2.8, a Unified Modeling Language (UML) sequence diagram. It illustrates the HTTP-based PID resolution in detail. As we can see from the figure that a pipeline with six tiers is needed to resolve a PID and to access the research data in a location-based network through a PID.
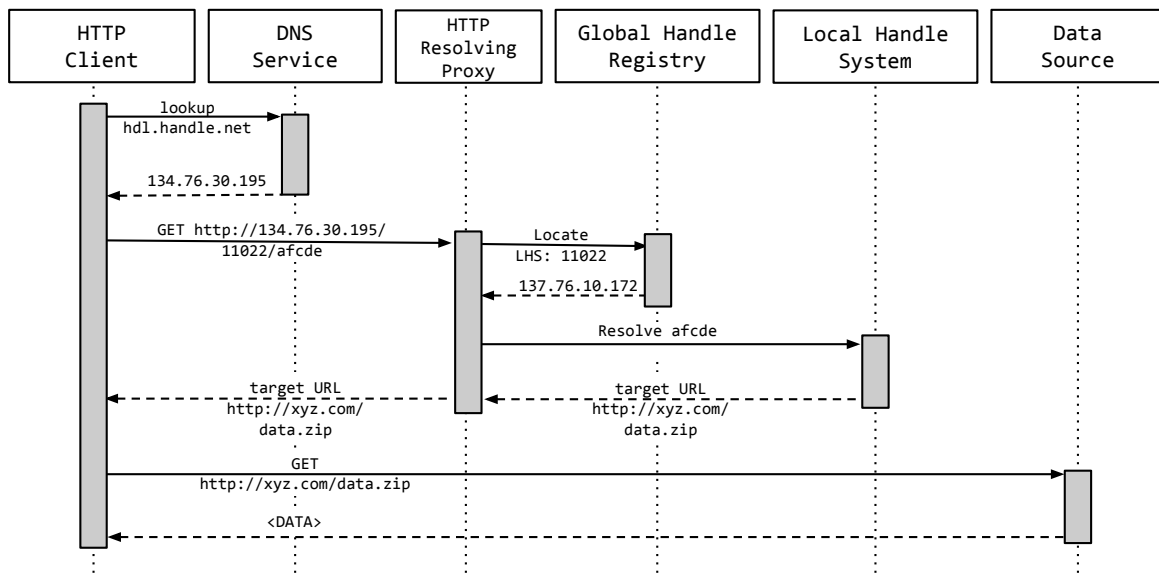
Figure 2.8: HTTP-based Location-Dependent Resolution And Data Access Using Handle PID

The usage of the HTTP-based proxy system also requires additional information concerning the network location of the HTTP proxy system that is located in the location-dependent network. In Figure 2.9, the augmented PID name scheme is depicted that adds a protocol scheme indicator (http://) and the URL of the Handle proxy service (hdl.handle.net) in front of an existing PID. The augmentation of the PID shows the

contradiction to the PID main purpose and the problem caused by non-persistent URLs. Also, the protocol scheme is subject to technical change and the usage of unencrypted communication may be disregarded in the future. Hence, challenges and problems of the location-dependent network on technical access become challenges for the PID system in use cases involving end-user. Thus, the augmentation may bring practical advantages, but endangers the persistent nature of all PIDs. As a result persistent citing of a PID on a media should only use the prefix and the suffix and not the complete HTTP proxy URL that includes http(s)://hdl.handle.net.
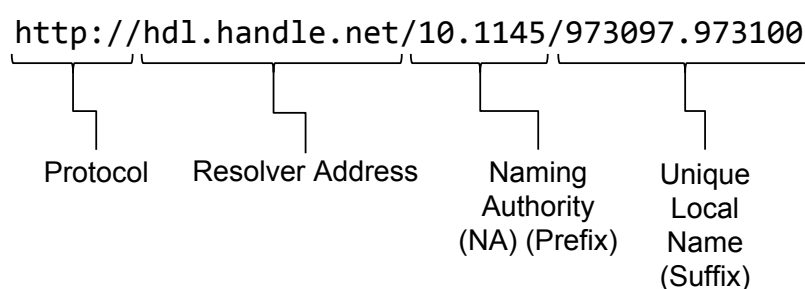
Figure 2.9: Augmented Handle Naming Scheme (based on [44])

2.5 Magnet Links

For bridging the gap between different applications in mobile and desktop environments, Magnet Links are used as transport container for complex access information. The structure and the usage of Magnet Links are described in the Magnet Link URI scheme that currently is a work-in-progress specification [45]. The target behind the work on Magnet Links is to integrate utility programs, such as file downloading programs, into hypertext media, such as websites. In order to be compliant with current standards and systems, the Magnet URI schema follows best practices of the Internet Engineering Task Force (IETF) specifications for Uniform Resource Name (URN). Furthermore, the usage of Magnet Links has also been discussed in the area of the World Wide Web Consortium (W3C) [46]. Magnet Links contain suitable service access information for location-dependent and location-independent data access. Today, Magnet Links are supported by numerous Peer-to-Peer (P2P) applications and are the de-facto standard in large file sharing communities [47]. In Table 2.2, we provide an overview of different data distribution systems that make use of Magnet Links for storing data access information.

System	URN	Value
Gnutella2	`sha1`	file hash (SHA-1)
Gnutella2	`tiger`	file hash (Tiger Tree Hash)
Kazaa	`kzhash`	file hash (proprietary)
BitTorrent	`btih`	unique file identifier

Table 2.2: Magnet URI Scheme Usage for Different Data Distribution Systems

To initiate a download with a Magnet Link from an application like a web browser or a smartphone app, a pseudo-protocol handler for the Magnet Link URL format `magnet:?xt=urn:<System>:<Access Information>` is registered. This protocol handler passes the information from the browsing application to the utility program that is able to process the access information and start a download.

A Magnet URI has the form of

`magnet:?xt=urn:<System>:<Access Information>`,

where `magnet:` is the URI scheme for a Magnet Link and all subsequent keys after the "?" character contain information about the digital object in form of a key-value dictionary concatenated by "&". Keys may contain location-based access URLs or descriptive information that are needed to access a digital object independent from its location. In Table 2.3, we can see an excerpt of the Magnet URI link scheme keys including its names and purposes.

Key	Name	Purpose
`as`	Acceptable Source	location-dependent download URL
`dn`	Display Name	file name
`kt`	Keyword Topic	search key word
`tr`	address Tracker	optional tracker information for BitTorrent
`xt`	exact Topic	location-independent access information in URN-format
`xl`	exact Length	size in bytes

Table 2.3: Magnet URI Schema (based on [45])

2.6 Overlay Networks with BitTorrent

In this section, the foundations of BitTorrent are introduced. BitTorrent is built on the principle of peer-to-peer technology. It is augmenting the location-based networks with a logical overlay network that resides on top of today's network structures. Similar to NDN, BitTorrent follows the principles of location-independent data localization and access

within its overlay network space. It tackles problems of classic networks by providing a decentralized and robust data dissemination design that allows parallel downloads from multiple sources. BitTorrent addresses the challenges of locating data in the network and reducing network congestion, when particular data sets become well-known and the operator is not able to provide as many servers as needed to fulfill all requests by sharing downloaded data between the clients.

2.6.1 General Principles

BitTorrent has been proposed by B. Cohen in 2001 as a P2P network for file sharing [16]. It has been formulated to distribute (large) files collaboratively using a swarm of computer which exchange data with each other in a peer-to-peer principle. To exchange information and data, BitTorrent is constructed as an overlay network that runs on top of TCP and UDP connections in Open Systems Interconnection Model (OSI) layer seven. An essential property of BitTorrent is the file dissemination strategy that uses a mechanism of knocking down files into small chunks (cf. Section 2.6.3). These chunks are exchanged between every BitTorrent peer that is interested in the file. Chunks that are downloaded are instantly shared with other peers, in order to use the upload bandwidth of participating peers to fulfill download requests of other interested peers. By this, BitTorrent pushes the network load from central servers to the decentralized peers. Furthermore, it provides a self-amplifying effect on very popular files, as with the amount of peers interested in the files, the amount of peers offering download capabilities rises [48]. Thus, every client that is downloading a data sets helps to distribute its already downloaded data to other clients. By this, BitTorrents bandwidths is scaling up with the popularity of downloads in the overlay network. As a result, BitTorrent features collaborative data dissemination like NDN and enables robust data dissemination that is capable of compensating the loss of network nodes.

2.6.2 Network Organization

For distributing files, BitTorrent uses dynamic overlay network topologies that is created individually for file distribution. In Figure 2.10, a complete overview is given. Although the physical topology remains identical, the logical topology of the network is determined for every new file, because a unique set of peers is owning specific parts (chunks) of the file. All nodes that are part of such a topology are called *swarm*. During the data distribution, network nodes grouped in a swarm hold fixed roles. In the following, we have a look at different roles of nodes in a recent BitTorrent network:

Clients (cf. Figure 2.10, number ❸) are mostly run by users that are interested in specific files [16]. The download information can be provided via web, e.g., through a torrent file (cf. Figure 2.10, number ❶). Using the access information stored in a torrent file, the clients

can download chunks from other BitTorrent clients for retrieving chunks in order to get a complete file. While downloading, clients share the existing chunks with other clients. The download process is called *leeching*, while the upload process is called *seeding*.

Trackers (cf. Figure 2.10, number ❸) are used to support the swarm in distributing files [16]. They gather and store the information which client holds which chunks and files. Thus, they help to coordinate and improve the chunk transmission between the nodes. This is done by exchanging information between the clients regarding the temporary network topology. By this, clients get information about new peers that entered the swarm and peers that offer a higher bandwidth for the file transfer. Since the introduction of Distributed Hash Table (DHT) in BitTorrent, trackers are not essential for running a BitTorrent network anymore.
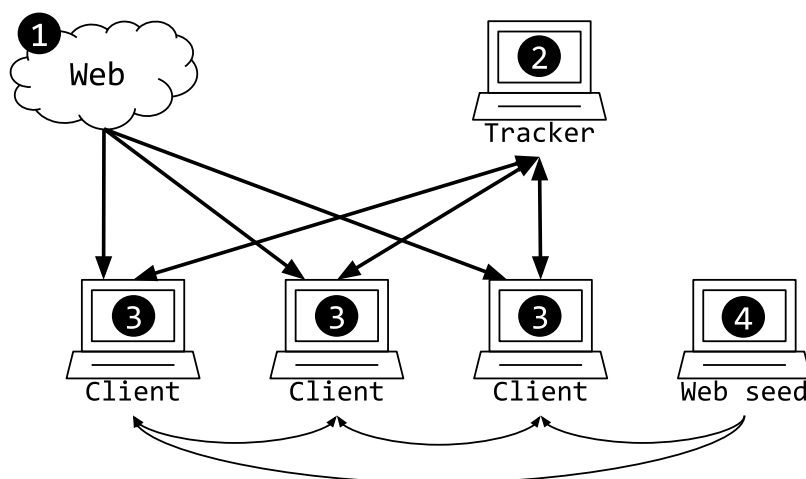
Figure 2.10: Structure of a BitTorrent Network using a Tracker

Distributed Hash Tables are used for locating peers in a decentralized manner. In contrast to tracker-based networks, BitTorrent can also operate *trackerless* using a DHT [49]. In this foundation section, we focus on the discovery of nodes with DHTs but not on the details of DHT operation. When using a DHT, the tracker is replaced by a data structure that is hosted by different BitTorrent clients jointly. The discovery of new peers and the transport optimization is done solely by the clients. There is no central coordination involved and DHT is used for locating new nodes and establishing data transfers between the clients. The official BitTorrent implementation uses a Kademlia DHT [50]. Figure 2.11 illustrates the working principle behind DHT in BitTorrent. The info hash of a torrent file (cf. Subsection 2.6.3) is used as a key in Kademlia. Every node is sending tuples of its network address and the info hashes of the files available for upload to the DHT (cf. Figure 2.11, number ❶). By looking up the info hash in the DHT, a client can retrieve other nodes that are also interested to the same file and may share file chunks (cf. Figure 2.11, number ❷). By connecting to the discovered peer, chunks can be downloaded

(cf. Figure 2.11, number ❸). Furthermore, connected peers can dispatch Remote Procedure Call (RPC) requests on each other, to share a node list with IP addresses from other nodes in order to extend their amount of data sources. This procedure is called Peer Exchange (PEX) and explained in following.
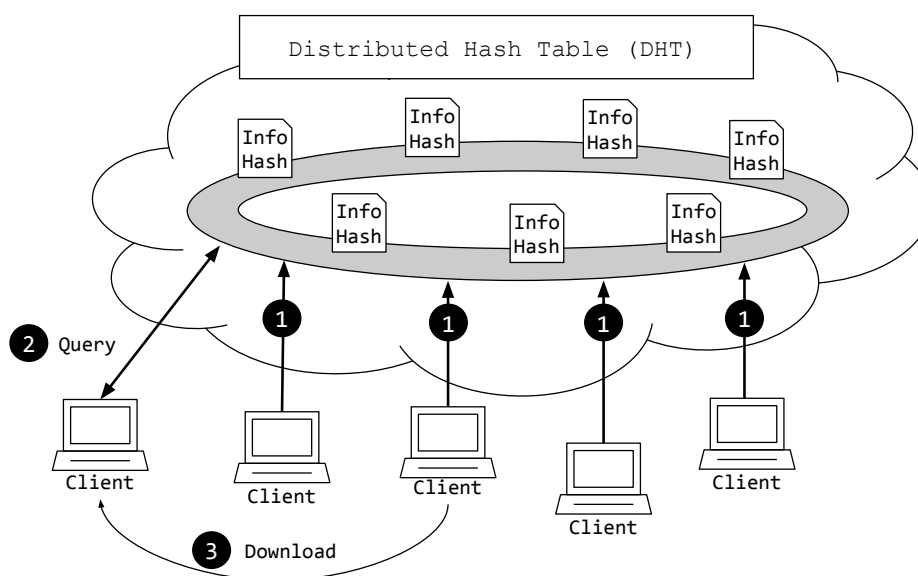
Figure 2.11: Structure of a BitTorrent network using a Distributed Hash Table [51]

Web seeds (cf. Figure 2.10, number ❹) are central web servers that support the swarm by serving files to clients [52]. They offer files using web protocols such as HTTP. Web seeds can operate independently from the network mode and support tracker-based and trackerless file distribution. The concept of web seeds is a contradiction to the principle of decentralized P2P data distribution. However, web servers have usually a better upload bandwidth than regular clients and therefore they are used for bootstrapping file distribution, when not enough clients have received the file to perform a powerful P2P file distribution yet. Another important role of web seeds is to provide file access, when no P2P node shares chunks for a file. This is the case for rarely requested files. As other BitTorrent nodes, multiple web seeds can serve a file simultaneously to speed up data distribution and to provide a failover, if a web seed is offline. Additionally, the concept of web seeds is suitable for supporting other location-based network protocols for data distribution to the swarm [53].

Peer Exchange (PEX) is an additional method used in BitTorrent for discovering peers in swarms. Using PEX, nodes in the swarm exchange lists of peers that share similar info hashes in order to get a more up-to-date view of neighboring peers. By this, additional peers are discovered and the number of queries towards a tracker or a DHT can be reduced. Figure 2.12 is depicting the exchange of peers along the download of data from other peers.

As peer information are not trustful by design, only peers that are actually sharing the file are included in the exchange list [54].

Figure 2.12: Peer-to-peer data distribution and peer exchange in BitTorrent

2.6.3 Data Organization

In this subsection, an introduction into the data organization of BitTorrent is given. We first explain the creation and handling of file chunks. Then, the metadata for BitTorrent access information is described that is necessary to access files. Afterwards, different distribution schemes for access information are given that form the foundation for data distribution used in Chapter 6.

File chunking is used in BitTorrent to segment a file into smaller parts. These parts are exchanged between the peers. File chunking allows peers to exchange parts of files (seeding), before a complete file has been downloaded. By this, a peer can start seeding, when the first chunk is complete and verified. As a result, a client is improving data distribution with the first (random selected) obtained chunks that are redistributed within the swarm immediately. The length of each part is by default 256 kilobyte (2^{18} bytes) by default. For each chunk, a SHA1 checksum is computed that is used to address and verify it. The length of the chunks is stored along a map with the SHA1 hashes of all chunks [48]. To distribute files efficiently BitTorrent uses a *rarest first policy* for chunk distribution [55]. With this policy, the clients request chunks that are least available within the swarm to improve the availability of underrepresented chunks. As a result, downloads are secured against missing chunks and a starvation of downloads is inhibited.

Descriptive metadata is stored together with the access information. It can be included optionally along the file chunk information set and provides space for a file name suggestion or for the total file size of the download [48].

Bootstrap information is necessary to start the data exchange with BitTorrent and is needed when using trackers or DHT. If trackers are chosen as swarm coordination mechanism, a list of trackers is part of the bootstrap information, which is called *tracker announcement*. To start its exchange operations, a client contacts a tracker in order to find other peers and becomes a part of the swarm. This approach has the disadvantage that trackers are a central infrastructure and if no tracker is available, file exchange is impossible due to a missing entry point into the swarm [48].

If the trackerless mode is chosen, bootstrap information with a different content are needed for joining a DHT. Hence, trackerless access information does not contain a tracker announcement and instead a *nodes* key is included for DHT joining. For initial bootstrapping of a BitTorrent software that has not started on the computer before (cold start), nodes are chosen using Kademlia algorithm properties such as the K closest neighbor of the generator's DHT routing table. If the BitTorrent has started before and was able to join a DHT (warm start), a list of DHT nodes has been saved from the previous run and is used to rejoin the DHT. As the tracker announcements, DHT bootstrapping is a critical phase of BitTorrent operation [49]. Modern BitTorrent software can contact an additional DHT bootstrapping server as fallback, to connect to a DHT swarm if the node key is invalid. A list of popular DHTs is available at [56].

Besides bootstrapping information related to swarm activities, access information to web seeds can be included into it as well. By using web seeds, downloads can be invoked directly from web sources. Web seed bootstrap information contains a list of web seeds that offer the file for direct location-based download on URLs [53].

Access information distribution can currently be realized in two different ways in order to access data in BitTorrent. The access informations contain all necessary data to invoke a P2P download within a BitTorrent network.

a) direct distribution
In the original draft of BitTorrent, the information of file chunks, descriptive metadata and bootstrapping information was given in a binary-encoded file. These files have been associated with the extension *.torrent* and can be distributed on the web as file downloads, via E-Mail as attachments or stored on any other digital media. They have a size of a few hundreds bytes for minimal examples and do not exceed one megabyte for even large donwloads [48]. Hence, this direct distribution of access information has some practical limitations, as binary-encoded access information is not suitable to be stored in web links used in websites. For sharing .torrent files, file distribution mechanisms have to be provided for exchanging access information. In the case of web distribution, a HTTP server is used for offering the access information downloads (cf. Figure 2.10, number ❶).

b) indirect distribution

For sharing a file with BitTorrent using indirect distribution, the access information are computed first. Then, the SHA1 hash of the access information is computed, called *info hash*. Afterwards, the access information is stored in the swarm together with the file chunks of the shared file. Now, if a user wants to obtain the file from the BitTorrent network he or she can use the info hash instead of a torrent file. For this, the info hash is used to download the access information of the file from the swarm. With the access information in place, the download of the file chunks from the swarm can be initiated. As we can see, indirect distribution with info hashes uses the swarm for distributing access information and uses the hash to link them from the outside, e.g., through a text link on a website (cf. Section 2.5). Hence, the mechanism of indirect distribution makes a BitTorrent swarm a self-hosting access information and metadata platform [57].

2.7 Information Centric Networks with Named Data Networking

In this section, the foundations of Named Data Networking (NDN) are explained. NDN is the most recent realization of an Information Centric Networks (ICN) and a current target of research in the area of network technology and information organization [58]. The intention behind ICN is to fix the major problems of the current Internet architecture in order to improve the access and dissemination of content in computer networks. In particular, ICN addresses the problem of locating data in the network and the problem of data congestion, also known as *Slash-Dot-Effect*, when particular data sets e.g a website, becomes well-known and the operator is not able to handle all requests any longer. This is done by shifting the network paradigm from the classic *where* approach to the *what* approach, where data demands are sent through the network instead of requesting data from certain locations. As we will see in this thesis, this paradigm shift is very useful, regarding long-term data availability.

2.7.1 Differentiation between CCN and NDN

In the context of ICN, the acronym Content Centric Network (CCN) is often used as a synonym for NDN. Therefore, the usage of NDN in favor of CCN in this thesis has to be explained first. CCN is an architecture project that was initially set up by Jacobson et al. at the Xerox Palo Alto Research Center before the NDN project was started [59]. NDN can be regarded as the successor project of CCN and has taken over many similarities like initial network specifications or early architectural drafts. In October 2015, the NDN project consists of twelve research facilities, universities and the Xerox Palo Alto Research Center (PARC), with V. Jacobson as principal co-investigator [60]. The NDN research project uses the source code base of CCN and is developing it further for the needs of the project and academia [58].

2.7.2 General Principles

NDN are one realization of ICN and make data available independently from its storage location in the network. As in most ICN families, it is not necessary in NDN to express places in the network like IP-addresses or URLs to access hosts and the data stored on it. This is achieved by building a semantic and technological bridge between the content stored in digital entities and its discoverability in the network. These principles classifies NDN as a location-independent data access technology.

But in contrast to overlay networks such as BitTorrent, which operate with distributed hash tables on top of existing IP-driven networks, NDN can replace the network transport layer completely or use it in conjunction with existing transport technology following the ISO/OSI model [61]. Thus, from the point of view of long-term research data access, the network becomes an integral part of the data curation stack. In this role, it is not only a transport layer that shifts data from a server to a requesting client, but it exploits the resources of computers and smart nodes in the network that can support to accomplish the task of data dissemination.

The fundamental principle behind NDN is that data consumers specify *what* kind of data they need, instead of connecting to a specific data source, where the place is known in advance, which would mean to solve the question *where* the data is located. Hence, in NDN there is no host-level notion for data access as it is in traditional networks. Thus, data requests in NDN are issued at the network level. To perform data operations, the network nodes in NDN need more functionality, than network nodes in classic location-based networks.

As most ICN architecture, NDN is designed as a consumer-driven architecture. In this kind of architecture data requests are stated as *interest* that are broadcasted through the network, directed by NDN routing algorithms until a valid data source is matched and data is sent back to the requester [11]. One fundamental aspect of the architecture is the principle that every node in the NDN network can answer to an interest with a matching data packet (interest consumption). As all NDN nodes posses an own cache for data packets they are able to respond to interest immediately without forwarding the interest to other nodes or the data source (cf. Section 2.7.5). Hence, content that is frequently requested by clients is served by the nodes using its caches, instead of the original data source. With this design property, the data distribution in the network is self-amplifying when sudden demand of a specific data set occurs. By this, more frequent data is pushed from the data source "closer" to the data requesters, meaning a significant load reduction for many Internet Applications such as downloads, e.g. for operating system updates or websites suffering under the Slash-Dot effect [11].

2.7.3 Naming Data

NDN data is identified by *Data Names* that are hierarchically structured into a series of components [11]. The components are octets of variable length. The values can be human-readable in order to reflect the usage in certain applications, but they can also be encrypted or reflect any other purpose which allows a large span of use cases. As a delimiter, Jacobson has chosen "/" in order to provide a common delimiter that is similar to the URL scheme specified in [19]. However, it is possible to use other delimiters, as long as they do not conflict with NDN control characters. The NDN project has defined all delimiters in its NDN implementation in a technical IETF draft [62] [63].

A very important property of naming data is that interests can be specified not only on a pre-selected data names, but also on identifiers like file hashes. Consequently, applications can access data with hash-based pointers similar to the SHA1 hashes of BitTorrent. Another important aspect is that data names can be resolved directly in the network without any auxiliary systems such as DNS because the relation between data content and data name can be directly managed by applications running on top of NDN, which can implement a semantic model between data names and content. Hence, the data names can be constructed by design long-lasting and persistent in the case of changing network locations. However, these naming properties do only provide the base for a convenient naming schema, but they do not assure other necessary naming conditions in a NDN network, such as the uniqueness of data names or the organization and partition of NDN names spaces. Thus, NDN data names are a foundation for persistent entity naming in networks, but they are no persistent identifiers, which require additional efforts, as we will see in the latter.

User/App supplied name   **Versioning & Segmentation**

`/gwdg.de/videos/win_setup.avi/_v<timestamp>/_s3`

Globally-routable name   **Organisational name**   **Conventional/automatic name**
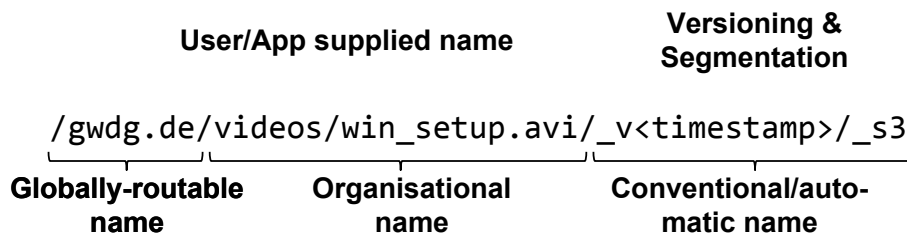
Figure 2.13: NDN Data Name Example (adapted from [11])

2.7.4 Packet Types

There are two types of packets in NDN networks [11]. *Interest packets* are used for expressing requests for specific data sets in the NDN network (cf. Figure 2.14, left site). It consists of three sections: a *content name* ❶ identifies the data set that is subject of the request. A *selector* section ❷ is used for multiple purposes in order to add details to the request for finer data selection. It can e.g. be used to request a specific version of a data

set, as multiple versions of a file can coexist in NDN networks, or to select a specific part (bitstream offset) of a data set. The selectors are stated as version string representation or data segment offsets. A *nonce* section ❸ is appended to the packet to provide it with an identity. For the majority of use cases, a random value is chosen as nonce, in order to make interest packets distinguishable by the source, if the same data name is chosen as selector from different interest packets in the network. Furthermore, the nonce is used to link the interest packet logically to the sending NDN node.

Data Packets (cf. Figure 2.14, right site) are used to transport requested data from a NDN node back to the requesting NDN node, which emitted an interest packet before. Thus, data packets follow the path backwards that interest packets have gone prior through the network. Similar to interest packets, data packets also have a *content name* section ❹ that identifies the packet through the name of the contained data. The content name can be extended with version strings or segment information if a data set is segmented into multiple data packets for better transmission handling. The section *signature* ❺ contains an optional cryptographic signature of the data, which is used to verify the content of a data set. Content verification is necessary, as NDN data can also come from other nodes or caches that are able to fulfill the request instead of the original data sources. The section *signed info* ❻ contains the access information (data names) for obtaining the data for signature verification, e.g. a public key or a certificate. In the section *data* ❼, the payload of the packet is stored.

```
        interest packet                 data packet

  ┌─────────────────────────┐    ┌─────────────────────────┐
  │ ①  Content Name         │    │ ④  Content Name         │
  ├─────────────────────────┤    ├─────────────────────────┤
  │ ②      Selector         │    │ ⑤      Signature        │
  │ (order preference, publisher│ │ (digest algorithm, witness, …)│
  │      filter, scope, …)  │    ├─────────────────────────┤
  ├─────────────────────────┤    │ ⑥  Signed Info          │
  │ ③      Nonce            │    │ (publisher ID, key locator, │
  └─────────────────────────┘    │      stale time, …)     │
                                  ├─────────────────────────┤
                                  │ ⑦      Data             │
                                  └─────────────────────────┘
```
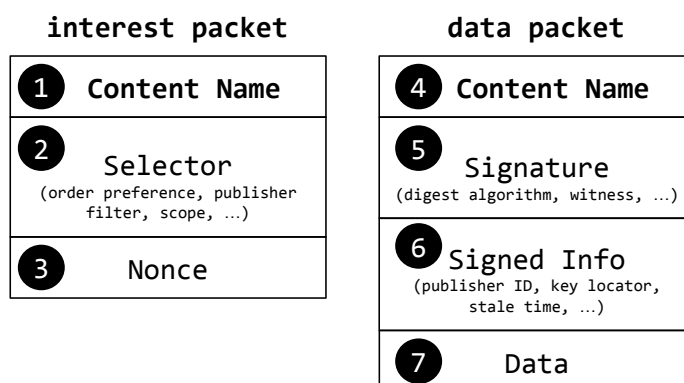
Figure 2.14: NDN packet types (adapted from [11])

2.7.5 Node Design

In order to understand the principle of NDN, it is necessary to have a look at the node design. As shown in Figure 2.15, typical NDN nodes consists of three major parts. The first part is the application part that is running a specific use cases like file sharing. Then, the second part is the *packet forwarding engine* (depicted in gray), which serves as a middleware and is realizing all the NDN specific network functions. It provides abstraction for the network details and allows to connect the application and NDN network part together in one

component. In the lower part is the operating system that is providing access to the network using the hardware of the network node (depicted as black boxes). For explaining the details of NDN, we now focus on the packet forwarding engine (cf. Figure 2.16) and look at the details of the gray-painted box.
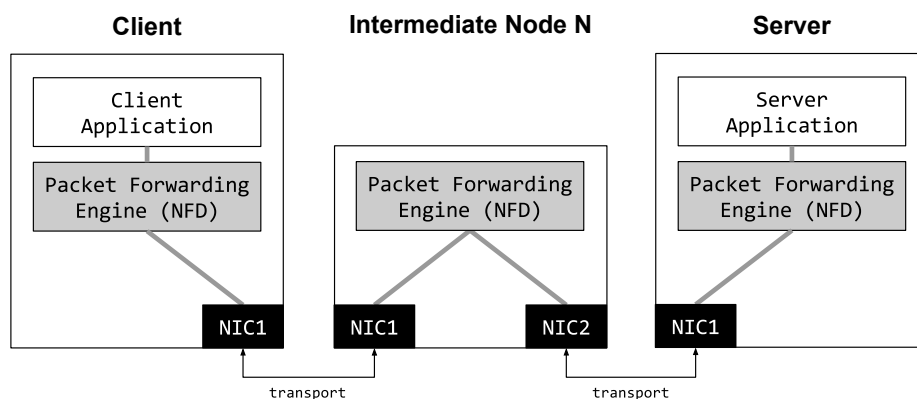
Figure 2.15: Overview of NDN Node Details

The packet forwarding engine is explained next using Figure 2.16. It is also referred as Network Forwarding Daemon (NFD) in the literature according to the name of the software component that implements the middleware. The packet forwarding engine consists of a Content Store (CS), a Pending Interest Table (PIT), a Forward Information Base (FIB) and an engine containing the NFD program logic that is connected to the faces. Faces handle sending and receiving of interests and data packets to other NDN nodes or applications. It is a network abstraction for NDN and can be considered as a mixture between a (hardware) network interface and a network socket available for user applications. The face represents different connection types and provides either a connection to another NDN node, or to an application serving and/or consuming resources over NDN. In NDN networks, faces are usually created for each network device present in the NDN node and each application connected to the NFD. As the connected NDN nodes and applications exposes different data namespaces, the availability of namespaces on the different faces is stored in the FIB ❸. By using the information in the FIB, the engine can determine on which face incoming requests and interests have to be forwarded. Hence, the FIB serves as storage for the routing information within the NDN node. Routing information are obtained through routing announcements of NDN names from applications or other NDN nodes (cf. Section 2.7.6).

As stated before, NDN nodes are able to respond to data requests autonomously, which is one secret behind NDN. For this, the NDN node has a data packet cache called Content Store ❶. Unless not suppressed by the data packet sender, the CS stores a copy of each received data packet by its name for a limited time span. If an interest is received that matches a name of in the CS, the interest is *consumed* and a data packet is emitted with content from the CS to the requester. The re-usage of CS content is only possible, if NDN data packets are self-contained and idempotent, meaning that they contain all necessary

information for application usage in the same packet. For this, applications have to build the data packets self-describing in order to support the idempotent nature of NDN [11]. The usage of the CS allows saving bandwidth dramatically, when the same data sets are frequently demanded. This allows shortening reaction times and latency and to push data simultaneously over different connections. As a result, NDN is able to build fast, robust and resource efficient network structures.

The Pending Interest Table (PIT) (cf. Figure 2.16, ❷) is the data structure that stores for all incoming interests the data name and the face, where the interest has been received. If a data packet with a name enters the NDN node and there is a match between the data names in the PIT, the data packet is forwarded to the face recorded in the PIT entry. After the forwarding of the data packet, the matching PIT entry is deleted (consumed). If no match for incoming data packets occurs within a certain time limit, the PIT entry is removed (time out). As we can see, the PIT controls the forwarding of data packets along the same faces that an incoming interest has taken through the node. By this, data packets flowing from the data source to the requester take the same route through the NDN network as the PIT-recorded interest. In contrast to the interest flowing from the requester to the data source, the data packets are forwarded using the PIT entries in the opposite direction to deliver the response back to the requester. Hence, the PIT entries stored in the intermediate NDN nodes between the requester and the data source serve as line of hints, which the data packets follow.

The Forwarding Information Base (FIB) (cf. Figure 2.16, ❸) is the data structure within the packet forwarding engine that is storing all data for interest routing. The FIB stores a combination of the NDN prefix (cf. Section 2.7.3) and the faces that are connected to other NDN nodes or applications serving the data using the prefix namespace. In order to perform name-based routing of interests to a specific face, the engine compares the interest name prefix against all FIB entries. If an entry matches based on the longest identical NDN prefix in the table, the interest is forwarded to faces stored in the entry. If no entry matches, the interest is discarded and the NDN request ends at this NDN node. Entries in the FIB can be created by prefix routing announcements, which we describe in the next section.

As we can see now, there is a parallelism between the PIT and the FIB. The PIT is responsible for data packet routing by recording faces and data names of incoming interests. The FIB responsible for interest forwarding using external routing announcements as base for an interest forwarding decisions. The difference in the creation of entries is that the PIT entries are created automatically through interest recording, while the FIB entries need external routing announcements for its creation.

Content Store (CS)		
①		
Name	**Data**	
/gwdg.de/videos/ win_setup.avi/v1/s0	1010101...	

Pending Interest Table (PIT)		
②		
Name	**Requesting Faces**	
/gwdg.de/videos/ win_setup.avi/v1/s1	0	

Forwarding Information Base (FIB)		
③		
Prefix	**Face List**	
/gwdg.de	0, 1	

Engine

Face 0 — WLAN card (LAN access)

Face 1 — DSL modem (WAN access)

Face 2 — Application (file sharing)
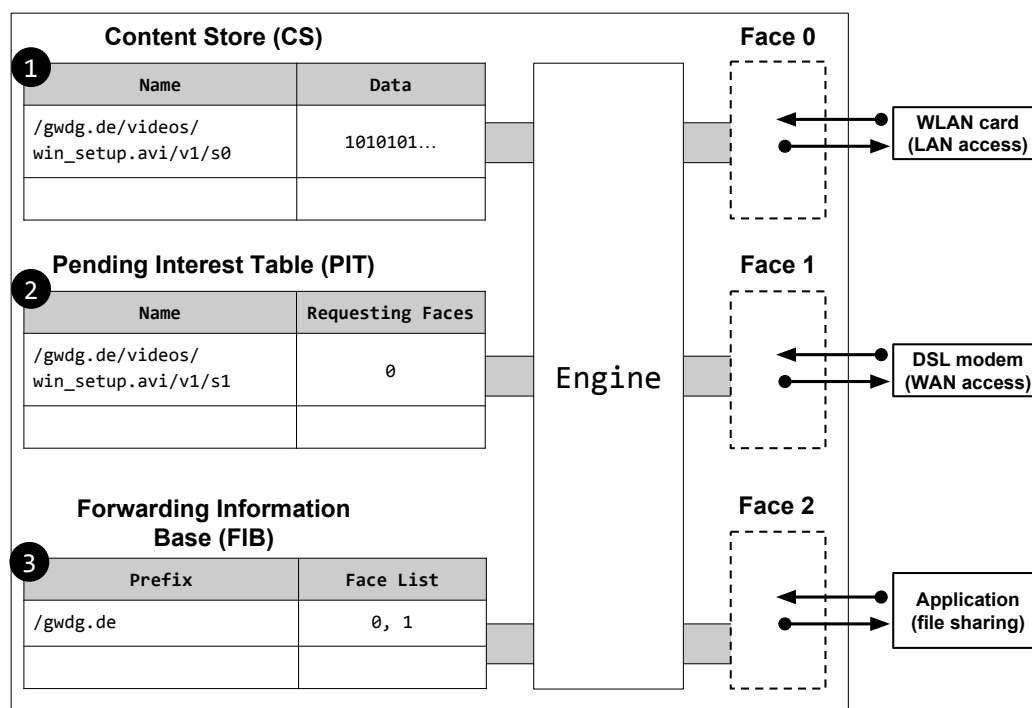
Figure 2.16: NDN Packet Forwarding Engine (adapted from [11])

2.7.6 Routing

In general, routing schemes that apply to IP-based networks can be applied to NDN networks, too. But in contrast to IP networks, the subject of routing in NDN networks are not single network packets but rather interests and data packets. NDN routing is used as a layer for logical organization of NDN interest and packet flows. When NDN is operated in a native environment, the NDN routing directly influences network behavior. If NDN is operated as an overlay network on-top of TCP/IP, the NDN routing is independent from the routing of IP packets that are used for data transport in lower network layers. More details on transport and flow control are described in Subsection 2.7.7.

Let us now have a look at NDN routing as a fundamental principle in NDN. There are several semantic similarities between the creation of routing decisions in IP-packets and NDN entities. But in contrast to IP networks, the routing of NDN packets has fewer restrictions in a direct comparison to IP, because NDN allows more network topologies that may include loops [11]. Hence, there are no restrictions to configure routes using loop-based multi-sourcing and/or multi-destination entity routing. Furthermore, the route look up is based on the longest prefix matching algorithm.

When looking at the routing procedure of NDN in detail, it can be split into two major phases. The first phase is a bootstrapping phase, called *pre-topology phase*, which is responsible for initializing the NDN connectivity of a node. In this phase, the NDN node has no connection to an existing NDN node and no network peers are known. The target of this phase is to detect other neighbor nodes and to verify their identity through cryptographic mechanisms. This identity verification is an optional step. As bootstrapping of NDN nodes is an own topic of research, we recommend the paper of Mahadevan et al., who investigate the problems of bootstrapping CCN/NDN networks using a list-based local resolution bootstrap approach with trusted initialization peers, similar to the DNS [64].

After a connection to peer nodes has been established in the pre-topology phase, the inter- and intra-domain routing is entered in the second phase. In the next step, after the node has acquired information on routing, it is able to participate in the network by sending and receiving NDN interests and packets. Intra-domain routing addresses the issue how nodes *discover and describe* their local connectivity, what resources are present in their environment and what structure the network graph around the node has [11]. Inter-domain routing addresses the problem of reducing the peering costs in NDN networks, as in large NDN installations nodes joining and discovering the network will produce significant load in the network. In order to avoid this, inter-domain routing uses the mechanism of peer-announcements to create a more efficient entering procedure for new NDN nodes to the network [11]. We will not describe NDN routing in detail, as this is an own research topic in ICN and particularly in the CCN and NDN community. Further insights into routing challenges are provided in Sun et al. [65].

In order to illustrate the routing process, a simple scenario is provided in Figure 2.17, where an application (depicted as a computer icon in the bottom left lower corner) is requesting data from a source application (depicted as a cylinder icon in the upper middle part). The different NDN networks are connected to each other. When the source application sends an interest for the file `paper.pdf`, the data name stated in the interest packet is analyzed by all intermediate NDN nodes. Based on the longest matching prefix (parts of the NDN name separated by /), the interest packets are sent to different segments of the network (depicted by clouds) that contain the data sources. If the data name stated in the interest is met, a data packet is sent back along the chain of nodes that were forwarding the interest before. The intermediate nodes store the information which interest packet has passed them in the PIT. If multiple sources are able to fulfill the data request data, packets are emitted by multiple sources, making NDN a true multi-sourcing network that is independent of data locations.
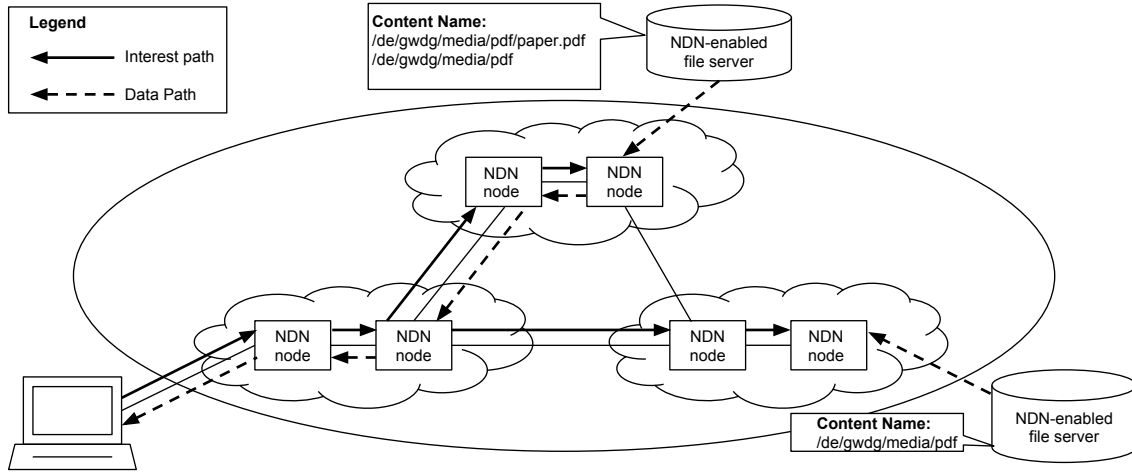
Figure 2.17: Simple NDN/CCN Routing (adapted from [65])

For advanced routing in NDN, routing algorithms developed for location-based networks can be adapted. Hence, NDN network segments can adjust their routings in case of NDN failures and detect alternative routes and namespace access to route interest and data packets. One adapted set of routing algorithms usable for self-diagnosting and self-adjusting NDN networks is Named Data Link State Routing Protocol (NLSR). NLSR is a link-state routing protocol for NDN [66] [67]. It ensures that the NDN name prefixes are propagated in a NDN network in order to ensure the reachability of NDN nodes serving data with a specific prefix. For this, NLSR is using ranked metrics that indicate the reachability of NDN sub-namespaces through different faces.
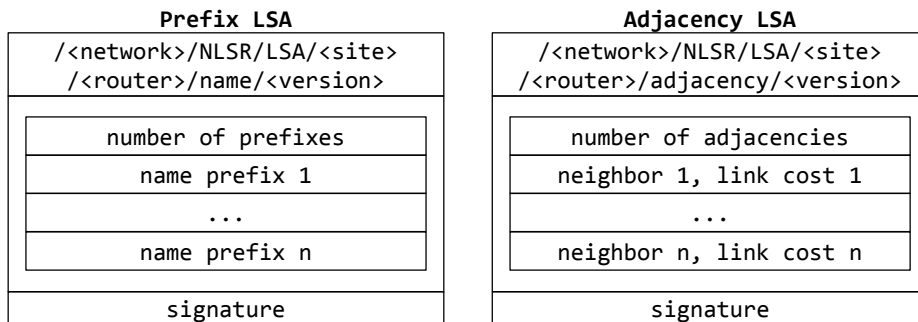
Figure 2.18: Link State Advertisement Packet Format [67]

This information, called Link State Advertisement (LSA) is shared between neighbor NDN nodes, in order to initialize and adjust routing of interests in the NDN nodes. It reflects the availability of NDN names on nodes as *Prefix LSA* and their network connectivity as *Adjacency LSA* (cf. Figure 2.18). For information exchange, *ChronoSync* is used as protocol

for data synchronization between NDN nodes [68]. In this way, the routing is synchronized from one NDN node to the other by sharing the availability information in a decentralized manner. This solves the problem of adding preconfigured network routes into NDN nodes for initial node bootstrapping or if the availability of a name prefix changes, due to (sudden) network restructuring [67]. Hence, if a name prefix changes its availability, the new routing information is shared with NDN network nodes using NLSR. Data sources and links that add new namespaces to a NDN network are propagated through NLSR. Thus, NLSR is not only suitable as an advanced mechanism for bootstrapping a NDN network, but it is very important to make NDN network robust to link and node fails and ready for instant upgrades of data sources and network links. As a result, NLSR-enabled NDN networks can provide a better resilience to network failures than IP-networks, if the network application using NDN is able to outplay these advantages through its design and operation. For this, NLSR supports the exchange of multi-path forwarding, as NDN in contrast to IP-networks does not require cycle free network topologies like spanning trees for a correct operation [67].

2.7.7 Data Transport and Flow Control

In general, NDN is not bound to network components that support natively NDN, as this step would require new network hardware. With the introduction of Software Defined Networking (SDN) it is possible to build native devices for NDN using SDN software. By this, the packet forwarding engine (cf. Section 2.7.5) could be integrated as an integral part of a SDN device into the network. Until SDN gains a significant market share, we can assume that the network technology remains unchanged for the next years. Hence, we can assume that NDN in the current test beds run on top of a classic network as an overlay network. For these overlay networks, we can assume an OSI network model [69], where layer four provides the transport function with TCP and UDP as protocols and layer three is using IP for relaying and routing datagrams [70] [71]. When operating NDN on top of a packet-based delivery media, interests that are not satisfied within a certain time frame are re-transmitted by the node using all valid routings stored in the FIB until the time out threshold is reached. This ensures that interest packets reach nodes over newly established network paths that have been created in the meantime due to dynamic changes in the network topology [11]. This design makes NDN a perfect solution for unreliable networks and frequently changing network connectivity [72]. Of course, this behavior and the fact, which has been mentioned before that NDN networks do not rely on spanning tree topologies lead to the result that packets are transmitted several times and that interest packets are even routed in a limited circular manner. A countermeasure to avoid network saturation due to circular routing and multiple transmissions is to add a *random nonce* to every interest packet in order to distinguish interests from different sources. Thus, duplicated or miss-routed interests are discarded by the node engine. Additionally, *sequence numbers* are added to NDN packets similar to TCP Acknowledgement (ACK) packets [11]. NDN has no need for a dynamic flow control as it is used in TCP that is based on an end-to-end principle

and uses dynamic window sizes to control the traffic and avoid congestions in the data transmission [73]. In contrast, NDN uses a hop-to-hop transmission principle between the nodes that only makes a flow control from one node to the other necessary. However, when running NDN on top of a TCP connections, dynamic TCP flow control occurs between the nodes connected via TCP in the lower transport layer. While writing this thesis (2016), data transport and flow control are active topics of NDN research. Hence, this paragraph only introduced the existence of a hop-to-hop flow control in NDN as a fundamental network principle and therefore does not include cutting-edge algorithms that are outside the scope of this thesis.

2.7.8 Content Validation and Content Protection

In general, NDN provides a separate field in the data packet that allows the integration of a cryptographic signature (see Figure 2.14). There, signature data can be included to implement different payload security mechanisms. This is necessary, as every node in NDN networks can answer to interests. Thus, in a location-independent network the security model has to shift from trusted locations to trusted content. Now, we have a closer look at the fundamentals of content validation and content protection.

Integrity checks verify that the message has not been altered in the meantime, e.g. through a transmission error or by malicious manipulation. Integrity checks can be done in a very simple way by generating a checksum over the payload or by applying a more sophisticated mechanism that relies on symmetric (cf. Section 2.8.4) or asymmetric encryption (cf. Section 2.8.3). *Sender identity verification* can be implemented either by obtaining the certificate with the public key from the node of origin using the built-in Public Key Infrastructure (PKI) of NDN or by using web of trust approaches. To realize these protection mechanisms, NDN implements a system of cryptographic keys that are stored by every node in a NDN network [11]. The NDN key system operates here in asymmetric manner, which assigns a pair of keys to every network node – a secret private key and an open public key (cf. Section 2.8.3). Obtaining public key from the NDN network is easy, as public keys can be accessed like any other data set using a data name. The PKI has to be in place to verify the authenticity of the public key using a certificate with the identity of the data source owner. For access the public key or the certificate, a data name is needed that should be inferable easily for end-users. For instance, a fixed name scheme for obtaining the public key from a node could be `<nodeId>/keys/root.pub`. It is open to the NDN implementation to choose an asymmetric encryption scheme such as Rivest, Shamir and Adleman (RSA) and to decide on a scheme for determining the data name for obtaining verification mechanisms from a node. This integrated and smart usage of asymmetric encryption and public key distribution allows securing data transmission and can additionally be used as an initial mechanism for bootstrapping advanced transport and authentication mechanisms [74]. Furthermore, routing announcements can be secured against malicious manipulation using the built-in verification mechanisms of NDN (cf. Section 2.7.6) [64].

2.8 Cryptography

As communication in overlay networks and information centric networks depends on cryptographic functions, we introduce the necessary foundations. Besides the domain of network communication, several PID systems employ cryptography to secure their communication, to authenticate user and to protect PIDs through cryptographic means. In the following, we describe the general principles of symmetric and asymmetric encryption. After that, we introduce the foundations of certificate management, hash functions and cryptographic signatures.

2.8.1 Symmetric Encryption

Symmetric encryption uses a shared secret key K for encrypting a plain text P into an encrypted text C. For both, encrypting and decrypting, the secret key K is required. Hence, the symmetry is given by the usage of K on both sides [75]. As a matter of fact, communication that is secured by symmetric encryption needs knowledge of K on the side of the receiver and the sender. K needs to be exchanged on a *secure* channel without observation of a malicious third party. If K is disclosed, C can be translated back into P by a malicious attacker [75].

Symmetric encryption can be operated in two ways: *block cipher* and *stream cipher*. *Stream cipher* XORs every chunk or bit of P with a respective chunk of the pseudo-random cipher stream that has been derived out of K. As the principle of a stream cipher works bit wise, it does not require a minimal set of bits to start its operation but is able to start immediately. *Block cipher* in contrast operates blockwise on P and handles a block of n chunks or bits at a time using K. Block ciphers requires a minimum size to start, which is at least the length of a block. It requires P to be extended in length that the length of P' is a multiple number of the block length. This extension is called *padding* [75].
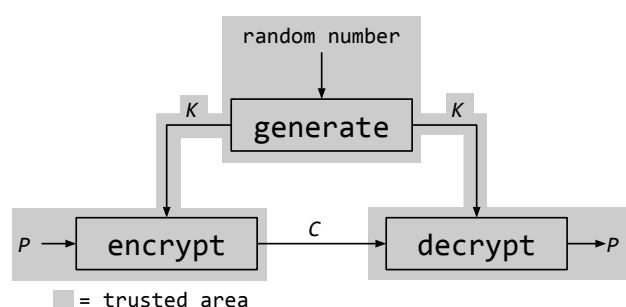

Figure 2.19: Principle of Symmetric Encryption (adapted from [76])

Figure 2.19 illustrates the key distribution process in a trusted area for symmetric encryption. Following the figure, we can see that the encryption and decryption functions have to be executed in the trusted area, too. The usage of symmetric cryptography poses the

challenge of exchanging the cryptographic key over an insecure communication channel. To solve this problem, the Diffie-Hellman key exchange has been invented by W. Diffie, M. Hellman and R. Merkle in 1976 [77]. This key exchange algorithm allows two parties to compute an identical key by sending information over a public unencrypted channel. To compute a secure key on both sides, the communication must be protected against changes or fabrication using a Message Authentication Code (MAC) or cryptographic signature in order prevent a man-in-the middle attacks. If this is assured, the attacker is not able to infer the key by observing the information on the public unencrypted channel.

Two well-known symmetric ciphers that are known to be secure at the time of writing (2016) are Twofish [78] and Rijndael, commonly referenced as Advanced Encryption Standard (AES) [79]. Both are designed to use XOR operations to perform its encryption and decryption. They have a small memory foot print in order to be implemented on hardware with low computation power.

2.8.2 Asymmetric Encryption

Asymmetric encryption has been developed to avoid the problem of secure key exchange between the communication parties. It employs two different keys, the encryption key K_E and decryption key K_D. While K_E is available for everyone publicly and can be transmitted through an *insecure* channel, K_D has to remain secret (private). To achieve these properties K_E and K_D are designed with certain special mathematical properties. Therefore, this technique is called public-key-encryption [75]. The most important realization are RSA using the hardness of prime number factorization to protect secret information [80] and Elliptic Curve Cryptography (ECC) as more advantage asymmetric encryption scheme using the hardness of dividing points on a cyclic group of an elliptic curve for information protection [81] [82].
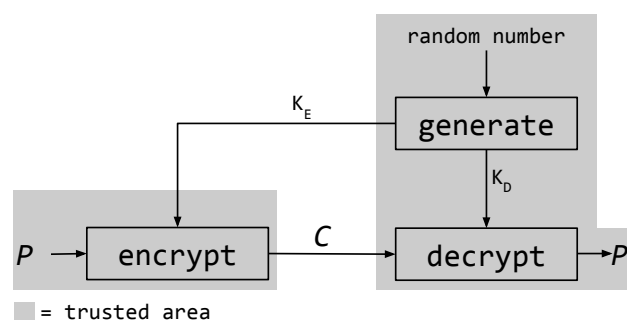
Figure 2.20: Principle of Asymmetric Encryption (adapted from [76])

The cypher text C is generated from the plain text P by using the public K_E, which does not need special protection. To decrypt C into the plain text P, the secret key K_D is needed, which has to remain protected to ensure confidentiality of P. Figure 2.20 depicts the principles of asymmetric encryption. The key generation, encryption and decryption has

to take place in a trusted and secure environment. Asymmetric encryption requires strong random numbers to generate the key pair as well as for the encryption process. The secret key K_D cannot be derived from K_E or C. As two different keys are involved, the encryption scheme is called *asymmetric*. In contrast to symmetric encryption, asymmetric encryption is comparatively slow concerning the data throughput [76]. Hence, asymmetric encryption can be used to exchange an encrypted key for symmetric encryption. After that, symmetric encryption is used to perform the heavy data load. By combining the encryption principles, the throughput for data encryption and decryption can be increased significantly [75].

2.8.3 Digital Signatures

Digital signatures are based on the principles of asymmetric encryption and have two fields of usage. In the first field, they are employed for securing information against fabrication and (malicious) changes. In the second field they can be used to proof that a party is owning a particular secret key [75]. This allows the setup of a non-repudiation schemes with digital signatures attributing signed data sets to particular parties owning a secret key. By this, an independent third party can attribute signed data to the key owner with public available knowledge and without having information on the secret key. Additionally, it allows implementing authentication mechanisms by exchanging signed data in a challenge-response protocol. From Figure 2.21 we can see that the key generation and the calculation of the message signature have to be performed in a trusted area. For generating a signature, a cryptographic hash is generated over the plain text message m. Then, the hash is encrypted with the private secret key K_D to generate the cipher text, which is the signature s from the data. The message is sent together with the signature s to the receiver. The receiver uses the public encryption key K_E to decrypt the message and check the obtained hash against an own hash that has been calculated independently using the message m. If the hashes match, the signature is valid. Leveraging the same mechanisms like asymmetric encryption, the key generation requires strong random numbers, too.
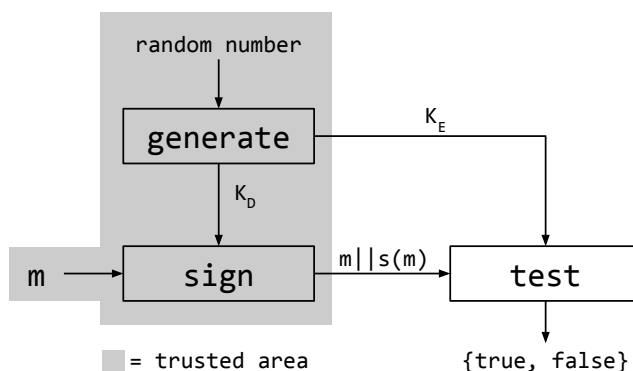
Figure 2.21: RSA Digital Signature System (adapted from [75])

2.8.4 Symmetric Authentication

In contrast to digital signatures, a Message Authentication Code (MAC) *a* can be computed using symmetric encryption. It can also confirm the integrity of a message but does not feature the non-repudiation, because every party owning the secrete key *K* can create valid signatures. The relation between the private and public key in asymmetric encryption allow a key attribution due to the mathematical relation of the keys. As systems based on symmetric encryption have only one key, this attribution is not possible. For message authentication codes Hash-based Message Authentication Code (HMAC) is a common choice [75].
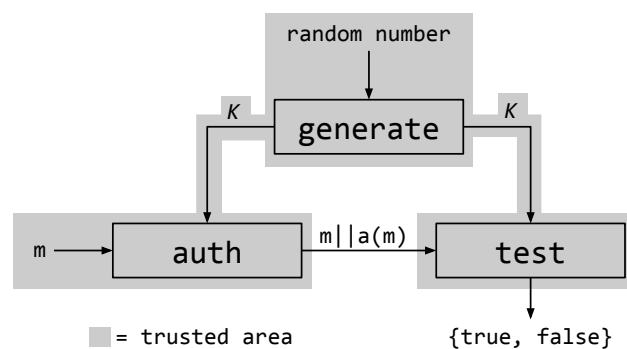
Figure 2.22: Symmetric Message Authentication (adapted from [83])

Figure 2.22 depicts the process. Similar to symmetric encryption, the secret key *K* needs to be generated and distributed in a trusted environment. In the first step, a cryptographic hash of the message *m* is computed. This hash is then encrypted using *K* generating *a*. Then, in a second step, *m* is sent jointly with *a* to the receiver. The receiver calculates the hash of *m* and using *K* for decryption. If the hashes match, the MAC is valid and the message has not been changed. This procedure confirms that the message has been sent by a party who owns a copy of *K* but it cannot assure the identity of the sender. Similar to other symmetric encryption protocols, symmetric encryption has the challenge of a key distribution over a secured trusted channel.

Chapter 3

Problem Statements

In this chapter, we address the major problem fields of PID that occur in the usage of location-dependent network paradigms. This motivates the contributions of this thesis, which aim on improving persistent identifier systems by using location-independent network technologies. The problem statements show the limitations of the current state-of-the-art approaches and techniques, which are also reflected by the related work in Chapter 4.

Today's Internet network technology is based on location-dependent services and end-users consume the Internet using Hypertext-based web media, also known as websites with links to other Internet resources. Researchers use the Internet with its web services and HTTP resources to present, exchange and disseminate research data based on URLs. As URLs point to network locations of web resources, they can break for several reasons and point to non-existing network locations. This can be cause by moving web resources from one Internet server to another, by restructuring of networks, new ownership of DNS domains or a simple renaming of web resources. Hence, broken URLs lead to a loss of information on the Internet and make researchers and librarians to question the usefulness of publishing research result on the Internet without a reliable publishing mechanism [2]. But in fact the problem of broken URLs is inherited from the underlying location-dependent network technology and thus persistent identifiers have been proposed as reliable identifiers replacing unreliable URLs. In this context, PIDs are used to solve the question of *"where is my data located?"* in location-dependent network environments in order to overcome changing network locations by a fixed indentifier, which is adjusted to the currently valid network location (URL) of the web resource [84]. The principle behind PID has lead to different PID infrastructure systems, such as DOI, that is based on the Handle system [85] and Persistent URL (PURL) as other broadly used system [86]. These systems are used in digital libraries, data repositories and literature databases, e.g., IEEE Xplore [87], to abstract the identification from the current valid network location. As the concept of PIDs for globally unique identifiers exists for a long time of over 20 years, billions of data sets have been tagged in the different PID infrastructures. Hence, when improving research data

access and dissemination with location-independent network technologies, the principles and infrastructures behind PID have to be taken into consideration on different levels, in order to keep access to all existing PID-tagged data sets.

For understanding the problems of location-dependent PID resolution and data access, we use Figure 3.1. It depicts the current as-is situation of PID systems. To resolve a PID in order to get access to a (research) data set, the end-user starts in the location-dependent space, as his or her network connections depend on this paradigm and the typical tool for browsing data sets on the Internet is a web browser. The PID itself is located in location-independent space, as it only identifies an entity similar to International Standard Book Number (ISBN) for books. The PID may contain information for the current location of an entity. However, the optional location information stored inside a PID has no impact on its location-independent nature, as it is only one attribute of many others. This means that a URN can serve as a persistent identifier, too. As it is also suitable for tagging real world-objects like serial titles [88] using an International Standard Serial Number (ISSN) or books using bibliographic descriptions [89]. Interesting is now the overlapping of the location-dependent space of the user and the location-independent space of the PIDs. Within this overlap, the PID infrastructure is located with its services. It allows to access PIDs under a location-dependent service using an URL. In the case of PID resolution, it retrieves a location-dependent data pointer (target URL) for the user, in order to provide access to web resources located in the location-dependent space.
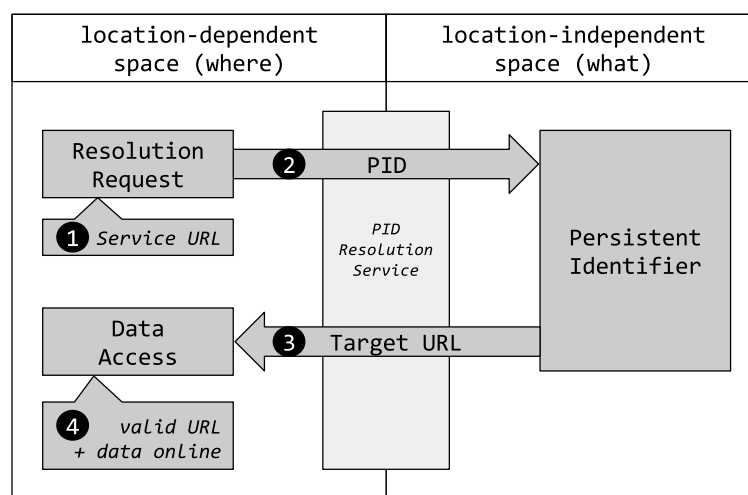
Figure 3.1: As-Is Situation in the Persistent Identifier Domain

When we continue to look at Figure 3.1, the data access using a PID can be described as follows. First, the user has to know the URL of a location-dependent PID resolution service capable of resolving its PID (step ❶). For different PID systems several web-based resolution services exist. The Handle system, offers HTTP-based PID resolution service [36] [39]. The German National Library Service is operating another HTTP-based

URN PID resolution service under the name space `urn:nbn:de` [89]. Then, in step ❷ the, *PID Resolution Service* resolves the PID into its target URL. After obtaining the target URL in step ❸, the user can access the data, if three prerequisites are valid (step ❹):

1. The **Service URL** has to be known by the user and central PID resolution service has to reachable under this URL.
2. The **target URL** has to point to a valid data location.
3. The data set has to be available in the network (**online**).

With the current setup of PID systems available from the location-dependent Internet we can derive the first problem statement:

Problem Statement 1: Breaking the central resolution service of a PID system breaks the PID system for almost all end-users world-wide

All PID systems that employ the principle above use a central (HTTP) resolution service service with a URL using a DNS domain. If the control over a part of the service URL is lost, e.g., the domain ownership changes (unintentionally), almost all PID resolutions request by end-users will fail. The same is true, if an attacker shuts down or takes over the majority of the central resolution services. Moreover, all automated PID resolutions over HTTP fail, as the service URLs are hard-coded in software and scripts. Hence, we can think of following scenarios:

1. If an attacker wants to shutdown the DOI system for end-users world-wide, he or she has to take control over the domain (dx.)doi.org.
2. If an attacker wants to shutdown the Handle system for end-users world-wide, he or she had to take control over the domain (hdl.)handle.net
3. Alternatively, an attacker can perform a distributed denial of service attack against the server located at the domains listed above to inhibit global end-user PID resolution for a limited timespan.

Although the PID infrastructure would continue to function, the system would appear broken to end-users, as they use HTTP-based services for resolution. This reveals the problems that although the PID systems were designed as distributed systems the centralization of the resolution services has introduced a needle eye at the overlapping of both spaces. Moreover, world-wide end-users PID resolutions can be observed at these servers centrally.

Figure 3.2 is depicting the current as-is setup of the Handle system for HTTP-based PID resolutions using the Handle HTTP-proxies located at the service URL (DNS domain) `hdl.handle.net`. Five HTTP-proxies are responsible for the PID resolutions (state 10/15/2016), forming a sensitive needle eye for outage and attacks. The DOI system is using a similar structure.
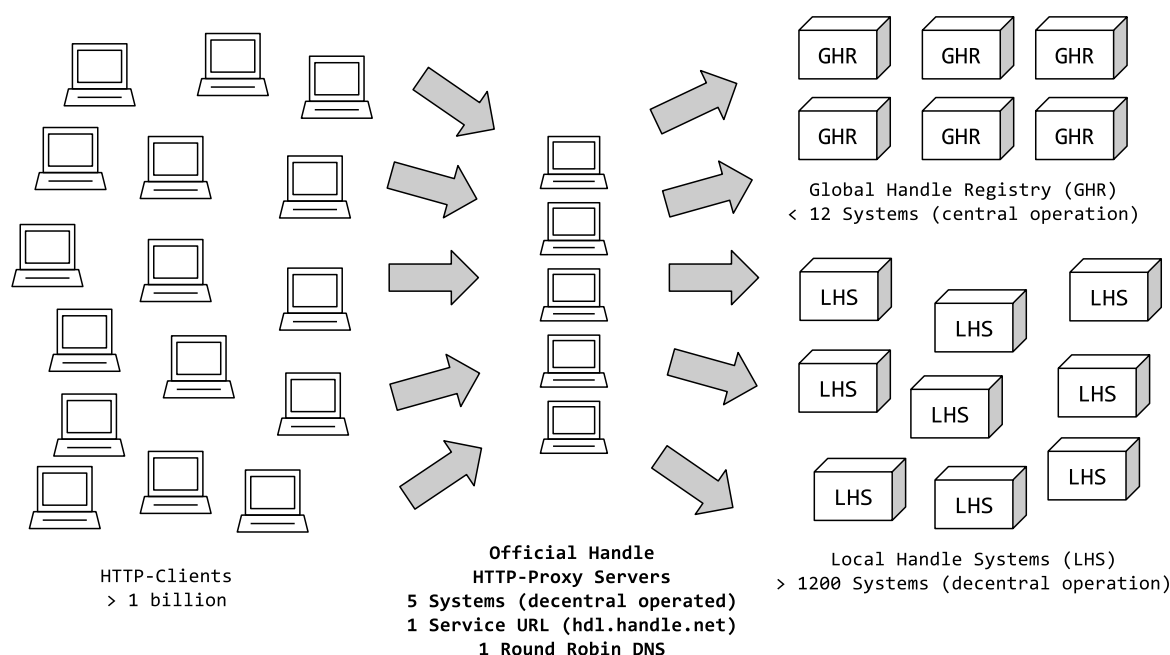
Figure 3.2: As-Is Situation of the official HTTP-based PID Resolution at the Handle System (10/2016)

The second problem statement originates from the challenge of determining network locations for Handle servers storing and resolving PIDs for a specific Handle prefix. The Handle system uses a two-staged approach for accessing PIDs (cf. Figure 2.6). First, the GHR is contacted to request the current network location of all LHS sites using pre-configured GHR network locations. Then, the LHS is accessed for a direct PID interaction. This legacy approach of bootstrapping PID-related communication generates our second problem statement:

Problem Statement 2: The hierarchical structure of the Handle PID system results in unbalanced conditions
The imbalance of the Handle PID system results from the fact that although the Handle system is designed as a distributed system, the entry points for communicating with unknown LHS are operated using a small number of centralized GHR servers. Thus, the central role of the GHR systems introduces following problems:

1. If the small number of GHR servers is not reachable (e.g. due to a Denial of Service attack [90]), resolving of PIDs is impossible on the Internet. While the small number of GHR servers is generating problems on the availability, the centralization has another impact.
2. Every LHS that should be part of the official Handle system, must be approved by the central GHR operators. For this, the LHS network address is added to the GHR and the Handle prefix information served by the LHS is signed by the GHR

operators. Unwanted Handle system operators can be shutdown centrally, by removing the LHS server addresses from the GHR. Hence, large PID collections from unwanted entities (universities, countries, companies, etc.) can be detached from global PID resolution instantly, rendering their PID collections useless for PID resolution on the Internet. In contrast to regulations for DNS entries on the Internet by Internet Corporation for Assigned Names and Numbers (ICANN), there is no possibility to dispute LHS detachment.

Problem Statement 1 and *Problem Statement 2* are addressed in Chapter 5, where we provide approaches for shifting the location-dependent PID infrastructures into the domain of location-independent networks.

The third problem statements originates from the usage of target URLs in PIDs. Data owners and users can check for broken PIDs that do not resolve to a valid URL target by resolving every PID. By this, they can estimate if the resolution works for all given PIDs. The target URL can be evaluated by using the HTTP response status codes that are provided by web servers or data repository software [91]. A complete safety on PID resolution can only be achieved when checking *all* PIDs regularly by a data owner. This is a large effort for data owners as data crawlers that iterate over all PIDs have to be implemented first. As data owners use individual software stacks for data repositories, the reuse of existing code for PID checking crawlers is limited. Furthermore, the decision on data fitness is bound to individual algorithms that judge the availability of data, tailored to the data repository stack. For example, a check against the HTTP status code may allow judging if data is available on a web server when the PID is directly linked to the data set. But when the PID links to a landing page, the HTTP status code of the landing page does not determine, whether associated research data is available, too. Hence, an inspection of the links stored on the landing page is needed as well, to have a complete evaluation of research data availability and for this, detailed knowledge of data repository software is needed. As a result, we see that an evaluation of research data availability is complex and time consuming for every data set that is linked by a PID and is served by location-dependent network technology. Hence, we can formulate the second problem statement:

Problem Statement 3: PID Target URLs have to be correct for usable PIDs

In Figure 3.1, two prerequisites exist after a successful PID resolution for accessing the data:

1. The data has to be online available through a network software like a web server or a digital repository.
2. The target URL has to reflect the correct network location of the data set.

To ensure a correct PID resolution, the PID owner have to maintain the PIDs regularly by checking and updating the target URLs. As we will see in Section 6.2, PID maintenance is a large effort that that is already growing dramatically. Furthermore, defect PIDs with broken target URL can be detected by users and the global PID

infrastructure providers. However, only the PID owners can repair the target URL, as they know where the current network location of the data is.

Now, we come to the last problem statement. This fourth problem statement is originated to the limited usage of PIDs as *simple* pointers to changing URLs.

Problem Statement 4: PIDs must be part of trusted data dissemination
The mechanism of using PIDs as pointers, does not assure that the PID is pointing to the correct data set. In the Handle system, the PID can be signed cryptographically, to assure the validity of the pointer but if the resource is replaced, where the PID is pointing to, then unwanted data sets could be served. Hence, information on verifying data sets that are referred by a PID have to be added to the pointing information (target URL) in the PID. By this, we are able to create a chain of trusted access information that enables secure access for research data and publications and we also secure location-independent data dissemination that cannot rely on network locations for trusted data serving.

Problem Statement 3 and *Problem Statement 4* are addressed in Chapter 6, where we address the unresolved challenge of PID maintenance efforts cause by location-dependent resolution targets and provide approaches for trustworthy data access in location-independent network using PID.

Chapter 4

Related Work

The approaches described in this thesis aim at decoupling PID from the necessity of location-dependent operations and at location-independent data dissemination through PID. Hence, PID and location-independent data dissemination are two essential key concepts in this thesis. In this chapter, we discuss the related work for the approaches presented in this thesis and cover their different aspects. For this, we have a look at the work in the area of data dissemination in overlay networks (cf. Section 4.1). Then, we focus on the research data dissemination in NDN in Section 4.2. The related work on PID in the context of NDN is presented in Section 4.3 together with a short look on running legacy applications in NDN provided in Section 4.5. Finally, a summary on the research delta is given in Section 4.6, which points out possible research contributions by this thesis.

4.1 Research Data Dissemination With Overlay Networks

Early approaches on using overlay networks for research data dissemination have originated in the area of data-intensive scientific applications. In those areas, massive amount of data is generated, e.g in experiments related to the High Energy Particle Physics (HEP) community such as the Large Hadron Collider (LHC) located at CERN. Therefore, the Grid community has developed own protocols and tools for sharing and replicating data sets in petabyte scale [92]. One technology is *GridFTP* that used multiple sources to download data in order to improve bandwidth utilization and distribute network loads [15]. Since the early 2000s, the amount of generated research data has significantly increased. Thus, the research community has proposed multiple ways to advance GridFTP in order to increase its performance. One publication that particularly aims at improving data distribution with overlay networks is provided by Khanna et al., who retrofitted GridFTP with a multi-point overlay network [93]. They explore the effects of multi-hop path splitting and multi-pathing to boost the file transfer performance in GridFTP. In multi-hop transfer data is stripped at the source and is then sent across multiple overlaying paths to the source. The overlay network

is dynamically constructed for each file transfer by calculating a graph based on the network properties such as bandwidth utilization. By this, different nodes are used for the transfer and multiple different independent network routes are employed simultaneously. At the data source, the chunks are combined into a working file. The approach uses location-based network transfer and relies on TCP for data transmission. However, this approach uses overlay networks for file transfer but is not linked to research data dissemination, where persistent data access is in place. For providing a long-term overlay network access, a landing page that is describing a dataset can be registered as a PID target. With the information on the landing page a data access can be provided. However, direct access to the data in the GridFTP overlay-network is not possible using a PID.

In 2010, Ramakrishnan et al. described an on-demand high throughput data transfer architecture for WAN data transport [94]. The target of the approach is to provide a reliable, secure and light-weight possibility for on-demand WAN file access with increased performance, minimized overhead and elimination of bandwidth bottle necks in comparison to standard location-dependent technology. The authors put their work in direct relation to other overlay networks such as BitTorrent and describe it as an evolution step, as not only the protocol layer is improved (as in the case of GridFTP by Khanna et al. [93]), but the entire approach is aligned for superior performance using overlay networks. Their approach is to provide a guaranteed bandwidth for file transfer between two WAN endpoints in order to facilitate replication of large research data sets that can grow up to a petabyte-scale. The endpoints can consist of multiple source and target servers. All TCP/IP nodes that are between the endpoints (routers, caching/proxy servers and upstream servers) are configured along a calculated network graph that forms the overlay network. This configuration involves adjustment of the TCP/IP parameters, a setup of the minimum bandwidth and an adjustment of the flow control settings with Quality of Service (QoS) measures. The calculation of the network graph for the dynamically provisioned overlay network is done on-demand. One interesting feature is the introduction of the on-demand scheme that enables scientists to place data on any source in the network, without updating a location-dependent identifier, because the architecture has to hold a central catalog of all data sets in the network in order to compute the network graph. Furthermore, telemetry data from the network is needed as input for the graph calculation. As shown in the last publication, the usage of overlay network is done, in order to improve certain aspects of data transmission and replication in the area of data-intensive scientific experiments and communities but not to improve long-term dissemination.

In 2012, Steer at al. presented an approach of using BitTorrent overlay networks for research data dissemination [95]. For realizing their approach, they used an existing research data repository software developed in the *data.bris* project. The software stack of data.bris is the foundation for the research data repository of the University of Bristol and features an extensible metadata regime, a Simple Web-service Offering Repository Deposit 2.0 (SWORD2) data deposit interface and makes use of BitTorrent for data dissemination. Furthermore, it allows the assignment of a DOI as a PID for each research data set and

it allows a HTTP download for research data and its associated metadata. However, a closer investigation reveals that BitTorrent is not integrated into data.bris in a technical sense. The integration is done by a modified ingest process for the research data sets. Instead of depositing the full research data, a torrent file is ingested in the repository, meaning that the direct access information for overlay data access is provided to the downloader (cf. Section 2.6.3). By this, the repository is only involved in the distribution of the access information but not in the distribution of the research data. The approach introduces a new stage between the PID-driven long-term access process and the data access using BitTorrent. The integration of PID is limited to the repository and is not extended into the domain of BitTorrent. The data.bris DOI PIDs point to the landing pages of the repository. Every further download of metadata or research data has to be invoked from the landing page after PID resolution. Hence, the availability of the PID target is dependent on the availability of the repository and its respective landing pages. As a result a broken approach is provided where research data is disseminated decentralized through a BitTorrent overlay networks, but the conceptual binding of the torrent access information to the torrent files available from the landing pages break the advantage. By this, the data.bris repository forms a focal point (or single point of failure) between the PIDs and decentralized data access.

In 2014, Cohen and Lo published a paper on the *Academic Torrents* community-maintained distributed repository [96]. Academic Torrents is a platform for sharing and exchanging scientific data and knowledge between researchers and interested people from the public. The platform is facilitating BitTorrent technology for a cost-efficient dissemination of large research data sets and publications. The goal of research data and publication dissemination using the potential of BitTorrent is to provide a low-cost platform with shared responsibilities and obligations. By this, the costs and efforts for data distribution capacity (storage and bandwidth) is shared between different voluntaries. It allows the acceleration of data distribution by using multiple sources using a swarm and web seeds (cf. Section 2.6.2). For this, Academic Torrents provides storage, upload bandwidth and a web portal for torrent access information [97]. Using the web portal, users can upload torrent files containing access information to their scientific data and publications. The access information are available in the web portal, where they are browsable and searchable for the public. For downloading the data sets and publications, uses can download the torrent files and download the files using a BitTorrent software.

4.2 Research Data Dissemination With Named Data Networking

In 2014, considerations on supporting climate research data exchange with NDN were published by Olschanowsky et al. [18]. In this publication, first approaches are made to improve research data exchange for climate data using NDN. In contrast to later

publications, research data management is not in the focus but rather the upgrade of the network and data repositories using a NDN approach. The approach aims at improving the data discovery using data names instead of location-dependent URLs or host names and thus at improving the data access with location-independent naming schemes implemented in NDN. For implementing the approach, a disruptive-free process is suggested that integrates NDN as back-end technology and uses well-integrated bridges such as *Filesystem in Userspace (FUSE)* adapters for integration. By this, users and information systems relying on interfaces and file systems are not requested to change their behavior. To interact with existing research data management repositories from the climate research community, translation rules from location-bound resources descriptors (HTTP-URLs) to NDN objects are employed.

As a next step of research data dissemination for data-intensive disciplines, publications exploiting the potential of NDN were published to solve the challenge of exchanging petabyte-scale research data. In November 2015, a publication by Fan et al. presented a scientific data management application designed and implemented on top of NDN [98]. This approach is an evolutionary step to the first research data distribution described by Olschanowsky et al. in 2014 [18]. The approach described in this publication covers the discovery, search and distribution of scientific data from the HEP and climate research community that are facing the challenges of distributing multi-petabyte data sets, which are located at different research facilities all over the planet. Hence, there is a specific need for a seamless publication of large data sets, reliable data transfers and efficient discovery of data sets in distributed storage environments. The authors' contribution is to decouple data access from data location for improving the data availability and to assure smooth data distribution through NDN by offering transparent fail-over and improved transmission performance through optimal source utilization. For formulating their approach, the authors investigate the working-principle of well-established tools from HEP and climate research community that are based on location-dependent network principles. These tools are *xroot* as distributed scientific data management system from the HEP community [99] and *Earth System Grid Federation (ESGF)* from the climate research community [8]. To replace those tools with NDN-enabled equivalents, a software architecture presented in this paper. This architecture features two software stacks – one for data set discovery and search and a second for federated and synchronized name catalogs. The data operations of the new replacement tools are designed and implemented on NDN network operations. For building a distributed data catalog, a scalable NDN name discovery system is proposed that uses NDN *ChronoSync* for decentralized data synchronization within NDN [68]. With ChronoSync, the catalog entries are exchanged between the data centers. For publishing the data within the federated data sites, hierarchical NDN names are required to submit entries to the data catalog because data names are used to assign namespaces to research data providers and individuals. In general, the publication by Fan et al. is important, as it shows that NDN is able to power scientific data management solutions. However, the approach does only cover basic aspects of research data curation and aims at providing an operational system. In particular, the publication of data is not in line with the demands

that research data publication makes. For publishing data, arbitrary namespaces are used that only adhere to the principle of hierarchical structure, as it is requested by NDN and the data catalog scheme. This approach is not sufficient for long-term data access, because data names are subject of constant change, especially if data names are reflecting organizational structures as suggested in the publication. Hence, the publication provides a valid approach for data distribution but does not fulfill the requirements of research data dissemination, where persistent data access is needed and can be provided by introducing PID concepts.

In December 2015, a publication by Shannigrahi et al. [17] was released that provides supplemental performance measurements for the approach provided by Fan et al. The performance measurements were conducted in a dedicated testbed of six nodes for climate applications consisting of machines with 40 cores each, 48TB disk space and 128GB RAM. The machines were interconnected with a 10GBit link and hosting over 50TB of climate data in total. The half part of the testbed was located at the Colorado State University, Lawrence Berkeley National Laboratory, NCAR-Wyoming Supercomputing Center. A second half was located on Energy Science Network (ESNet) nodes located in Fort Collins, Denver (Colorado, USA) and Sacramento (USA, California). This realistic network testbed that connected NDN nodes from three states over a distance of more than 1.500 km provided approximately 50% of the performance delivered by a mature TCP/IP-based based solution. Although the first transmission performance results delivered by NDN appear disappointing, the potentials for speed optimization have not been addressed in NDN software design yet [100]. Thus, transmission performance speed ups can be expected with a gradual mature NDN software. Regarding request delay performance, Shannigrahi et al. provide simulation results that show a better performance of NDN in comparison to TCP/IP-based solution. The request delay was reduced by 76% with NDN and provided faster remote access for shorter incremental data synchronization and random access. Our evaluation results presented in Section 5.5 are in line with the results of this publication.

4.3 Persistent Identifier in Named Data Networking

When moving from location-based data access to location-independent data access, persistent identifiers have to reflect this technological alternative. This adaption is necessary for providing access to location-independent resources through PID resolution and to resolve PIDs from a location-independent access technology in order to ensure the accessibility of billions of PIDs used for tagging scientific literature and research data. Hence, PIDs have to embrace location-independent access technologies such as BitTorrent and NDN.

In 2012, K. Sollins published a conceptual conference paper that introduces principles of persistent identifiers in the context of ICN in order to create a better identification system for ICN data objects [101]. The objective of the paper is to provide an identification

system, called Pervasive Persistent Identification System (PPINS) for information centric networking that meets the requirements of scalability, longevity, evolvability and security. The approach of the paper is to tackle the challenges of identifying and naming objects in different ICN systems, such as DONA, Network of Information (NETINF) and Publish-Subscribe Architecture (PURSUIT). The findings in the paper are aggregated into a selection criteria continuum that is used to build a PPINS based on the usage of URN. The URN system is transformed in the context of PPINS into a Pervasive Persistent Object Id (PPOID) suitable for the usage in different ICN families. Hence, the content of the paper is to formulate a common identifier for location-independent data access using ICN that provides an abstraction of the different naming schemes used in each ICN family. The target of Sollins is not to provide a persistent identifier scheme for ICN like networks but rather user PID principles to organize namespaces and bridge naming scheme heterogeneity. Hence, the design principles of PPINS are based on modularity and layering for making PPINS applicable for different ICNs like DONA, NETINF and CCN. A compatibility towards NDN can also be assumed, as the naming principles between CCN and NDN follow the same rules in its foundations (cf. Subsection 2.7.1). As the publication focuses on the design of PPINS and PPOID, its goal is to provide a solution for typical challenges that ICN naming schemes are facing, such as scalability, assignment of distributed namespaces and bootstrapping of new namespaces. Figure 4.1 shows the principle proposed by Sollins. In this figure, a data request is generated either as a result of a search engine query, a data catalog (attribute set) lookup or as a media reference (book reference). This request consists of a URN formatted data set. The data set is then transformed by the PPINS framework into a suitable naming scheme that is processable by an ICN family set. One particular criticism that is applicable towards the assumptions of this publication is that Sollins assumes a certain commonality between the ICN architectures. On a superficial view, commonalities between the architectures are existing, but the literature on ICN shows that there are fundamental differences in the architecture, naming schemes and the realizations in software. This fact is also mentioned by Sollins in the end of the paper (*"there is a small but growing trend in considering sets of architectural criteria."*), which indicates that the concept needs to be verified against real ICN architectures. In the next two paragraph, we follow the references from the Sollins publication that underline the differences between the ICN systems. By looking closer at the two referenced publications, we see that the differences of ICN families can be considered problematic towards the approaches presented in her paper.

The first reference is Ghodsi et al. [102]. They pointed out in 2011 that the *naming* of named objects is the most significant difference between ICN families and organization of the names has a significant impact on basic principles such as data routing. Hence, a modular resolution approach as suggested by Sollins must also incorporate the impact on the ICN specific principles that adhere to object naming – a simple mapping of the ID spaces as suggested in her publication is not sufficient.

The second source is Ahlgren et al. [103], which is a journal article from 2012, cited by Sollins. This journal articles points out the differences in all named ICN architectures. The article explains the most important differences regarding the namespace organization of

NETINF, DONA, and CCN/NDN, which have a different granularity in name structuring starting from *flat namespaces* to *structured flat namespaces* and reaching well-structured, *hierarchical namespaces*. In practice, a naming convention may overcome the problem of different granularity at the first hand but it will break the compatibility with other ICN parties that adhere to the name-dependent principles of route aggregation concerning data name granularity. Therefore, it is questionable whether the approach presented by Sollins is realizable beyond the state of concept, when an abstraction of ICN design and architectural fundamentals is not possible.
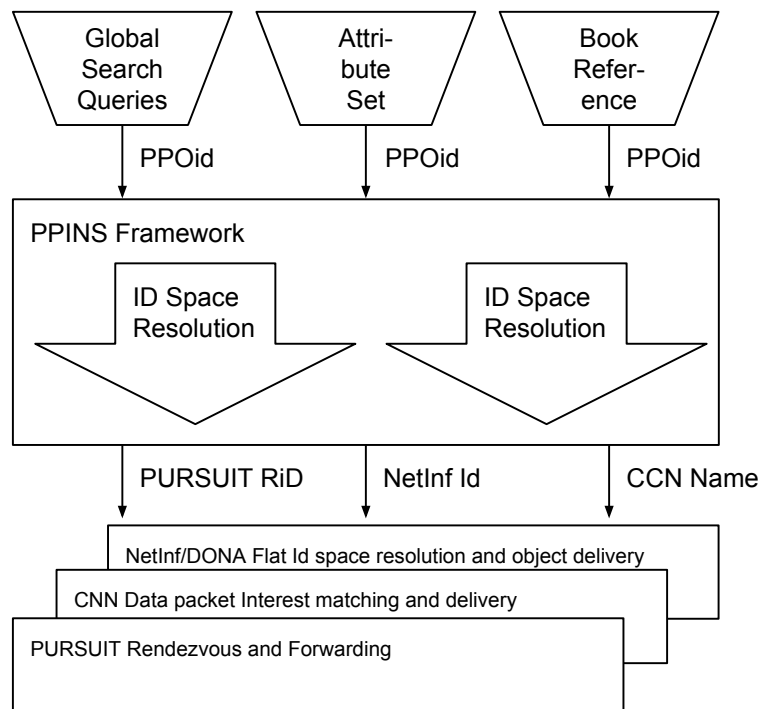
Figure 4.1: Pervasive Persistent Identification System (PPINS) concept proposed by Sollins [101]

Karakannas and Zhao published a research project report in 2014 which covers the topic of PID resolution to NDN data names [35]. Their work covers the resolution from the most popular PID systems to NDN data names using a centralized meta-resolution service. By this, their approach covers the mapping of different PID systems to NDN data names but not how the PID resolutions are performed in detail for PID each system. For their work they focus on URN, the Handle System (thus, also the DOI system), Archival Resource Key (ARK) and PURL as PID systems. Although, most PID systems are designed to be distributed and use delegated namespace hosting and responsibility, Karakannas and Zhao suggest abstraction layer on-top of the existing PID systems that does the translation from a given PID to a NDN name. This concept leads to the creation of a *meta-PID* service. They justify the need of a meta-service by the commonality found between the structure of PID

naming schemes. When looking at the assumption of PID system comonality more closely, we observe two fundamental problems.

First, all suggested PID systems contain the possibility to store a *resolution target*. This resolution target is available in all PID architectures and contains a location-based identifier often expressed as a URL. If a client wants to access any data behind a PID, the PID needs to be resolved by the PID infrastructure. Hence, if the client has a network connectivity to the PID infrastructure, no central meta-service like Karakannas and Zhao suggest is needed, as PIDs are self-contained. Hence, instead of introducing a meta-PID service, the connectivity of PID services into the NDN space is sufficient to solve the problem conceptually without adding any complexity. This approach is presented in the already published results of this thesis [104].

Secondly, the commonality between PID systems that is claimed in the paper, does not exist for all PID systems. On page 15 in Table 3 of their report, they interconnect the concept of PID resolution and PID naming in order to claim similarity between PURL and other PID systems. This interconnection is not true for URN, where namespace organization [105] is defined separately from the resolution mechanism [106]. However, it is true that the HTTP-based PID resolution for all PID systems work similar in practice. This could lead to the assumption that the same concepts of PID naming holds true for all PID systems. But the similarity is caused by the usage of HTTP that enforces certain design properties and not by the design of PID services and its namespaces.
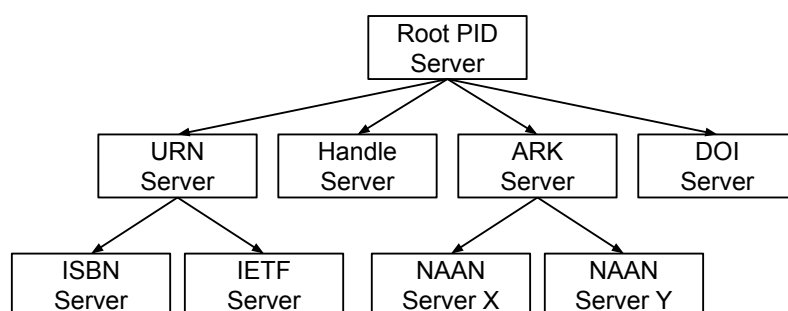
Figure 4.2: Proposed meta-PID service architecture by Karakannas and Zhao [35]

In order to understand this confusion in the appraoch of Karakannas and Zhao, the applied methodology has to be investigated [107]. For their research work they followed a three-stage approach. First, they conducted theoretical studies on latest ICN projects and PID standards. Then, they proposed a *mapping architecture design* based on the theoretical study, which is depicted in Figure 4.2. Finally, an evaluation of in-network caching was performed for big data objects but no evaluation was done for the mapping scheme or mapping architecture using a system comparison from literature. As a result, the contradictions in the PID part remains undiscovered.

In 2010, Dannewitz et al. published a conference paper on the realization of complex secure naming schemes for content-centric data [108]. In their publication, they formulate

the principle of name persistency for resources related to a content-centric network. However, they did not incorporate the principle of PID, although they state that basic security functionality for secure naming must be attached to the data and the employed naming schemes. They justify this principle by the fact that the identity of network locations cannot be used as a trust base in content-centric (location-independent) networks. The approaches presented in this thesis (cf. Chapter 6) follow the demands of Dannewitz et al. for secure location-independent data naming but attachs these principles directly to the persistency and security mechanisms provided through PID. Hence, the location-independent research data access through PIDs is fulfilling the requirement formulated by their publication and advances the state-of-the-art with complement data management principles inherited from PID.

4.4 Naming Schemes for Archive Data Access

For organizing and accessing archived data with the URN-like Magnet Links scheme, only little literature exists. Haun and Nürnberger introduced in their conference paper from 2013 a persistent identification scheme for organizing archived Personal Information Manager (PIM) application data on file systems [109]. For this, they proposed an approach using the Magnet Links scheme that is connected to attributes of common persistent identifier systems such as global id uniqueness, persistency and scalability. Their approach aims at accessing and archiving data on archive file systems stored on Wrote Once Read Multiple (WORM) discs. In contrast to this publication that initially introduces the usage of Magnet Links in the domain of data archiving, we propose an approach for globally distributed data with changing online data locations that is not bound to static structures of archive media. For persistency of access information and data identification, we employ a real-world PID system. Thus, we extend the usage of Magnet Links from the limited domain of data archiving in closed archives to the general domain of research data dissemination in a global network system.

4.5 Running Legacy Network Applications in NDN

For using legacy applications based on classic location-dependent networks in NDN environments, there are several approaches. An overview of these approaches will be given here, as they are important for adapting and realizing network functionalities in NDN that originates from location-based network technology.

A general approach on shifting applications from the IP domain to NDN is provided by Dai et al. [110]. In their publication, they investigate the behavior of the NDN PIT for application use cases of HTTP, File Transfer Protocol (FTP), P2P applications,

streaming media, E-Mail, online games and instant messaging. For this, they look at the challenge of NDN for solving the problem of bootstrapping two-way communications on top of a fundamentally one-way service, which is the nature of NDN. They sketch for each application use case possible transitions to NDN communication from the angle of communication models. Their approach does not include an in-depth explanation of protocol implementation but rather provides a high-level view on communication modes in NDN and the impact of the applications on NDN routing and PIT size and growth.

4.5.1 Location-based Network Protocols over NDN

One obvious approach is to run a TCP or UDP based connection over a NDN network. This approach uses encapsulation of network packets that are transported with a different network media. In location-dependent networking, this method is applied when running unencrypted protocols on an encrypted Virtual Private Network (VPN) that uses virtual network cards for operation system abstraction. In this approach, the transport layer of the OSI model is replaced by a NDN stack that takes care about transport. For NDN connection maintenance and bridging semantic gaps between the network models, additional functionality such as NDN session management for simulating a persistent TCP/IP connection over NDN has to be provided. This principle of transparent transport has not been widely adopted in literature, as most ICNs try to establish a contraposition against classic host-based networks and thus against location-dependent networks using TCP and UDP [11] [111]. As we can derive from the literature, this approach has limitations. In 2013, Xia et al. hold the opinion that conventional TCP cannot be mapped to ICN-based networks [112]. According to their publication, a modified variant of TCP Tahoe [73] [113] can be used for archiving a lossy TCP/IP transport in ICN. The challenges of using TCP within ICN networks is, according to the authors, based on four principles of ICN that are also valid for NDN applied in this thesis. The first principle highlighted by Xia et al. is that ICN is pull based. This means that sending data to another node in ICN requires a notification in form of an interest to the receiver first. The receiver consumes the interest and sends back a request for the data pull process (cf. Subsection 2.7.2). The second principle is that ICN uses an *end-to-network* principle that is in opposite to the location-based principle of TCP/IP, which uses a *host-to-host* principle. Additionally, as a third principle, ICN facilitates network caches. This caching is a source for errors in raw TCP communication and therefore needs to be turned off there. In NDN avoid the cache usage is possible by setting a `MustBeFresh` flag in the interest [114]. As a fourth principle, the authors emphasize the *point-to-multipoint* design of ICN that adheres also to NDN. In location-based IP networks, congestion control is designed to work on a single transmission channel. As NDN uses multiple connections and nodes simultaneously, multi-channel transmission control is necessary.

4.5.2 Application Protocol Adaption for NDN

As native TCP/IP transport is possible with limitation using NDN, the literature concentrates on porting specific protocols to NDN by focusing on the application protocol semantic and its associated data models. This can be done either by recreating the protocol and its semantics in an ICN specific implementation or by providing a proxy-based architecture that allows tunneling a specific protocol such as stateless HTTP through an ICN.

A reimplementation of a Voice Conference System (VCS) with protocols for conference discovery and voice data transmission has been created by Zhu at al. in 2011 [115]. In this publication, the mechanism of conference discovery that is done on location-based VCS via UDP broadcasting is realized as a NDN broadcast namespace. The voice communication requires an enumeration of voice devices and a negotiation of the available audio and video codecs, as well as the transport parameters. This is done in the publication by encoding the device IDs and available codecs into the NDN names. By this, a NDN replacement for every necessary mechanism that is included in a location-dependent VCS is provided. Hence, no adaption of the transport and session layers for NDN is necessary according to Zhu et al. but only a reimplementation of the VCS features in NDN.

Another example for an application protocol adaption for NDN towards HTTP is given in the following papers. In the paper by Shang et al. a JavaScript implementation of a client library for NDN is presented and compared against native HTTP implementation that relies on TCP [116]. Their JavaScript implementation is intended to work with modern web browsers. A comparison of data transfers done through CCNx (NDN.JS), the technological base for the current NFD, and TCP-based XMLHttpRequest (XHR) from the paper is provided in Table 4.1. From the table, we see that the performance of the throughput in Megabytes/s is bound to the transfer volumes. This is caused by the larger overhead in NDN for establishing a client-server connection and transmitting data for each request. With larger file sizes, fewer requests are needed and the share of protocol-related overhead for request handling is reduced in comparison to the transfer volume. As a result for smaller transfer volumes that come with a large amount of protocol overhead in comparison to the transported payload, the NDN performance degenerates up to $\approx 30\%$, while the HTTP throughput is diminished by $\approx 5\%$. Hence, the paper is a good example of successful NDN adaption of an application protocol, which does not yield a comparable performance in well-behaving networks.

File Size	NDN.JS (WebSocket)			Native HTTP (XHR)			CCN Test Utility
	Chrome	Firefox	Safari	Chrome	Firefox	Safari	
742 KB	46.01	48.66	65.66	83.6	84.73	82.51	71.21
28.7 MB	62.26	71.07	74.75	88.71	89.33	89.02	75.41

Table 4.1: Throughput Comparison of HTTP using NDN- and TCP-transport (in Megabytes/s) [116]

Another observation of an outperformed NDN network, against classic location-based technology using TCP-based HTTP, can be found in the publication on content distribution evaluation by Yuan and Crowley [117]. Figure 4.3 is presenting their results. On the left side, the average client download duration in relation to the number of connected clients of a CCNx-powered NDN CDN is compared against Lighttpd [118], a light-weight HTTP server. For a download of a 100 megabyte file in a clean network environment without any packet delays or packet loss, we see that Lighttpd is outperforming CCNx dramatically (lower is better). Furthermore, while the HTTP server scales out almost linear with an increasing number of clients, the CCNx realization is offering a less efficient scaling out behavior with an increased overhead for more than 20 connected clients in a clean network conditions. On the right side of Figure 4.3, the impact of network caching is depicted, where a comparison of network level caching for Squid [119] HTTP-caching proxies is done against native NDN caching. For the comparison two network topologies are used. The first topology includes one intermediate network layer with caches (*1-Level*), while the second topology contains two intermediate network layers with a doubled number of caches (*2-Level*). We can see in the Figure that the TCP-based HTTP implementation is outperforming NDN in a single and double level network caching hierarchy (lower is better). Moreover, NDN scales up worse with more introduced overhead in comparison to classic location-based network technology.

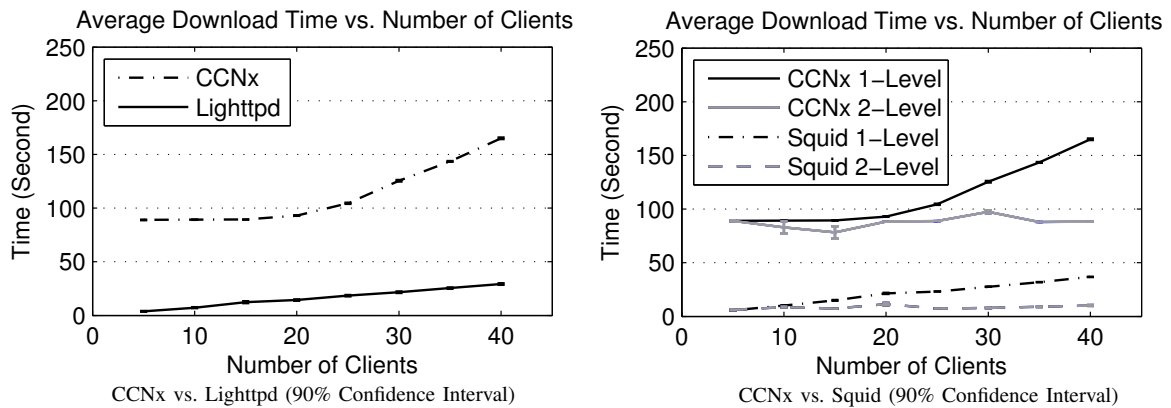Figure 4.3: Evaluation of Content Distribution with NDN and TCP-based HTTP [117]

But as this thesis goes along, we will see that the sheer throughput is not the most important factor for an efficient NDN adaption. However, the two publications on application protocol adaption for ICNs show that decades of optimization in location-dependent networks based on TCP and UDP are hard to outperform with application protocol adaption.

4.5.3 Communication Application Interfaces Adaption

Another approach is to exchange high-level Application Interfaces (API) in operating systems, software libraries and applications to provide ICN network access for legacy applications relying on location-dependent network connectivity. By this, the application can access data using a native (and often unmodified) API while the software components responsible for network communication are replaced by an ICN-capable version. This pattern provides an abstraction from network operation foundations. The implementation of the ICN capable version supports one specific set of protocols and uses for these a NDN adaption pattern, as presented in the publication of the VCS adaption by Zhu at al. The adaption of HTTP is particularly popular, as HTTP is stateless by design [43], which allows an easy encapsulation of HTTP-requests in interests and a data transmission through NDN data packets. Thus, it does not require a sophisticated session management adaption in NDN and already implements the semantics of caching that is also reflected on NDN network level. Figure 4.4 depicts an example for exchanging an existing software library for HTTP-based network access with a NDN-enabled software library, which offers the same interfaces to the network components of the software. In this figure, a NDN port of a web browser is shown. In order to make the web browser ready for NDN, an additional network library is added for NDN access besides a HTTP access library. This allows leaving most parts of the web browser (user interface, rendering and scripting engines) unmodified while also providing access to NDN network resources.
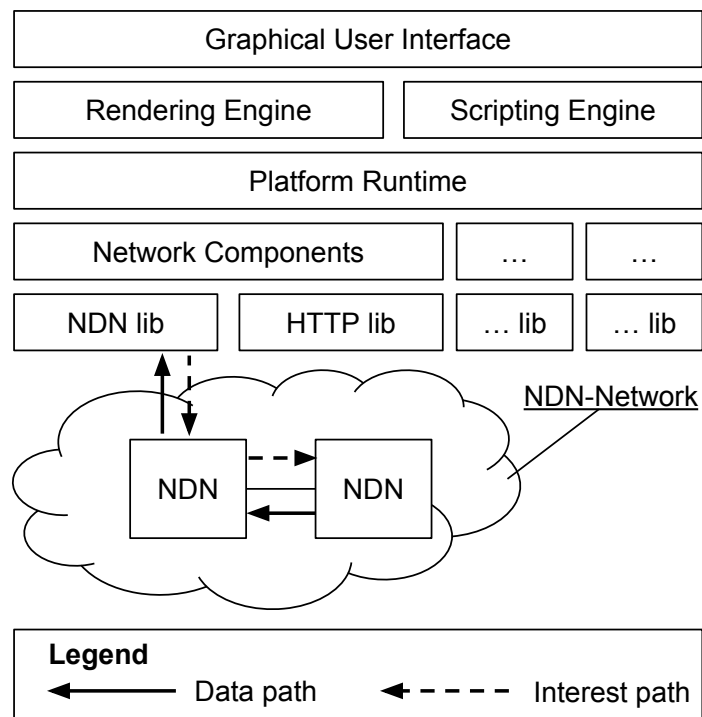
Figure 4.4: API-based NDN Integration into Legacy Applications (adapted from [120])

4.5.4 Transparent Proxies

An additional approach that is similar to the exchange of high-level APIs is the usage of application protocol aware NDN proxies. These proxies have two interfaces. One interface provides a native location-dependent OSI layer 7 application protocol such as HTTP. The other side is an ICN transport interface that sends and receives data and interests using a location-independent network connectivity. The proxy itself provides a boxing and unboxing of protocol related messages. For instance, HTTP requests are boxed into NDN requests and HTTP responses are boxed into NDN data packets. For applications that use the proxy server, the NDN transport is invisible and may only result in different response timing. Based on the location-dependent protocol, the proxy has to maintain additional data structures such as queues to fulfill the native protocol behavior. Furthermore, the proxy endpoints have to communicate for establishing and managing the NDN connection (out-of-band communication). Figure 4.5 shows two applications that communicate over a NDN proxy and use NDN for communication. By using a proxy server, applications can make use of NDN advantages without altering the application.
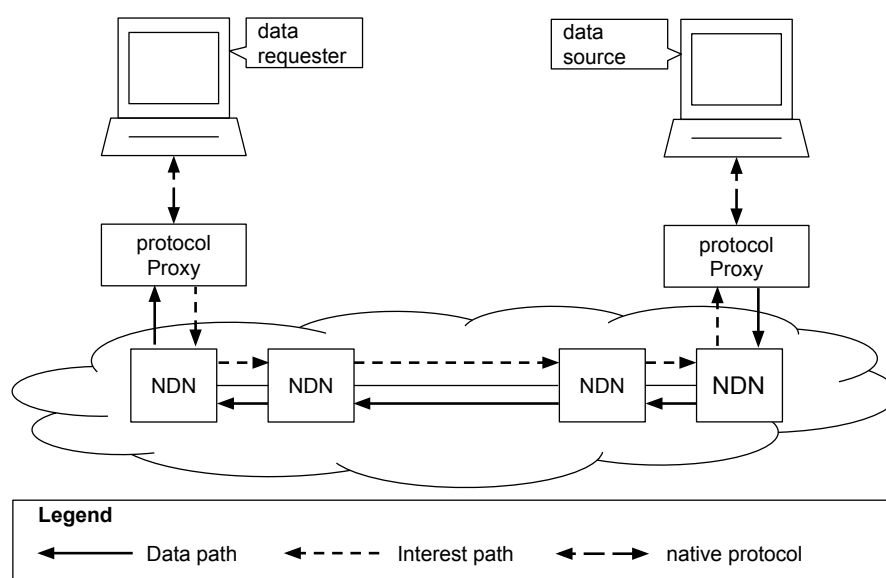
Figure 4.5: Proxy-based Architecture for Running Legacy Network Protocols Using a NDN Network

Publications that leverage the principles of application library and application protocol patterns particularly exist in the domain of HTTP, although the approach is not limited to this group of OSI layer 7 protocols. Wang et al. published a NDN enabled web browser in 2014 [120]. For this, they applied the concept of content access abstraction through application libraries and also make use of HTTP-NDN-proxies. An unmodified Webkit-based Hypertext Markup Language (HTML) rendering engine was combined with

NDN client library for accessing web content over NDN and CCNx in order to build an ICN-enabled web browser.

Another publication provided by Nan et al. in 2015, which also follows the approach of using application library setups, is comparable to Wang et al. [121]. In contrast to Wang et al., ICN-enabled network libraries are not only used in a web browser but also in Tomcat Java web container. By this, the web browser is able to consume ICN web resources through CCNx but also the modified Tomcat web container is able to offer web communication to Java web applications through CCNx. By this, they cover the full web stack by providing a web client (web browser) and a web server (Tomcat container). The modified Tomcat Java web container was developed by one of the coauthors in 2014 by Qiao et al. [122]. By making the web servers and web browsers ready for ICN, they overcome the proxy pattern that adds additional overhead and limits the application of ICN advantages on web applications. Hence, this publication can be seen as an evolutionary step based on the approaches provided by Wang et al. Tu et al. delivered also a performance assessment for the application-based ICN support together with Qiao and Nan in 2015 in a conference publication [123]. In this publication, Tu et al. provided an improvement to handle web related traffic with ICN connections by formulating a priority based dynamic web requests scheduling mechanism for NDN. This scheduling mechanism aligns the client web resource request patterns efficiently to the request and transmission patters that are suitable for NDN network operation. By this, the number of NDN round trips can be reduced and the response time is decreased.

4.6 Summary and Research Delta

The publications analyzed in the previous sections show that research data dissemination with location-independent access is not a completely new topic in the research community. Approaches from the early 2000s already came up with a break from a classic one-to-one connection in order to support multi-path sourcing for improving data transmission rates. One of these efforts is GridFTP [15]. The next step was to introduce simple overlay network structures that form a virtual network topology on a physical existing location-dependent network topology. Khanna et al. improved GridFTP by using a multi-point overlay network that uses intermediate network nodes to pass data from one node to the other [93]. By this, the end-point principle of classic location-based networks has been replaced by a node-based principle. Although this approach overcomes many problems of today's location-dependent networks such as host-failures, network breakdowns and congested connections, it relies on a location-dependent data organization that uses URLs for data classification or a central data catalog for GridFTP access. Additionally, the approach by Khanna et al. does not include long-term data access using a PID.

The approach provided by Steer at al. presents an idea of using BitTorrent overlay networks for research data dissemination [95]. This method combines research data repositories with the techniques for research data dissemination using overlay networks. The problem of this approach is that research data is not streamed from the repository to participants of the overlay network but rather the access information for overlay data access is stored within the repository. The research data is then streamed from external data sources (and not from the repository) using the overlay network to the participants. Hence, the data repository is only storing access information and the PID is linked to storage place of the access information in the repository. As a result, the publication by Steer at al. leaves open the challenge of a long-term dissemination of research data in location-independent overlay networks using a persistent identifier.

Olschanowsky et al. presented an approach to improve research climate data exchange with NDN [18]. Their work has similar goals as the publication by Khanna et al., particularly to improve the data transmission throughput and the availability of research data sets. Moreover, the discovery of research data should be improved using non-location-based data identifiers based on NDN data names. In this publication, long-term data access and research data management is not in the focus but rather the upgrade of the network and data repositories using a NDN approach.

Fan et al. took the next step in the improvement of research data distribution for very large data sets and decoupled data access from data location for improving the data availability and to assure smooth data distribution through NDN by offering transparent fail-over and improved transmission performance [98]. Their work concentrates on well-established tools from HEP and the climate research community that are based on location-dependent network principles. A direct link to PID concepts and long-term access is not provided. Similar to the GridFTP approaches, data access information has to be provided by external infrastructures that perform the necessary steps for long-term access. An integration between location-independent data access and long-term access information, e.g. through PIDs is not given. Concrete performance tests with NDN research data distribution in this area have been performed by Shannigrahi et al. [17].

Although, Cohen and Lo have discovered the potential of BitTorrent for the resource efficient dissemination of research data and publications, the platform Academic Torrents [96] [97] does not include PID support in October 2016. Thus, it is currently not possible to link research data stored as BitTorrent resource in a scientific publication or any other media using PID. A download of the torrent file or the Magnet Link is possible from the data catalog of the Academic Torrent digital repository. However, there is no possibility yet to generate or obtain a PID to the data set, because there are no theoretical or practical foundations for this case yet. This important piece is added by our approach presented in Chapter 6 that bridges the gap between PID and research data available through PID. By this, our approach presented in this thesis may have a significant impact on the success of data and publication dissemination using BitTorrent. It is the missing link to reliable citing of research data in publications that is disseminated through BitTorrent technology.

While the research area of ICN is currently lacking a direct and self-containing link to persistent access information for long-term data access, e.g. through a PID, approaches of PID-related communities have been investigated. The work by K. Sollins can be regarded as an initial central publication from the URN community that introduces the usage of PID in the context of ICN technology [101]. But while not looking at PID-enablement for long-term-specific location-independent data access, the concept of URN, that is also applicable for creating a PID infrastructure, is used to build a universal object identification scheme that bridges the differences between the ICN families and their different object naming schemes. Hence, the approach aims at improving ICN development through providing a naming scheme based on URN and to transport techniques and experiences from over 20 years of URN naming into the ICN community.

Another contribution in the field of PID towards NDN access is provided by Karakannas and Zhao [35]. Their work covers the resolution from the most popular PID systems to NDN data names. For this, they suggest using a meta-PID system, that maps NDN data names to different PID systems. But instead of improving the PID services directly for supporting NDN access and NDN resolution, an overarching abstraction layer is proposed. This abstraction layer in form of a meta-PID service is not solving the problem and makes the chain of services used for long-term data access longer and thus more fragile. As pointed out above, their approach is not complete and has issues regarding the methodology and assumptions on PID systems. Furthermore, the usage of a meta-service is superfluous, when PIDs are stated as native NDN data names (cf. Subsection 5.3.2).

Besides previous work on improved research data distribution and PID usage in the area of location-independent data access, techniques for implementing and adopting existing network applications in the ICN families of NDN and CCN have been proposed as well. Three common patterns have been identified from the literature. The first pattern is to encapsulate raw TCP traffic into ICN packets. Xia et al. explained that raw encapsulation of TCP is limited and that only a specific version of TCP, called *Tahoe*, is suitable for implementing this pattern [112]. The second pattern is to reimplement a location-dependent OSI layer 7 application protocol with an ICN-adapted version. This has been done for VCS applications by Zhu at al. [115] and for HTTP by Wang et al. who implemented a HTTP stack for an application library [120]. As a third pattern, the usage of application libraries that serve internal APIs to provide ICN access are another form of layer 7 protocol adaption. This has been done for web containers and browser by Qiao et al. [121] [122].

While the approach presented in this thesis makes use of the techniques described above, following research deltas exist, where we provide contributions by this thesis. We provide an approach for true long-term dissemination of research data through PID that is location-independent and allows the creation of PIDs that do not require adjustment of resolution targets. In contrast to the work of Steer at al., we provide data access directly through the PID to location-independent research data in overlay and named data networks without storing the access information in an intermediate infrastructure

as e.g. a digital repository. Additionally, in contrast to the work of Karakannas and Zhao, we formulate a concept that allows direct access to NDN data through PID without service abstraction or meta-services. By this, we can also contribute with our work on accessing location-independent research data to the challenges that have been formulated by Olschanowsky et al. and Fan et al. Our approach, that brings location-independent access and PID concepts together, does also support improved data dissemination *and* long-term data access. Thus, we make contributions to improved data transmissions through overlay networks and NDN usage by providing a natural way to access research data in PID resolution processes that are already part of today's researchers' workflows. Additionally, we bring NDN research data dissemination from early adopter stages to real applications by adding the concept of PID access. For this, we use techniques for NDN implementation presented by Xia et al., Qiao et al. and Nan et al. In the end of the thesis, we provide a long-term data access with integrated robustness against data source outages, low maintenance efforts due to persistent resolution targets. It is a natural link from PID resources located in web and print publications to research data stored in an intelligent location-independent network.

Hence, we can summarize the research delta from the related work as follows:

1.) Research dissemination has been investigated using location-independent access technology in an early stage. The primary goals were the enhancement of throughput and robustness using those technologies. However, an improvement in following the requirements of long-term location-independent research data dissemination has not been investigated in depth to our knowledge.

2.) Location-independent PID schemes have been proposed in the literature as high-level approaches. The past optimization goals were directed towards a cross-system access of different ICN families without taking the differences of the ICN designs into account. Other goals in the literature were directed towards providing an abstraction from the different PID systems, without incorporating the different foundations of the systems. As a result, a proposal for improving PID systems using location-independent technology has not been investigated in literature yet (beside our publications) that incorporates the PID system with the largest user base and includes its system foundations into a concrete ICN system. By this, we formulate an approach that includes the foundations of a distinct ICN and PID system to synthesizing a solution, which persists rigor inspections.

3.) Citation of research data in location-independent networks has been covered as a first approach in state-of-the-art research repositories employing BitTorrent. However, the citation information only include descriptive metadata of the data sets and no persistent identifier that allows direct access to the research data using BitTorrent or NDN. Hence, to our knowledge, the long-term dissemination of research data in location-independent networks is not possible in the approaches provided in the literature yet.

4.) NDN access information exchange is crucial for the success of NDN. In the literature provided before, the exchange of data in NDN networks has been described in depth on the network and architectural level. However, an improvement of storing and exchanging access information to NDN resources within the network that include more than the bare NDN data name has not been discussed. From our point of view, it is essential to provide an advanced access information scheme for location-independent network, as the location of the data cannot be used as a base of trustworthy data. For BitTorrent, these mechanisms exist, but they have not been extended to our knowledge into the domain of NDN, which is facing the same challenges of secure data provenience.

Chapter 5

Location-Independent Persistent Identifiers

In this chapter, we introduce our approach of conceptualizing, shifting and operating PID systems on location-independent network environments using NDN foundations. We point out the nature of PID and its relation to data locations in networks and sum up the benefits that are provided by transferring location-dependent network principles into the domain of persistent identifiers. For this, we use a multi-stage approach. At the start, we formulate our general principles for location-independent PID access. Secondly, we point out the namespace convergence for NDN and PID and explain the location-independent access models for Handle PIDs. Thirdly, we come to interoperability models for a seamless interaction of PID infrastructures for our and existing state-of-the-art approaches. Then, we present an implementation of our approach, which is evaluated using real-world PID data in a simulated environment. Finally, a discussion of our results is provided in Section 7.1, where we answer the research questions and point out the limitations of our methods in Section 7.2.

Aspects of our approach presented in this chapter have been published in a conference paper at the 10th IEEE Networking, Architecture and Storage (NAS) conference 2015 in Boston, MA, USA [104]. In this thesis, we provide an improved and extended version.

5.1 Persistent Identifier in Location-Independent Networks

In contrast to location-dependent data access, location-independent networks do not work with a transition of locations in order to access data. Hence, there is no challenge of locating data and solving the question of *where* data is. Instead, only the data content, meaning *what* needs to be specified. For persistent identifiers, the usage of location-independent paradigms can bring significant advantages. When looking at Figure 6.1, we can derive that a resolution does not cross the border between the location-dependent space and the

location-independent space. Thus, PIDs are resolvable in the location-independent space without any given resolution service URL or network address. In this space, PIDs can become true first-class data names that are reachable without any intermediate service as HTTP-proxies. The PID can be sent to the location-independent network and the network automatically routes the information to the PID-owning network entities. Then, the information behind the PID, which is also known as *resolution target*, is sent back to the requester. It remains invisible to the user, if the network topology is changed or the primary LHS site is changing its network location. Consequently, location-independent technology in conjunction with PID allows very robust and long lasting mechanisms for tagging and accessing digital research data that is resilient to network changes. Furthermore, it provides a reduced number of abstraction layers and eliminates the need for a centralized PID infrastructures.

5.2 Improvements and Benefits

Now, we look at the benefits and improvements of location-independent persistent identifiers:

1.) Reduced complexity in the PID resolution is an essential improvement of the approaches we present in this chapter. When moving PIDs into a location-independent environment, like NDN, no centralized end-user PID resolution services are needed, as stated in *problem statement 1* (cf. Figure 5.1). By this, a changing ownerships of central domains, such as `doi.org` or `handle.net` is not a thread to the end-user PID resolution on the Internet anymore.
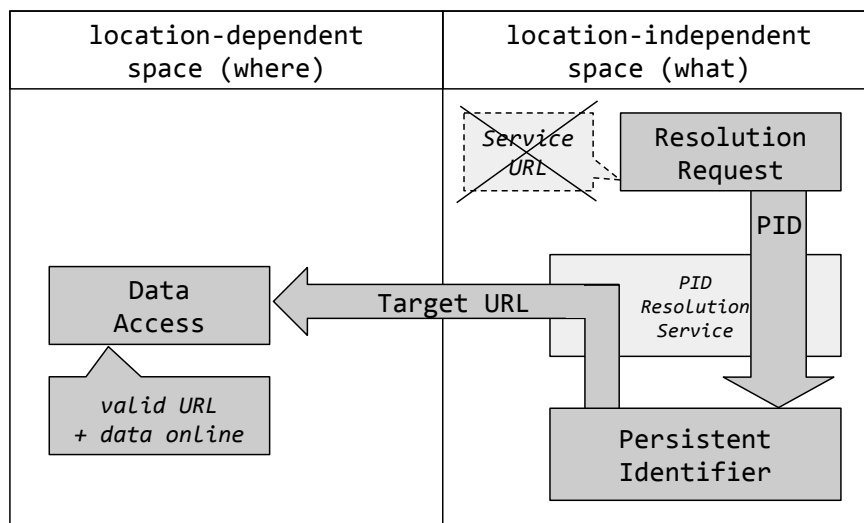
Figure 5.1: Location-Independent PIDs

2.) Decentralization of PID operation is an additional improvement that we provide with our approach. A decentralized infrastructure can be used to overcome the hierarchical bootstrapping procedure of the PID resolution process, as PIDs can be resolved directly from the LHS without contacting a GHR. This improvement is also helping to solve the challenges formulated in *problem statement 2*.

3.) Robust PID resolution is an additional benefit, we provide through our approach. As we will show in Section 5.5.4, our approach of location-independent PID access is able to resolve PIDs in erroneous network conditions, suffering under packet loss, more robust than classic location-based realizations.

4.) Multisite PID serving can be implemented with less effort in a NDN-enabled PID system. This is possible with a truly decentralized infrastructure (cf. benefit 2, above). In contrast to classic location-based data access, NDN supports multi-site data serving without any additional infrastructure. This allows accessing PIDs from the best available site without contacting the original data source. By this, PID access can be realized even if parts of the network or other Handle hosts responsible for a PID namespace are out of order or not reachable by the client. In contrast to the location-based PID access, this approach does not require maintaining a list of alternative PID mirrors with their respective DNS names or IP-addresses. This also allows adding and removing PID servers dynamically to the network, in order to compensate load peak situations, planned system maintenance or unplanned down times.

5.3 Approach

For releasing PIDs from its location-dependent network principles, a new concept is needed that embraces all aspects of the Handle PID fundamentals and its realization. In this chapter, we present our full-fledged approach of a complete PID system transformation into the location-independent paradigm provided by NDN. For this, we first turn our attention to the world model of PID that needs to be augmented to a model that does not rely on known and trust-worthy network locations, but embraces PIDs as self-containing instances and prototypes. As a result, the architecture of the Handle system as a realization of a PID system needs to be recomposed for location-independent operation, data access and maintenance, as well as assuring the necessary trust in PID data. Furthermore, the layers that are closer to the realization of PID use cases are extended. This is also the case for application protocols used in the Handle PID system. In the end, we provide an interoperability model for interaction with the existing location-based Handle world because the transition of the network paradigm requires a full-operational link to billions of existing PIDs to provide the paramount of *backward compatibility* and *cross-world access*.

5.3.1 General Principles

In its foundations, the approach of operating a PID system in location-independent networks based on NDN includes several fields of action. For this we will look at the following focus points:

1. **Namespace Convergence** – Section 5.3.2
   The Handle PID namespace and the NDN namespace are two distinct namespaces. As identification is a core concept of PID and NDN, a convergence of namespaces is crucial for realizing location-independent access through NDN towards PIDs.
2. **Access Model** – Section 5.3.3
   By the replacement of location-based network structures, the PID data access model will change. While location-based Handle PID systems only offer communication between a client and a server, location-independent access models offer more communication patterns and a two-tiered data access model depending on the dimension PID *instance* and PID *original*.
3. **Interoperability** – Section 5.3.4
   For interacting with the existing Handle PID world, an interoperability model is needed that provides access to and from PIDs located in the different worlds.

5.3.2 PID NDN Namespace Convergence

Naming of entities in NDN location-independent services is one of the most crucial design properties because it has direct impact on the transport and processing of data in the NDN network. As a consequence, the transition of PID to NDN requires an accurate naming scheme. This scheme has to follow four principles:

NDN Naming Restrictions
The NDN naming systems provides restrictions to creation of names [63]. In general, names have to be unique to prevent naming collisions and thus the creation of unsolvable routing decisions. The naming prefixes, which are the first components of NDN names, have to be announced to route and naming announcement services. Thus, the NDN naming prefixes have to be chosen in line with network naming authority mechanisms inherited by the NDN root namespace. Furthermore, they have to be hierarchical to provide efficient look up data structures [124].

Handle Naming Restrictions
The Handle system also provides, as any other PID infrastructure, restrictions concerning the naming of PIDs. In order to comply with existing PIDs, as explained in the Section 1.2 on the motivation of the thesis, the naming scheme has to adhere to the given naming principles. Thus, the naming scheme has to integrate a Handle *prefix* for dividing the Handle namespace into local sub-namespaces and a *suffix* for naming PID entities in local

namespace (cf. Section 2.4). Hence, the naming of entities has to include the tuple of prefix and suffix for every PID [37].

Global Accessibility

The ubiquitous accessibility of PIDs is of paramount importance. Thus, a hierarchical selection of the NDN name based on organizational ownerships like countries or network operators is not acceptable for PID, although it is stated in literature [98] [124]. Organizational structures are subject of constant change and thus NDN names that have an one-to-one relation to organizations do not meet the requirements of long-term access. Hence, the NDN data name that is chosen for the PID service has to be unique within the NDN root namespace. Therefore, a mechanism has to be found that assures the uniqueness of PID sub-namespaces in NDN. This implies that the root name of the PID system would be similar to a Top Level Domain (TLD) in DNS like *com* or *net*. The owner of the root NDN PID name is then not only responsible for assigning Handle prefixes but he simultaneously assigns NDN sub-namespaces. Of course, a decentralized assignment of sub-name spaces could be also possible. If an interconnection of PID infrastructures is required, like the resolvability of DOIs within the Handle System, the same root namespace can be used with a static name transformation rule.

Human Understandable

The location-independent PID access for NDN has to be intuitive for human beings. Thus, a simple rule has to exist that allows a transformation of a given Handle PID into a valid NDN name. In contrast to location-based PID systems, the user does need to know a currently valid URL of a resolution service like `dx.doi.org` or `hdl.handle.net`.

By incorporating the naming constraints from above, we can now formulate the naming convention for NDN-enabled PIDs. For lining out the details, we use Figure 5.2 and the numbers in bold font for explanation. The upper part of the figure shows the elements of the Handle PID naming system that are required for creating, resolving and maintaining a PID. The lower part shows the elements of the NDN system to formulate a valid and globally-routable data name. As a result, we show in the middle gray bar our suggested naming convention that includes the constraints of both systems and transfers a PID into a NDN name and vice versa. The PID names are enclosed as native NDN data names using a starting "/" according to the NDN naming conventions [63]. Then, the separation of the Handle Prefix, Handle Suffix and a *conventional extension* is also performed using a "/". Hence, the Handle Prefix and Suffix form hierarchical and thus routable NDN names. This is possible, as both systems treat naming particles of entities as first class citizens in their respective data models for entity naming [104]. The Handle Naming Authority ❶ is propagated as name in the NDN root namespace. Therefore, it allows a global resolution of PIDs from any given node of the NDN network and as a result, the Handle Prefix is a globally routable NDN name at the same time ❷. The unique local Handle name, known as suffix, is the distinct name of a PID in a sub-namespace ❸. This name often reflects the organizational naming of a PID related data and can it can exist multiple times in different

sub-namespaces, while it has to be unique at its own namespace. Similar to NDN, it allows a structuring of the sub-namespace to control the routing of Interests within NDN name ❹. For the application in NDN, we suggest that ❹ follows the tradition of PID infrastructure providers that either use an opaque naming scheme, e.g., random characters for naming, as it is applied in EPIC [125], or a naming scheme that does not need further adjustment of PID names, when organizational changes are necessary. For resolving a PID over NDN, the information of prefix and suffix is sufficient. Users have to keep in mind that they have to add a starting slash before the Handle PID to resolve it using NDN. For interacting with PIDs for maintenance and to perform special queries using parameters and filters, an optional extension of the NDN name is possible. These Handle PID messages and query paramemter ❺ can be added to the NDN name done using the field for *conventional extension* ❻. However, the *conventional extension* field is also used for other purposes in NDN (cf. Subsection 2.7.3 and Figure 2.13).
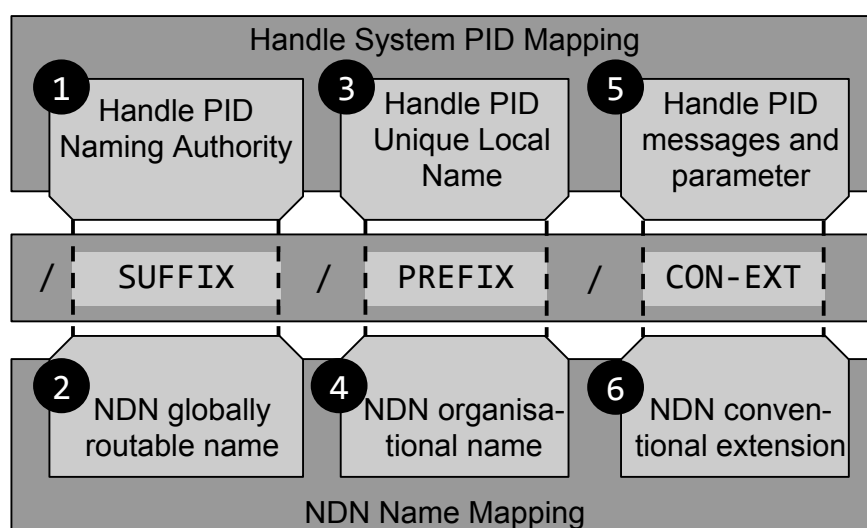
Figure 5.2: Naming Convention for NDN-enabled location-independent PIDs

5.3.3 Access Models

In classic location-based networks, PIDs stored as Handle records in the Handle system can be obtained from either the original data source or from a data source that provides a copy of the data sets (mirror). The GHR nominates the *primary data source* (LHS) that is responsible for serving a particular Handle prefix by stating its network location and sending the public key of the source in order to verify its genuineness. The GHR is also announcing mirror servers holding copies of the PIDs (secondary LHS) source [38]. As the Handle system uses a client-server paradigm, the connection is established between the client and the server endpoint, so changes on PIDs are immediately communicated back as success or fail [38]. Thus, for accessing PIDs for resolution and maintenance, the user can

be sure that in location-based networks PIDs are obtained from the original or an authorized mirror data source. If the client communicates with the primary LHS, it can be assured that changes are persistent to a PID. The changes are propagated from the primary LHS to the secondary (backup) LHS for data safety in fixed intervals or using polling.

Beside the client-server communication, the Handle system is capable of relaying connections between a client and a server through an intermediate Handle server. This relaying works transparent for both sides and is indicated by the *Authoritative bit (AT)* in the `OpFlags` of the Handle message header (cf. Figure 5.3) used in the native Handle protocol. Thus, the Handle system adheres strict to the location-based client server paradigm with optional intermediate relaying.

```
bit     →                                      1 ... 2 ... 3   3
offset  →  0   1   2   3   4   5   6   7   8   9   0 ... 0 ... 0   1
OpFlag  → ┌──┬──┬───┬───┬──┬──┬──┬──┬──┬───┬──────────────────┐
          │AT│CT│ENC│REC│CA│CN│KC│PO│RD│    │ ... reserved ... │
          └──┴──┴───┴───┴──┴──┴──┴──┴──┴───┴──────────────────┘
```
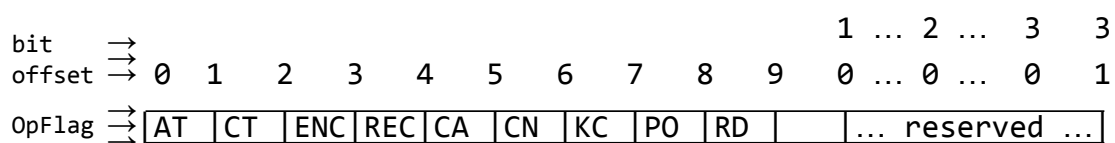
Figure 5.3: NDN Handle Message Header OpFlag [38]

In contrast to this, the location-independent access using PID spans an augmented access model into the domain of PID. The classic client-server communication is still active and needed to maintain the leading / original PID data set. But beside this, location-independent access, controlled by the description of the data in the access information, offers access to instances of PIDs that can exist as copies on different servers and network caches at the same time. This means that a request towards a PID can be answered by different (unknown) network entities. The primary LHS can answer the PID request for resolution but also network-based caches, like Content Store (CS) in the NFDs and the secondary LHS can fulfill requests. As a result, the requesters get their request fulfilled, not necessarily by the original PID but rather by a copy of the data existing outside the original PID source. The copies of the PID can also be of different ages and states. This instance-based access provides a new set of possibilities but also needs additional measures for controlling, trust and optimisation. We will investigate the potentials in Section 5.3.3.1 and 5.3.3.2.

5.3.3.1 Primary Source Access With NDN PID Push

Primary source access mimics the classic client-server communication within NDN for the Handle System. We call our approach for end-to-end communication as primary source access within the NDN space *NDN PID push*. It enables the communication to a specific server in the NDN network using a distinct server name. Thus, we provide an approach to fulfill the complete set of Handle communication between the client and the server. This includes all operations that can be performed using the native location-bound Handle

application protocol. By this, Handle servers can be fully administrated within through NDN communication. Furthermore, it facilitates a cross-world communication that expects a server peer or a client peer in the location-dependent network sphere and as a result, PID administration and resolution can be done from a location-based network into the NDN network using a gateway and vice versa. Figure 5.4 gives an insight into the basic mechanism of NDN PID push. As we can see in the figure, the client wants to communicate only with `LHS Site 4` from a LHS server group responsible for serving a specific Handle prefix. Although the NFDs provide multiple routes trough the NDN network and multiple sites are able to fulfill the request, NDN PID push encapsulates the Handle application traffic into a virtual network tunnel between the Handle client and `LHS Site 4`. The other peers are not contacted and the traffic is routed through the NDN network using different NDN techniques such as NLSR (cf. Section 2.7.6).
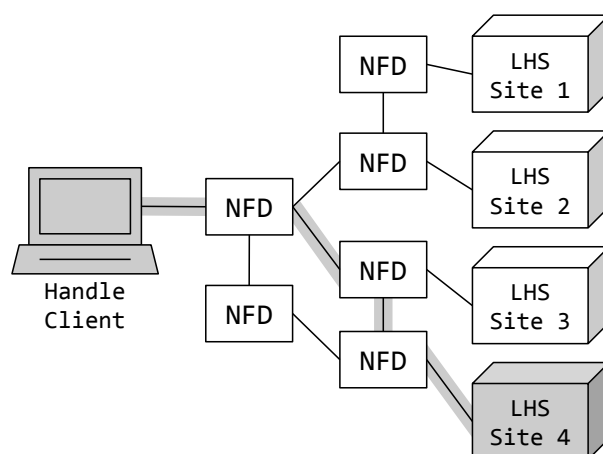
Figure 5.4: NDN PID Push Enables End-To-End NDN Communication

To perform arbitrary PID actions in NDN, the native Handle protocol originally designed for location-dependent end-to-end communication can be recomposed into a NDN protocol. This has not been covered in literature yet and will be part of our approach for a location-independent PID system. In order to make NDN PID push work within the NDN network, a number of prerequisites have to be fulfilled:

Distinct Server Names
In order to provide an end-to-end communication, each peer of the network needs a distinct NDN network name that makes it reachable via NDN network. As NDN-enabled Handle servers use their Handle prefix as NDN data name, a second data name for NDN PID push communication needs to by registered at the NDN face to provide this dedicated communication.

Re-enabled End-To-End Communication
NDN is a content-centric network design that is based on an interest-data retrieval cycles (cf. Section 2.7.2). *"It redefines the key primitives at the "waist of the network" to include simple request-reply interactions and a publish-subscribe information distribution paradigm, whichs enable features such as loop free forwarding, location independence and flow balance. By casting away the explicit requirement for end-to-end communication and by enabling caching en-route, it makes content readily available from different locations and distances from the receiver"* [111]. These design fundamentals are very beneficial for distributing content within a NDN network but for the case of PID-specific communication that adhere to end-to-end principles for secure PID creation and update propagation, the NDN principles leads to challenges. Thus, the end-to-end principle needs to be re-enabled within NDN, in order to make the PID usable within this network. This is necessary, as the newest state of the PID stored at the primary LHS needs to be determined for PID updates and non-cachable resolution requests. These primary PIDs have to be issued by a primary LHS and thus end-to-end connection between a client and a server is needed. The insights provided for the Handle PID system might be transferable to other domains like voice conferencing over NDN, where end-to-end connections are also required [115].

Control of NDN Caching
For persistent creation, update or change in Handle PIDs and administrative data, the built-in network caching mechanisms of NDN need to be controlled. This has to be done for all intermediate NFDs that are used for the data transfer between the two Handle parties (client-server or server-server) for primary source access using end-to-end communication with NDN PID push. Although network cache controlling is a feature of NDN [114], we present a new approach of transmitting data with a full-cache suppression using the NDN design paradigms. By this, we assure that only up-to-date data reaches Handle PID end-points and becomes part of the PID database at the primary LHS.

Intact Routing
Another perquisite that is needed for a successful end-to-end communication is an intact routing between the Handle client and the Handle server. This is not in the focus of this thesis but essential to guarantee a working data transmission through the NDN network.

With the measures in place, we can define our approach of NDN PID push:

Definition 5.1 (NDN PID Push) *NDN PID push is defined as end-to-end communication protocol between a NDN node representing the PID client and another NDN node representing the PID server. It allows a cacheless end-to-end communication over NDN using interest-based asynchronous communication with a reduced number of NDN round-trips for communication. By using distinct NDN data names, end-points can invoke PID-related communication between any PID system at any time. For realizing its functionality, NDN PID push is located in the OSI-model at the bottom of the application layer and provides an abstraction for the transport layer using location-independent NDN connectivity [69].*

Next, we explain the principles of the NDN PID push protocol, by having a short look at the native Handle application protocol that is used for PID-related communication in location-based networks.

For placing the Handle relevant parts into NDN PID push, we first have a look at the message format of a Handle request (cf. Figure 5.5). The native Handle application protocol is designed as a message oriented binary protocol with a small data foot print. This means that every section of Handle message and specific data fields are available at certain bit offsets. The values are stored as binary-flags or (unsigned) integer values. Based on the transport media functionality, Handle messages are broken down into transmission units like TCP or UDP packets. It is used for different purposes and it serves for transporting data and making RPC calls. To complete most use cases, like PID creation, multiple Handle messages are needed for executing a specific function. To realize the use cases, the Handle protocol implements a session mechanism for peer authentication (cf. Section 2.8.3 and Section 2.8.4) and key-exchange for synchronous encryption of application traffic (cf. Section 2.8.1). It is designed with the purpose to communicate over stateful and stateless communication media. Therefore, the protocol features functionality that is implemented in lower network layers redundantly to allow operation even on unreliable transmission media, like simple serial data buses. E.g., it features a message sequence number on application level, which is also implemented in the lower transport layer of TCP. The message format consists of a `Message Envelope`, `Message Header`, `Message Body` and a `Message Credential` part and is defined in the *System Protocol Specification* stated in IETF RFC 3652 [38]. In the following, we will briefly look at the parts of Handle protocol message, depicted in Figure 5.5.

❶	Message Envelope
❷	Message Header
❸	Message Body
❹	Message Credentials
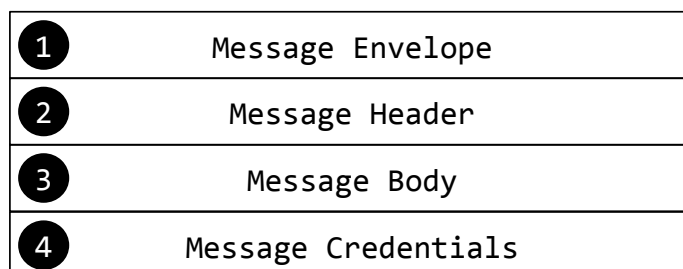
Figure 5.5: Handle Message Format [38]

The **Message Envelope** (cf. Figure 5.5, number ❶) is an obligatory data structure that is used in the Handle protocol to assign a message in its specific context. `MajorVersion` and `MinorVerion` are fields indicating the version of the protocol. They are used to determining the availability of cryptographic cipher suits and features available in the different Handle versions. The message envelope contains more state information that assign a Handle Message to a specific application context. This could be a session established between two Handle communication parties using a `SessionId` and/or part of a request using multiple messages joint by a `RequestId`. The `SequenceNumber` is used, if a Handle message is

fragmented into multiple transmission units. The field `MessageLength` depicts the size of the message in bytes. A complete overview is provided in Figure 5.6. The message header's Operation Flags (OpFlags) are depicted as a gray box in Figure 5.3.

```
bit      →  0                    1                    2                    3
offset   →  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
fields
 ↓↓↓     ┌──────────────────┬──────────────────┬────────────────────────────┐
         │   MajorVersion   │   MinorVersion   │        MessageFlags        │
         ├──────────────────┴──────────────────┴────────────────────────────┤
         │                            SessionId                             │
         ├──────────────────────────────────────────────────────────────────┤
         │                            RequestId                             │
         ├──────────────────────────────────────────────────────────────────┤
         │                         SequenceNumber                           │
         ├──────────────────────────────────────────────────────────────────┤
         │                         MessageLength                            │
         └──────────────────────────────────────────────────────────────────┘
```
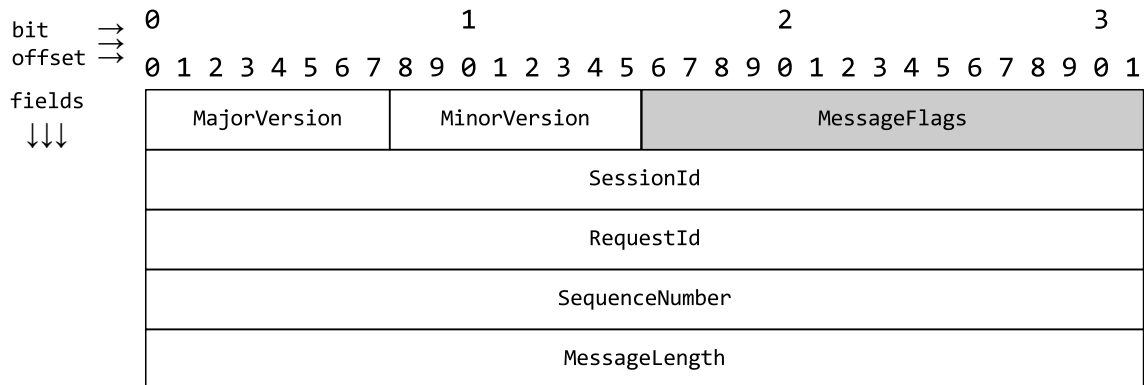
Figure 5.6: Handle Message Envelope [38]

Message Headers (cf. Figure 5.5, number ❷) hold the relevant information of the request and consist of operation codes, the requested operations, the associated operation options and response codes.

```
bit      →  0                    1                    2                    3
offset   →  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
fields
 ↓↓↓     ┌──────────────────────────────────────────────────────────────────┐
         │                              OpCode                              │
         ├──────────────────────────────────────────────────────────────────┤
         │                           ResponseCode                           │
         ├──────────────────────────────────────────────────────────────────┤
         │                              OpFlag                              │
         ├──────────────────────────────────┬───────────────────────────────┤
         │       SiteInfoSerialNumber       │        RecursionCount         │
         ├──────────────────────────────────┴───────────────────────────────┤
         │                          ExpirationTime                          │
         ├──────────────────────────────────────────────────────────────────┤
         │                            BodyLength                            │
         └──────────────────────────────────────────────────────────────────┘
```
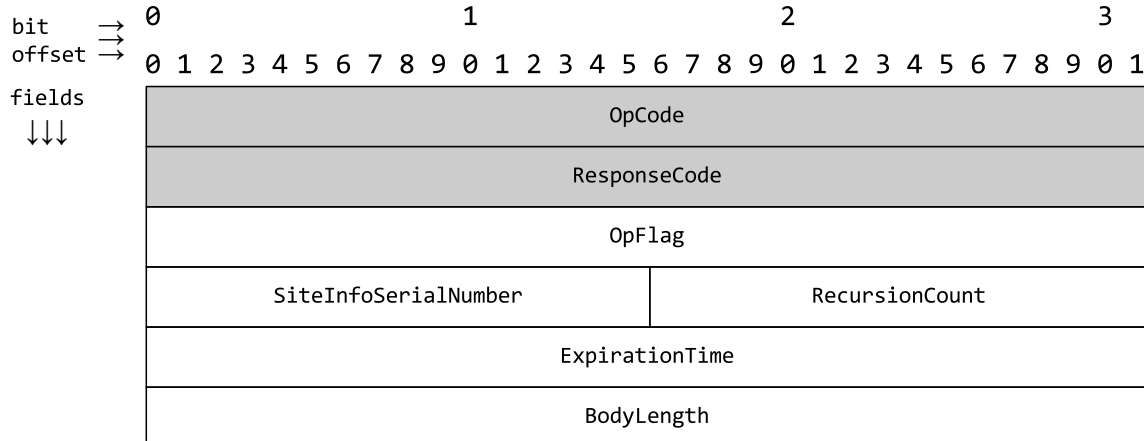
Figure 5.7: Handle Message Header [38]

The `OpCode` (cf. Figure 5.7), is a four-byte unsigned integer, which specifies the intended operation. With its defined set of operations, it forms the command set between the client and the server in the Handle system. For the adaption in NDN, the `OpCode` remains unchanged. However, the execution of the commands is done differently in the location-independent setting of NDN. In Table 5.1, we provide a shortened overview of the OpCodes.

OpCode	Symbolic Name	Remark
0	OC_RESERVED	Reserved Command
1	OC_RESOLUTION *	Handle Query
2	OC_GET_SITEINFO *	Get Handle Site configuration
100	OC_CREATE_HANDLE	Create a new Handle
101	OC_DELETE_HANDLE	Delete a Handle
102	OC_ADD_VALUE	Add Handle values
103	OC_REMOVE_VALUE	Remove Handle values
104	OC_MODIFY_VALUE	Modify Handle values
105	OC_LIST_HANDLES *	List Handles
105	OC_LIST_NA *	List sub-naming authorities

Table 5.1: Overview of the Handle Header OpCodes [38]

The message header shares the size with the ResponseCode (cf. Figure 5.7, number ❷) that indicates the result of a service request. Both fields can remain identical for the NDN transition. The ResponseCode is the answer from a Handle peer that is returned as answer to a request. For better understanding, we provide an excerpt of the Handle ResponseCodes in Table 5.2. The response code reflect the application logic of the Handle system but is also used to return responses related to remote procedure calls and communication related requests.

ResponseCode	Symbolic Name	Remark
0	RC_RESERVED	Reserved for request
1	RC_SUCCESS	Success response
2	RC_ERROR	Error response
3	RC_SERVER_BUSY	Server too busy to respond
4	RC_PROTOCOL_ERROR	Unrecognizable message
5	RC_OPERATION_DENIED	Unsupported operation
100	RC_HANDLE_NOT_FOUND	Handle not found
101	RC_HANDLE_ALREADY_EXIST	Handle already exists
102	RC_INVALID_HANDLE	Encoding or syntax Error
200	RC_VALUE_NOT_FOUND	Value not found
201	RC_VALUE_ALREADY_EXIST	Value already exists
202	RC_VALUE_INVALID	Invalid Handle value
300	RC_EXPIRED_SITE_INFO	SITE_INFO outdated
301	RC_SERVER_NOT_RESP	Server not responsible

Table 5.2: Overview of the Handle Header ResponseCodes [38]

Message Body (cf. Figure 5.5, number ❸) is the subsequent part of the message header. Its structure is not modified for the adaption of NDN but the transport mechanisms is changed for NDN data transport. The message body can be empty for `OpCodes` that do not require input data and change an internal state of the Handle server. For requests that require data such as the addition or the modification of a PID, the message body contains bytes as UTF-8 string representation consisting of key-value pairs (cf. Figure 2.5). The value section allows besides single strings also an entire vector of strings. Cryptographic information is exchanged in the message body, using the same content encoding for all non-content related security.

The **Message Credentials** (cf. Figure 5.5, number ❹) contain information for end-point authentication against a PID data source. For this, challenge-response messages are exchanged to prove the ownership of a pre-shared key between the client and the server or the possession of a cryptographic key. After the authentication, it is used to establish an encrypted session using a derived session key. This session key is generated for every new session. Message credentials are essential for conducting administrative tasks in the Handle system and to access and update PIDs with restrictive access rights.

In the following, we describe our approach for transporting Handle protocol messages with NDN. For this, we create a low-level communication adaption for the network API and which is responsible for transporting application protocol messages from one NDN network node to the other. In Figure 5.8, we provide a scheme of the design effort where the narrow boxes depict the existing native Handle communication stack. The wider boxes below the layers for communication transport over NDN depict our contributions. This allows us to use the full functionality of the native Handle protocol over NDN, without a major protocol redesign. It is particularly useful for all administrative and authenticated command sequences and allows a true end-to-end communication.
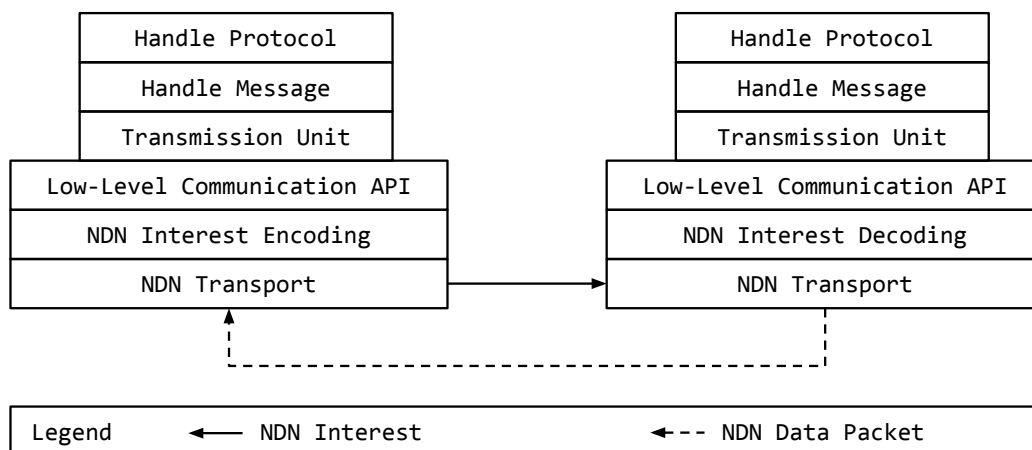

Figure 5.8: NDN PID Push Transport

For interfacing with the low level communication, our approach provides a *socket API* that is transparent to the native Handle protocol. The challenge for socket API adaptions in location-independent networks that mimic the nature of location-based socket APIs is that no complete implementation of location-based network transport protocols can be formulated in NDN. Hence, as pointed out in Section 4.5.1, a direct replacement of TCP or UDP is not possible with a standard conforming realization. Thus, to provide a socket stub that is sufficient to transport and provide the necessary properties of the Handle transport, we have to find shortcuts in the Handle protocol design that allow an adaption of a low-level socket API in NDN. For realizing this, our approach utilizes all information from the native Handle protocol and its design properties to create a functional socket API stub. For the Handle protocol this is possible due to following properties of the working principle and design:

1.) Low Level Information in the Application Protocol
Unlike other OSI layer 7 application protocols, the native Handle protocol features low-level transport information in its application protocol. We see in Figure 5.6 this kind of information such as the `message length` and the `sequence number` in the Handle message envelope. Additionally, we observe in the Handle message header useful information, as e.g. the `Expiration Time` (cf. Figure 5.7). This information is maintained in the application protocol and provides a priori details on the data that needs to be transported over NDN. This information is very important for our NDN API socket approach, as we can use them lateron to compute the missing information for a Handle message transport over NDN.

2.) Separate State Maintenance
The Handle protocol is able to store state information in Handle messages that are incoming and outgoing. Thus, the Handle protocol implementations keep track on Handle request states and assigns Handle messages to requests. It is also able to keep track of application sessions states. As a result, the native Handle protocol can operate in highly asynchronous network environments where the underlying network transport layer cannot make guarantees on connection persistence and latency. Hence, the Handle protocol is able to run using UDP and TCP that have different assumptions on the transport layer operation. The Handle message envelope provides state information in each Handle message like `RequestId` and `SessionId`. (cf. Figure 5.6). As a result, our transport socket adaption requires no maintenance of complex states on established end-to-end connection as this is done by the Handle implementation. This is very important, as NDN cannot provide those state information due its design principles of omitting the need of end-to-end connections.

3.) Loop Detection
As a special design property of the native Handle application protocol, the detection of network loops is part of the specifications. In classic location-based networks based on IP, routings must be free of routes that route data packets in circle. Thus, most application protocols running on top of IP do not feature a loop detection but rather rely

on the transport layer mechanisms like Time To Live (TTL) for IPv4 counters [126], or Hop Limits for IPv6 [127]. As a result, if the underlying transport protocol does not support TTL-like mechanisms with an identical semantic, as it is the case for NDN [11], application protocol adaption is complex or even impossible for certain routing topologies. The Handle protocol designers were also aware of this problem and thus, the Handle protocol features `RecursionCount` in the Handle Message header (cf. Figure 5.7), which is decremented for each intermediate station a Handle message passes. Hence, native Handle protocol implementations are able to detect circular routings in networks that break end-to-end communication. For this, the socket API adaption can observe and control the `RecursionCount` in the Handle protocol header, in order to drop Handle messages that were passed in an (unintended) NDN loop routing (cf. Section 2.7.5). Consequently, the socket API adaption can determine if a NDN network face is suitable for NDN transport with the Handle protocol by detecting inappropriate routing situations for a required end-to-end communication.

To formulate a socket-like API approach, we focus on the mechanisms of UDP communication sockets that are adapted for the native Handle protocol data transport. The UDP socket is a stateless end-to-end communication API, which is able to send and receive Handle protocol messages. It works with a fire-and-forget principle to send transmission units to remote Handle peers. For a working transmission of UDP packets, another peer has to listen to a UDP socket and a valid UDP routing has to exist within the location-based network.

Now, we look at the adaption of existing NDN principles for realizing a UDP-like socket. As we will see, the existing principles are not sufficient for doing this. By design, NDN implements a request and response network principle, thus spontaneous data transmission from one peer to another has some hurdles in NDN and can be archived with a mechanism adhering to the original NDN design. The approach that can be found in the literature is a regular polling from the receiving peer towards the sending data source, requesting potential data using NDN interests. If new data is available, it is delivered from the data source to the receiver using a NDN data packet. Although this approach works, it has the drawback of using resources very inefficient due to frequent unsuccessful polls [68] [128]. This means that additional NDN round-trips are required to initiate receiver-based polling from the data source by invoking the polling process using a RPC command over NDN. Polling has also the disadvantage of slower latency because data has to wait at the source until the next polling interval is reached. Furthermore, it does not scale well, as only a limited number of receivers can poll the data source due to limited network bandwidth and system resources. As a consequence, different approaches have been developed for ad hoc data transfer that does not rely on polling in NDN. Chen et al. have proposed *Content Oriented Publish/Subscribe System (COPSS)* for more efficient distribution of data removing the limitations of NDN polling [128]. Zhu and Afanasyev have proposed *ChronoSync* as an alternative source to receiver ad hoc data transmission scheme [68]. We propose NDN PID push which is an additional approach for ad hoc pushing data from a source to a receiver that does not use NDN polling.

The approach of NDN PID pull aims at embedding payload data into NDN interests. As NDN interests do not have a data payload section by design, we use the NDN data name section of the interest for embedding arbitrary data payloads. Figure 5.9 gives an overview of the embedding procedure. The principle of embedding data into NDN interests is not new at this stage and was proposed in [68] before. However, we evolve this principle to a parallelized data streaming mechanisms, which incorporates additional low-level information from the native Handle protocol. By this, we can transport Handle-related data of arbitrary sizes over the NDN network. A three-tiered payload conversion process for payload encoding is integrated to meet the requirements of optimal NDN interest sizes. The payload encoding uses a binary-to-text method that encodes payloads into two byte character sequences. This encoding brings the challenge that NDN data names also uses two byte character sequences for the organization of the technical fields in the interest name. As a result, an escaping of the encoded payload sequences is necessary, to avoid a collision of the two byte payload octets and the reserved field markers. Two byte octets that are responsible for collisions are the sequences 0xC0, 0xC1, and 0xF5 to 0xFF (shown as hex values). When this is done, the encoded payload can be part of the interest NDN data name without corrupting it [63] [104].

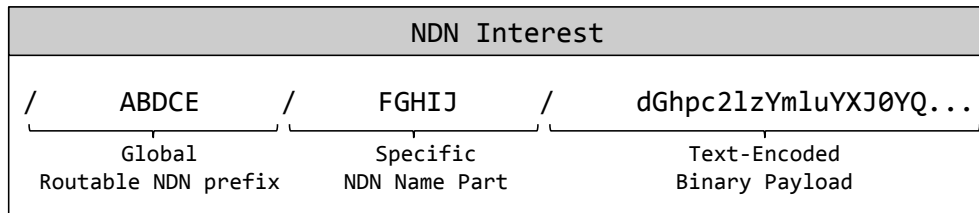| NDN Interest		
/ ABDCE / FGHIJ / dGhpc2lzYmluYXJ0YQ...		
Global Routable NDN prefix	Specific NDN Name Part	Text-Encoded Binary Payload

Figure 5.9: Payload Embedding in NDN Interest

For transporting Handle protocol messages in NDN interest, the payload needs to be broken down into multiple interests. In this process, the size of the interests can be adjusted as well, to assure an efficient handling of the interests within the NDN network. For this, we construct a pipeline that features a one-way design, which is only able to breakdown, encode and send at once. As a socket API semantic offers a two-way communication paradigm that is able send and receiver data using a single socket instance, two pipelines are needed for each communication direction. Now we describe the pipeline for sending data over NDN and then, we describe the reversed pipeline location at the receiver side in the latter.

In Figure 5.10, the streaming pipeline at the sender side is depicted. It features a parallel design to utilize multiple CPUs at once for encoding and data compression. The generated interests are handed over to multiple threads at the NFD for a parallelized transmission. At the first stage ❶, the Handle message is decomposed into transmission units that are on the same conceptual stage like UDP packets. This step splits the Handle messages into *n* chunks with a fixed size limit and transport relevant meta-information are attached to the transmission unit. At the second stage ❷, the transmission units pass the

binary-to-text encoder. The transmission units can be encoded in parallel, as they contain all transport-relevant information. After the encoding, the NDN interest are generated. At the last stage ❸, the interests are sent out to the NFD. As text-encoding is increasing the size of the data, an optional data compression can be applied on the payload in stage ❶ [129].
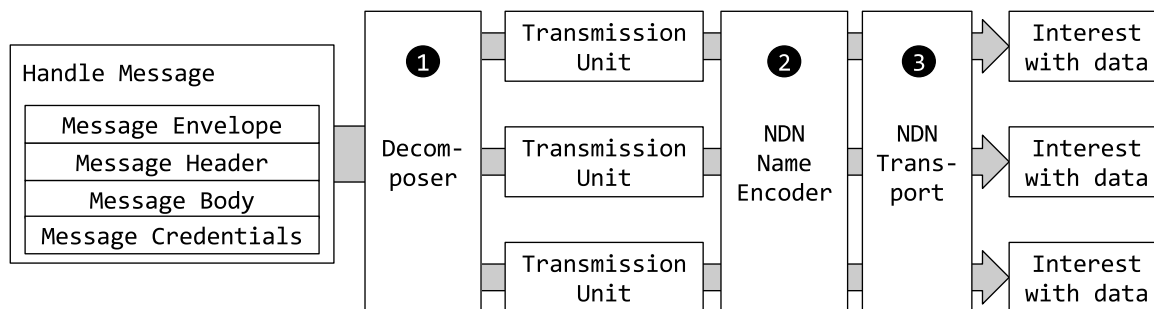
Figure 5.10: NDN PID Push Message Decomposition

For a better understanding of NDN PID push, we now describe our proposed scheme of the transmission units using Figure 5.11. The transmission unit contains three different fields. The first field `Binary Payload` contains a chunk of a Handle message. Payload sizes are stored is the field `Payload Size`. The `Chunk Nonce` is an identifier that assigns all transmission units to a Handle message. It is used to reconstruct the Handle Message at the receiver side. The `Sequence Number` describes the binary offset of the wrapped Handle message to reorder the transmission units on the receiver side. To attribute the transmission unit to the sending NDN node, the NDN name of the sender is added in the field `Sender NDN Name`. This is necessary because the Handle server and client can simultaneously operate transmissions to different network end-points over one NDN face. On the receiver side, queues for incoming transmission units can be maintained for each data sender to provide an efficient reconstruction of the Handle messages.
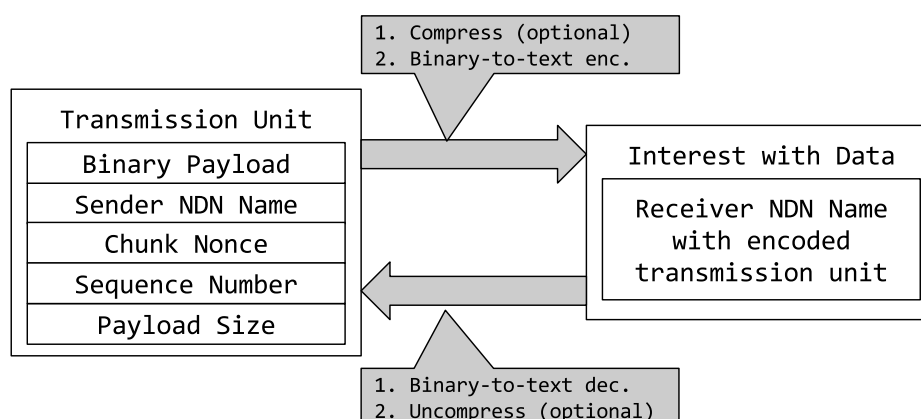
Figure 5.11: NDN PID Push Transmission Unit Encoding / Decoding

89

For receiving Handle messages using NDN PID push, a reverse data processing pipeline needs to be setup with a queueing/buffering system capable for handling parallel transmissions. The reconstruction of Handle messages from transmission units produces computational overhead. In Figure 5.12, we depict the data receiving and reconstruction process in detail. The NDN transport stage ❶ receives all incoming interests. Then, the Name Decoder stage ❷ separates the text-encoded payload from the NDN interest and decodes them back into transmission units (cf. Figure 5.11). In the third stage ❸, the transmission units are buffered. Due to the parallel design of NDN PID pull and the asynchronous communication principle of NDN, interests can take different routes within the NDN network and arrive in a different order. Thus, the chunk nonce and the sequence number are used for rebuilding the Handle message out of the transmission units. Stage ❹ *Composer* is responsible for reconstructing the Handle messages. After the reconstruction, the Handle implementation takes over the Handle message for further high-level interactions, such as PID resolving.
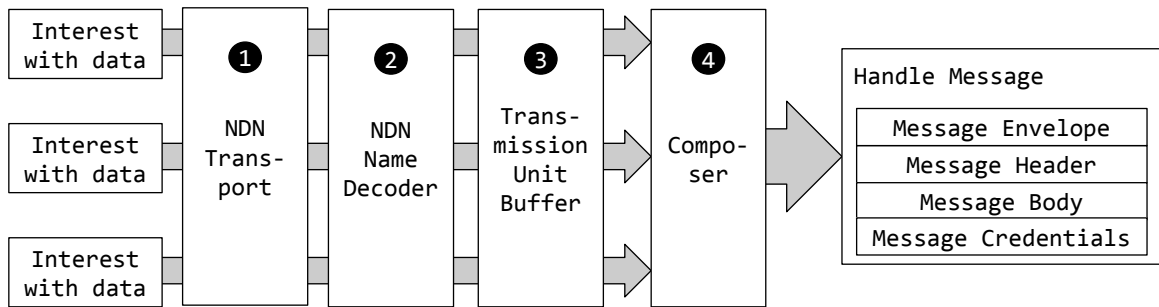

Figure 5.12: NDN PID Push Message Composition

The NDN network design aims at transporting payload data through data packets and not through interest but as we break in NDN PID push with this design convention, we have to investigate the impact of this decision. For this, we provide an insight into the transporting sequences of a NDN network node, while running payload-enriched interests through it. In Figure 5.13, a NFD is depicted with all relevant data structures involved in PID push. In step ❶, the interest arrives at one face of the NFD and is forwarded to a face, which is connected to the next NFD ❷. The forwarding of the interest is determined by the stored routing information in the FIB. In parallel, a copy of the interest is stored in the PIT ❸ until a matching data packet is received by the NFD. This behavior is enforced by the NDN design. However, this copying of interests is useful, because if new routing information is added to the NFD, the pending interest can be retransmitted by the NFD using a different outgoing face. When an interest has been received by the end-point, an empty data packet, we define it as *ACK packet*, is emitted and flowing back the chain of all intermediate NFDs to the data source. If an ACK data packet arrives ❹, the interest copy is deleted from the PIT ❺. The ACK data packet is forwarded in parallel to the face, where the interest has arrived ❻, in order to pass it to the previous (intermediate) node or the NDN PID push socket node.

The usage of ACK packets has the advantage that it is possible to determine for each data sending end-point if an interest has been arrived at the receiving network end-point. If no ACK packet has been received at the sender side within a certain time window, a timeout can be sent to the NDN PID socket in order to perform an error correction through an interest retransmission.
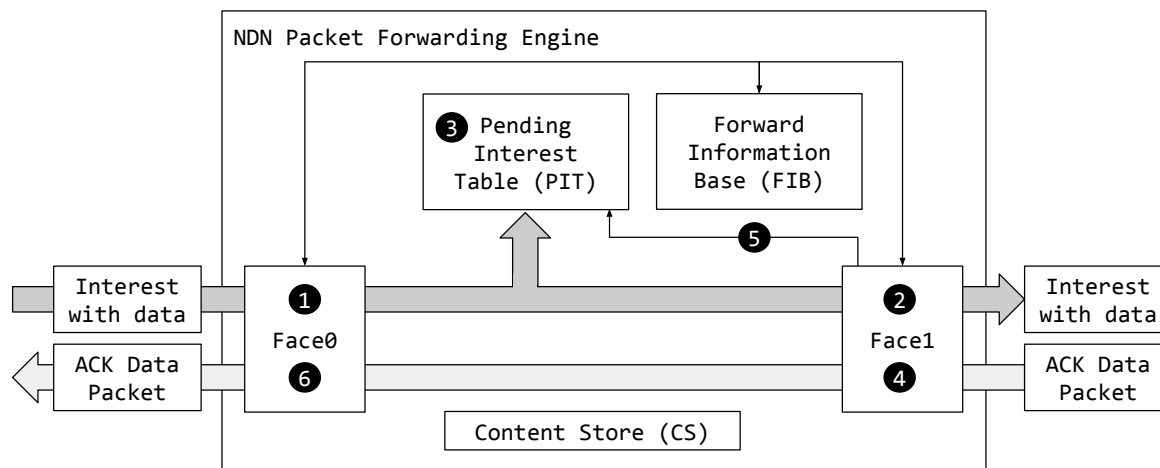

Figure 5.13: NDN PID Push Interest Data Forwarding

In order to show that the approach of NDN PID push is feasible on NDN networks, a clean room implementation is given in Section 5.4.3 and is evaluated in Section 5.5.3.

5.3.3.2 Instance Access With NDN PID Pull

In contrast to NDN PID push that is focused on Handle end-to-end communication, NDN PID pull focuses on accessing different available copies of a PID (or a Handle record) within a NDN network. Thus, we call this a PID *instance access*, as multiple PID instances can exist simultaneously. To be precise, we use Handle values as an atomic data unit of the Handle system that consist of a typed, indexed key-value pair (cf. Section 2.4 and Figure 2.5). A PID consists of at least one Handle value and hence, Handle value and PID are often used synonymously in the context of the Handle system. Figure 5.14 shows four different scenarios, in which multiple PID instances exist to improve the PID resolution and the retrieval of Handle values. In Scenario ❶, the PID request is served directly from the primary LHS. This is the regular scenario for PID retrieval and can be compared to the PID resolution using the native Handle protocol and our NDN PID push approach. In addition, Scenario ❷ shows a benefit of NDN, where a broken network connectivity of the primary LHS is circumvented by an automatic routing adaption towards the secondary LHS serving the same PIDs. By this, failover mechanism, an instance of the PID is sent to the Handle client, while the original PID is on the primary LHS, disconnected from the network.

Scenarios ❸ and ❹ show that the NDN network is able to respond PID and Handle value access requests from the Content Stores (CS) integrated in the NDN network. This can be a remote CS, located within the depth of the NDN network or even the local CS that are near to the Handle client. Serving requests from the CS allows a fast responding to frequently requested PIDs without contacting the LHS. Thus, the LHS can cope better with high load situations due to network cache-based demand offloading.
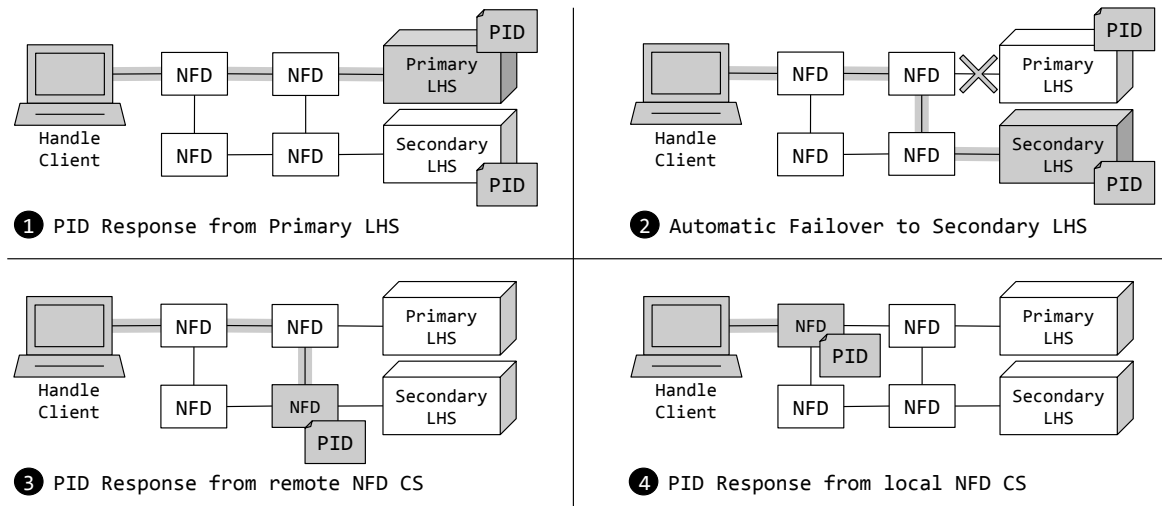

Figure 5.14: PID Response Scenarios in NDN PID Pull

As we can see from the scenarios depicted in Figure 5.14, the network location and the origin of the PID are not important, as long as a robust and fast PID resolution and access to administrative and structural data of a Handle prefix and its associated infrastructure is possible through the NDN nodes reachable by the Handle client. For this, NDN PID pull exploits the design of NDN's network design principles that imply following features:

Content Retrieval

The retrieval of Handle values and thus PID resolution targets is based on the standard NDN data retrieval mechanisms. Each request is encoded as an interest and the Handle value is returned as a NDN data packet Hence, the PID data is *pulled* from a data source out of the NDN network.

Read-Only Access

NDN PID pull is able to access original Handle records and copies of the Handle record on the NDN network. A distinction between copies and original Handle records cannot be done, as the Handle records are by default delivered from the fastest available NDN data source. As a result, NDN PID push can be used for durable writing operations on a LHS, while NDN PID pull cannot provide direct access to a LHS. Hence, the NDN PID pull approach is designed for read-only access to Handle records.

Content-based Verification
As instances of PIDs are accessed through NDN PID pull, the network location cannot be used for assuring the integrity and origin of PIDs. Thus, a content-based verification has to be integrated into the NDN PID pull approach that allows a verification of the content within the NDN data packets.

Improved Resilience
As different instances of a PID can exist within a NDN network, network problems have less impact on the resolution of PIDs and the access of Handle records. If the primary LHS is not available, the requests can be served by the CS of an intermediate NDN network node or routed to the secondary LHS (cf. Figure 5.14) By this, the PID resolution provides an improved resilience against network outages. In the evaluation Section 5.5.4 we show, that the transport of PID data is more resilient in fault network conditions that suffer under packet loss.

After introducing the basic elements of our NDN PID pull approach, we define it for further use in this thesis as follows:

Definition 5.2 (NDN PID Pull) *NDN PID pull is defined as communication pattern between a NDN node representing the Handle PID client and the PID-related data sources in a NDN network. With NDN PID pull, instances of Handle values and thus PIDs can be retrieved from the network. Hence, it is suitable for PID resolution and obtaining structural and administrative information for a specific Handle prefix. For serving the data, different data sources within the NDN network are used that are not selectable by the client and may not be the original data source. Those data sources hold copies of Handle records (instances) for fulfilling incoming retrieval requests autonomously. Hence, data needs to be verified through its content and only guarantees its up-to-dateness for a limited time range. As a consequence of instance access, NDN PID pull merely provides read-only capabilities on Handle values and PIDs.*

We now explain the principles behind NDN PID pull as a NDN-based application protocol that is free of legacy assets and is able to exploit the benefits of NDN technology. It co-exists with the NDN PID push stack. The client can select the protocol stacks based on the semantics of the following actions. Actions that only require PID-related information and do not rely on the latest update of the PID, NDN PID pull is a suitable choice. Actions that require the current state of Handle value / PID from the primary LHS, have to use NDN PID push. The same protocol choice is needed for performing changes to a LHS and its data.
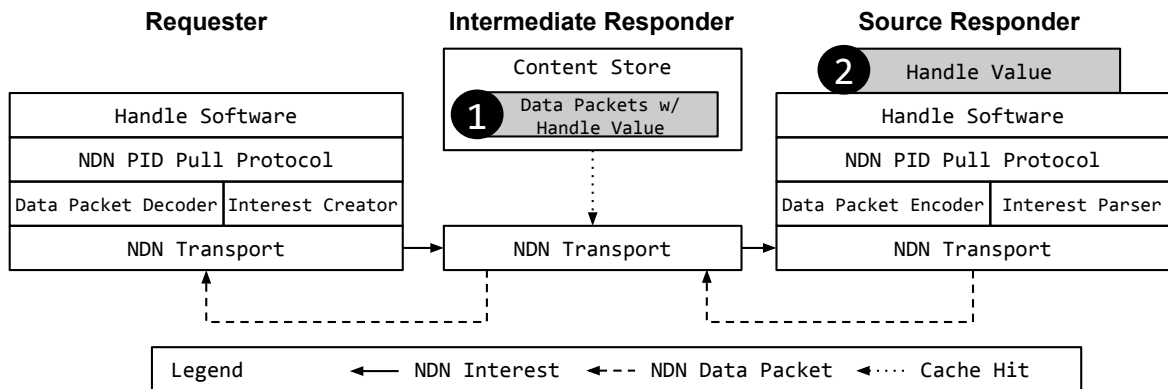
Figure 5.15: NDN PID Pull Transport

In Figure 5.15, we give an overview of the NDN transport mechanism. For issuing a Handle value request, a NDN interest is formulated by the *Requester*. Within the data name of the interest, the full Handle identifier (prefix, suffix and an optional set of parameters) is encoded. This is done by the *Interest Creator*. The interest is transferred over the NDN network from one intermediate node to the other. If an intermediate node has already processed the same request recently, it is able to fulfill the request using the data in its CS ❶. It is then acting as an *Intermediate Responder*. Alternatively, if there is no cache hit at the intermediate node, the NDN network forwards the interest to another potential data source. This could be either the primary LHS or a mirror holding a copy of PIDs from a specific prefix (*Source Responder*). When the interest reaches a valid data source running a Handle software, the interest is parsed and the Handle values ❷ are retrieved from the Handle software and encoded into a data packet by the *Data packet Encoder*. The responder sends the requested Handle values back as NDN data packets. At the requester side, the data packets are decoded and the Handle values are available for further processing.

For the NDN PID pull approach, it is important to wrap the requests into a NDN data name that reflects the shortest routable name of the PID resources in the network (cf. Section 5.3.1, *Namespace Convergence*). For NDN namespace convergence, the Handle prefix has the function of a NDN naming prefix. By this, a PID and its associated Handle values become a named resource in the NDN network. Then, the Handle prefix can become part of the FIB and all interests containing NDN PID pull requests are forwarded to a Handle server data source (cf. Figure 5.16, left gray FIB table). The availability of the Handle prefix can be communicated into the NDN network with auxiliary techniques like NLSR (cf. Section 2.7.6).

Beside the fundamental naming principles, optional query parameters indicated by a question mark "?" can be attached to a NDN name. Query parameter allow selecting and narrowing of Handle values scopes, e.g. selecting a specific value using an index or a set of values selected by the Handle data type. The full data names including the query parameters are stored in the PIT, as they describe the content of the response data (cf. Figure 5.16, right

white PIT table). PIT entries are used for routing pack data packets through the network.

Forward Information Base (FIB)

Name	Face
/prefix_1	0
/other_prefix_1	1
/other_prefix_2	2
/other_prefix_3	3

Pending Interest Table (PIT)

Name	Face
/prefix_1/suffix_a/?index=1	0
/prefix_1/suffix_a/?type=URL	0
/prefix_1/suffix_c	0
/prefix_1/suffix_d	0

```
                        Prefix 1

        Suffix A     Suffix B    Suffix C    Suffix D

   Handle     Handle      Handle      Handle      Handle
   Value 1    Value 2     Value 3     Value 1     Value 1
```
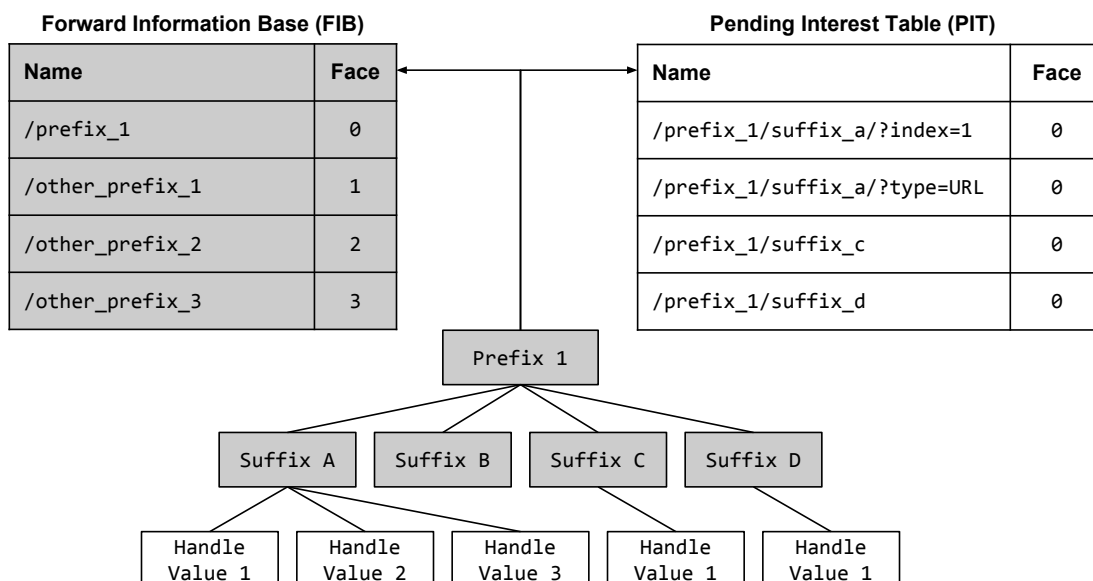
Figure 5.16: Storing NDN PID Pull Data Names in the NFD Data Structures

After determining the advantageous NDN naming schemes for Handle resources and describing the announcement for resources in the NDN network following routing constraints, we can formulate the request query scheme for the NDN PID pull interests. In Figure 5.17, the interest scheme is depicted. Similar to our NDN PID push approach, the entire request information is stored within the interest data name. However, no complex request name encoding is needed, as requests for Handle values are formulated using the Handle/NDN prefix (Global Routable NDN Prefix), the Handle suffix and an optional set of parameters. To avoid clashes between the data name and NDN control commands, the data names need to be URI encoded, according to the NDN naming conventions [63]. Parameters for changing or narrowing the scope of selected Handle values returned by the query can be stated as key-value pairs and concatenated using the parameter "&". The evaluation of the parameters and the execution of selection queries is in the responsibility of the Handle implementation and hence not part of our approach description.

```
                    NDN Interest

 /     11858      /      ABCDE     /     <PARAMS>

        Global              Specific            Optional
  Routable NDN prefix     NDN Name Part       NDN Name Parts
    (Handle Prefix)     (Handle Suffix)  (Selection Parameters)
```
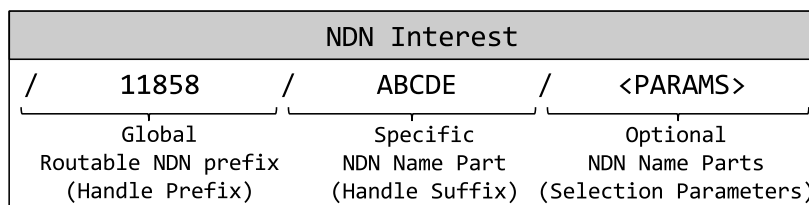
Figure 5.17: Handle Value Request Using NDN Interests

As shown in Figure 5.18, the scheme of a NDN data packet is featured that carries a NDN PID pull response data. The name area ❶ holds the complete data name including all query parameters. This is needed for clearing the PITs of all intermediate NFDs that were involved in the interest transport. For content verification, a signature can be added in the signature area ❷. For checking the content signature, a link to the public key or the cryptographic certificate might be added using the NDN data name in the signed info area ❸. The responses for NDN PID pull are transported in the data section ❹ of the NDN data packet. In this part, the Handle values are transported as payload.

```
┌────────────────────────────────────────────────────┐
│                  NDN Data Packet                    │
├──────────────────────────────────────────────────────┤
│ ❶    Data Name: / 11858 / ABCDE / <PARAMS>          │
├──────────────────────────────────────────────────────┤
│ ❷                  Signature                         │
├──────────────────────────────────────────────────────┤
│ ❸                  Signed Info                       │
├──────────────────────────────────────────────────────┤
│ ❹               Data: Handle Values                  │
│      ┌─ <index>        : 3 ─────────────────────┐    │
│    ┌─ <index>        : 2 ───────────────────────┐    │
│  ┌─────────────────────────────────────────┐    │    │
│  │ <index>       : 1                        │    │    │
│  │                                          │    │    │
│  │ <type>        : URL                      │    │    │
│  │ <data>        : https://gwdg.de/...      │    │    │
│  │ <TTL>         : {Relative: 24 hours}     │    │    │
│  │ <permission> : PUBLIC_READ, ADMIN_WRITE  │────┘    │
│  │ <timestamp>  : 927364617838              │         │
│  └──────────────────────────────────────────┘         │
└────────────────────────────────────────────────────┘
```
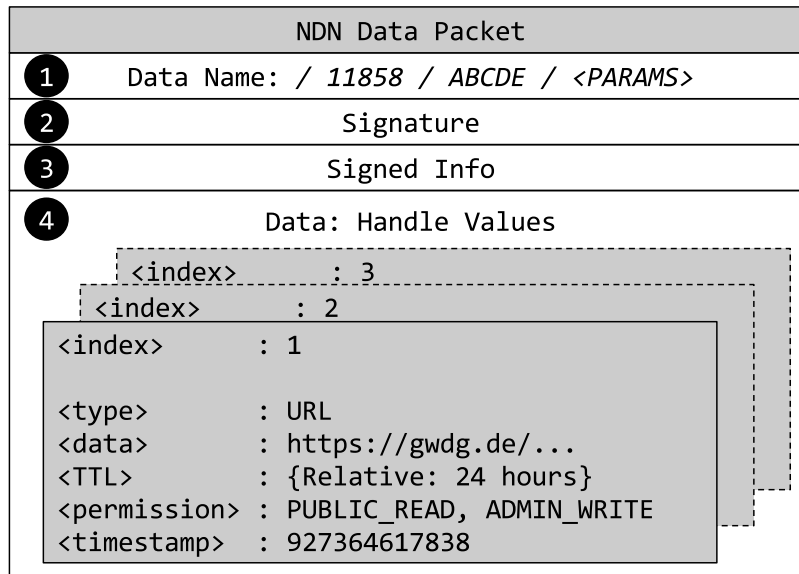
Figure 5.18: Handle Value Response using NDN Data Packets

For formulating a NDN PID pull request we propose a logic shown in Figure 5.19. An *Interest Creator* ❶ wraps the request into an interest as described above. The *NDN Face* ❷ is emitting the *Interest* ❸ into the NDN network. For receiving response data packets, the logic on the lower part of the figure is used. Incoming *Data Packets* ❹ are received by the NDN face and then transported to a *Data Packet Decoder* ❺. The decoder serves two purposes. First, it has to unpack the payload of the data packets and transform them into Handle values. Secondly, if a response is exceeding the maximum payload size of the NDN data packet and thus is fragmented in multiple packets, it has to collect all payloads from further incoming packets related to a specific response. When the sets of collected data packets is complete, they are transformed back into Handle values. For unpacking fragmented packets, the NDN fragmentation principles can be applied, which are subject of current NDN-related research [130] [131].
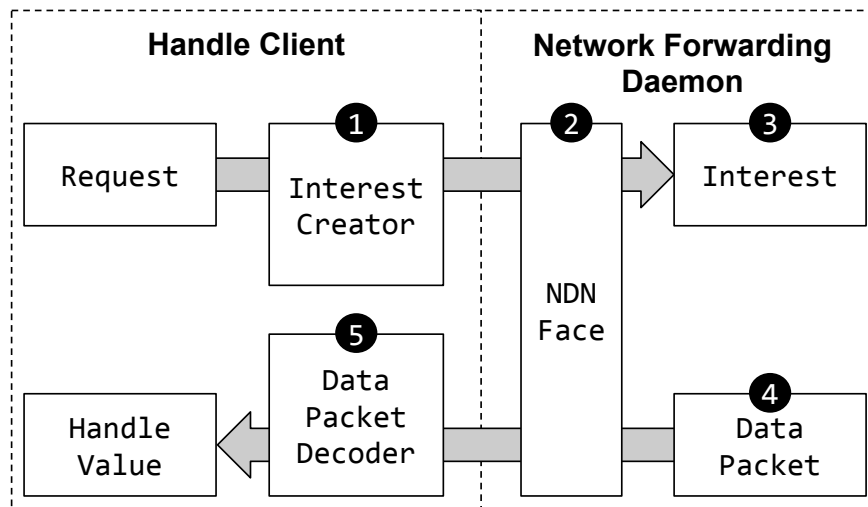
Figure 5.19: NDN PID Pull Request Issuing

On the side of the Handle value data sources request processing logic is required as well. This is described in Figure 5.20. For processing a request, the information on Handle value access is received by the *NDN Face* ❶. Then, the *Cache Manager* ❷ of the NFD is looking up the data name in the CS. If a cache hit occurs, a matching data packet containing a response is sent back to the receiver. In case of a cache hit in the CS, the Handle server is not involved in processing the response at all, as this is done automatically by the NDN software stack outside the Handle server application context. If a cache miss occurs, the interest is forwarded into the Handle server, where the *Interest Parser* ❸ is extracting the request information from the interest. The standard *Handle Library* ❹ can be used for retrieving the Handle value from the database. Handle values from the database are now encapsulated into NDN data packets by the *Packet Encoder* ❺ and then sent to the requester via the *NDN Face*. Retrieving Handle values from the database requires more computational effort than a cache-based responding, but the chain of request parsing, data retrieval, response encoding and transporting works linear and may be implemented parallelized for simultaneously handling requests.
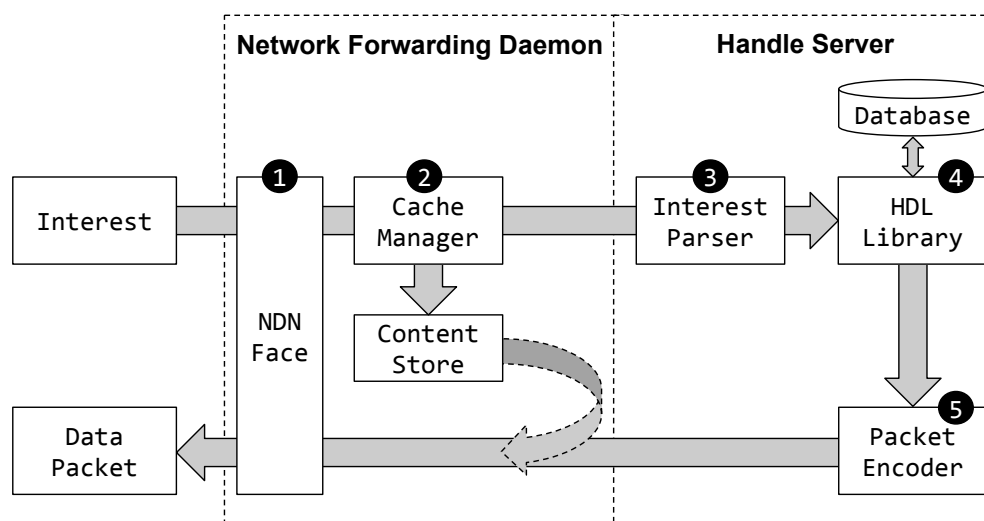
Figure 5.20: NDN PID Pull Request Processing

5.3.4 Interoperability Model

All location-independent efforts presented in this thesis do only provide advantages if they are interoperable and anticipate the slow change momentum of research data management. Thus, a location-independent approach for PID needs a connectivity between the location-dependent classic network and the location-independent NDN-based approach of this thesis. For this, we formulate the concept of dual-connectivity presented in our IEEE paper in 2015 [104]. The interoperability model assumes a *two world scenario*. The first world is the classic location-dependent network world with our current state-of-the-art technology. In this world all official current PID systems are running including all LHS and all sites of the MPA GHR. In the second world the location-independent NDN network proposed in this thesis is operating its experimental local Handle Systems. Network nodes located in this scenario are differentiated into nodes with *single connectivity* either featuring location-dependent or location-independent access and nodes with *dual connectivity* featuring both connection types. For classic single connectivity, we assume the possibility to send and receive UDP and TCP packets and to communicate using the native Handle protocol or HTTP for data exchange. Within the classic connectivity, the native Handle protocol is used for communication. This is necessary to access GHR and LHS over the location-based WAN. Nodes with a single connectivity are connected to a NDN network. Within the NDN network, the Handle nodes can use NDN PID push for a full-featured Handle communication, similar to the native Handle protocol or NDN PID pull for retrieving Handle values and resolving PIDs only. Nodes with dual connectivity have access to both worlds with a location-dependent and -independent access. These nodes can send and receive all three Handle access protocols. Figure 5.21 is depicting the *two world scenario* as a simplified chart. In the upper part, the NDN network is depicted together with

native NDN clients and LHS. The lower part is showing the classic location-based network containing a sample client, all GHR sites and two sample local Handle systems. A *Handle Gateway* with dual connectivity is connecting both worlds.
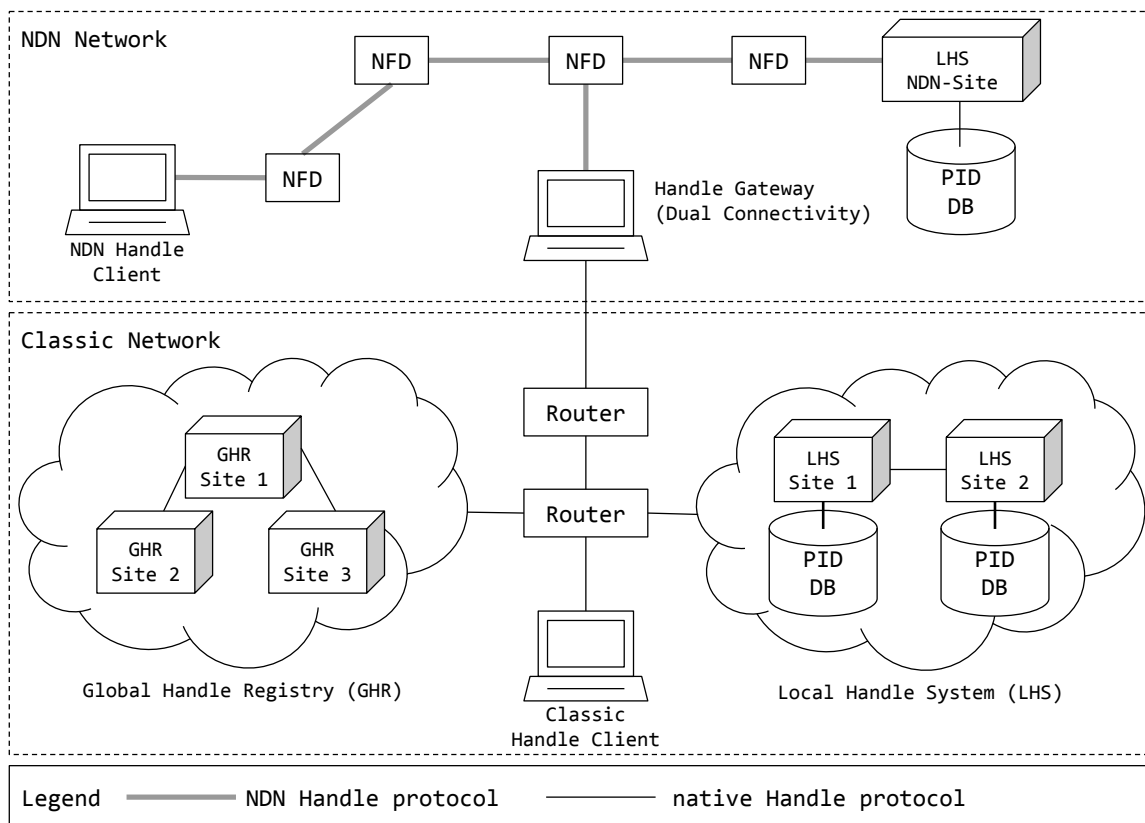

Figure 5.21: NDN Gateway Architecture for Handle Interoperability

5.3.4.1 Incoming Requests

We now have a look at requests coming from the location-dependent classic network going to the NDN network (location-dependent to location-independent inbound traffic conversion). A natural approach for forwarding incoming network traffic is to unpack the Handle messages from UDP/TCP traffic and transport them using our NDN PID push approach. By this, all types of requests can be transported from one network into the other. Although this approach theoretically works, it comes with the drawback of higher latency and reduced reaction time, as we show in Section 5.5.3, when only using NDN PID push as default protocol. Hence, for only suitable Handle requests, the protocol for forwarding has to be changed to NDN PID push, to exploit the benefits of NDN and to accelerate network traffic, whenever it is possible. As a result, the challenge arises of selecting the suitable network

protocol for each request that is either NDN PID pull or NDN PID push for an optimized forwarding of traffic.

In the following, we discuss the protocol selection of the NDN transport. The traffic transition function of the Handle Gateway aims at shifting the data transport from a location-dependent protocol set to a location-independent protocol set. In this case, we have full control over the transported application protocol payload on layer seven of the OSI model for the location-based classic network connection part. Thus, we can inspect all application payloads of the Handle messages on the incoming location-dependent site. As stated in Section 5.3.3.1, the Handle protocol holds transport-specific meta-information in the application protocol layer, mirroring meta-information of the transport protocol layer. The meta-information stored in the Handle message envelope (cf. Figure 5.6) contains information for a request classification needed for the protocol selection. Thus, a set of rules for Handle message inspection can be generated that allow an appropriate protocol selection using a Protocol State Machine (PSM) [132].

Figure 5.22 shows the PSM for selecting the appropriate protocol for Handle request forwarding within the NDN network as an UML state machine. The state machine represents the algorithm of forwarding protocol selection of incoming location-dependent traffic. It does not contain any improvements for faster protocol selection based on partial Handle message evaluation but rather presents the approach with full Handle message evaluation in order to show the fundamental principle of our approach. The selection of protocols is based on following assumptions:

1. **Handle Message Evaluation**
   Decision Input for protocol selection is derived from the Handle Messages and its data structures. Thus, relevant data structures in the Handle messages may be spread over multiple location-based data packets and hence we need to reconstruct Handle messages first.

2. **Fast Forwarding**
   If a protocol selection has been done using the first Handle message of a request, all Handle messages belonging to the same request context can use this decision. As a result, a fast forwarding of Handle messages is possible. This is possible, as all Handle messages that are related to a request, share the same request identifier (`ReqId`) in each Handle message.

3. **Handle Protocol State Awareness**
   For requests requiring end-to-end connection setups (e.g., PID creation or update), NDN PID push is the only valid protocol selection in our approach, as determined in Definition 5.1. Through observing the state flags in the Handle messages headers, we can determine whether a Handle message belongs to a state aware end-to-end connection setup. Interesting state flags are the `SessionID` indicating the context of a client-server session or the `EC` message flag showing the context of an encrypted end-to-end connection. If a specific set of these state flags is set in the Handle message, the protocol selection can be done towards NDN PID push.

4. **Envelope-to-Header Evaluation**

   Our approach presented for Handle message evaluation could perform an envelope-to-header evaluation (cf. Handle message scheme, Figure 5.5). For this, we would first evaluate the envelope of the incoming Handle message and afterwards its header. In location-dependent networks, the envelope are sent first, as packet fragmentation algorithms start with message sending at the lowest bit index. With a partial evaluation approach, an improved version of the PSM could be implemented that takes decisions using partial Handle messages, by evaluating the first bits of the Handle message to make early decisions on the incoming Handle messages. However, we leave this optimization for future work and just point out here that our approach offers this possibility of optimization.

5. **Parallel Processing**

   The approach of PSM allows the parallel processing of data flows based on the connections. As an intermediate node, the Handle Gateway can process packets and Handle messages for each client connection in parallel. This allows using parallel processing capabilities of modern computer systems to increase the throughput.

After stating the assumptions on which selection approach is based, we explain the decision process in detail. Figure 5.22 depicts the PSM that is used as foundation that materializes the selection algorithm. For modeling the PSM, a UML state machine diagram is used [133]. The decision flow is from top to bottom. In the initial state, data packets arrive at the incoming location-dependent network interface of the Handle Gateway. The data packets originating from a UDP or TCP connection are first collected in a buffer, in order to reconstruct the Handle message. The reconstruction for location-based data packets is depicted in the upper gray box ❶ (*Message Reconstruction*). After the Handle message has been reconstructed, a look up at the *decision state table* is made using the request identifier for the Handle message (`ReqId`). In the decision state table, former protocol decisions for Handle messages in the same request context are stored. If an entry for the Handle message's request identifier is found in the table, the protocol decision is immediately applied, as further evaluation would lead to the same decision result due to the inner protocol state of the Handle protocols. Additionally, the Handle protocol features a fragmentation mechanism, called truncation for Handle messages [38], in order to optimize packet flows in the location-based network. If there is no match in the decision state table, further evaluation of the Handle message is needed. For this, the fragmented Handle messages need be reconstructed, in order to evaluate Handle messages on the application protocol layer. This is depicted in the lower gray box ❷ (*Reversing Message Truncation*). After reversing the message truncation, the message envelopes are checked using low-numbered index bits. If the message is encrypted (`EC` bit set) or a session identifier is contained in the envelope, the message is part of an end-to-end connection and requires NDN PID push as protocol for transport. Finally, the message header is inspected. All requests that retrieve information from Handle data sources and that do not imply a state change on the Handle data source side or even the original LHS are suitable for NDN PID push. In Table 5.1, an overview of Handle OpCodes is given. All OpCodes retrieving information from a Handle data source,

without altering the internal state of the data source and thus suitable for NDN PID pull, are marked with an asterisk (∗) in the table. Following Table 5.1, we can see that the OpCodes for Handle resolution (`OC_RESOLUTION`) and Handle enumeration (`OC_LIST_HANDLES`) are included for NDN PID pull resolution. Handle resolution and Handle value listing form the vast majority of requests that the Handle PID system is facing. Thus, most of the Handle message forwarding protocol decisions are made for NDN PID pull and as we show in Figure 5.42, message forwarding will be accelerated as much as possible in the NDN space. As a result, forwarding incoming PID traffic from the location-dependent to the location-independent space is fast, efficient and reduces overhead. All protocol decisions are stored in the decision state table if the Handle message contains a request identifier with an expiration time stamp for table clean up. The Handle message is then either forwarded with NDN PID push or the request is rewritten into a NDN PID pull request that is sent to the LHS via NDN.
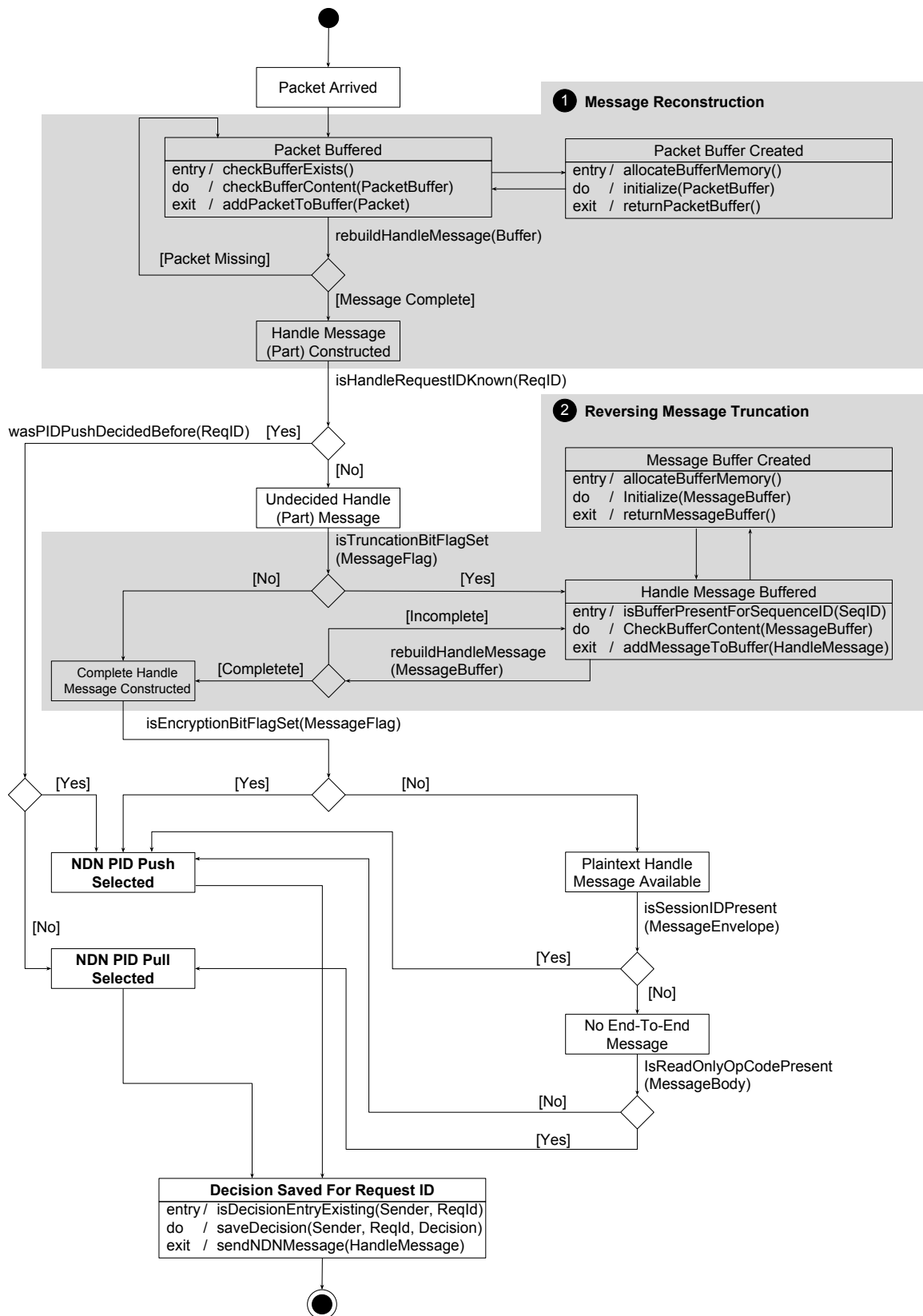
Figure 5.22: UML State Machine for Protocol Selection of Incoming Gateway Traffic

5.3.4.2 Outgoing Requests

Outgoing requests are sent from Handle clients within the NDN network to LHS and the GHR in the location-dependent classic network. For this, the incoming requests wrapped in different NDN PID protocols have to be repacked into the Handle application protocol. To perform a correct routing between the inner peers located within the NDN network and the outer Handle data sources in the classic network, the maintenance of a stateful routing is needed. In contrast to incoming requests, where the protocol decision for forwarding is done centrally at the Handle Gateway, the clients located in the NDN network individually decide on the protocol usage based on the request they issue. Hence, only a static routing and forwarding mechanism is needed for outgoing requests that allows applying an algorithm with reduced complexity.

For outgoing requests, using NDN PID push, the Handle messages encoded in the interests need to be transferred into the Handle messages transported by UDP or TCP. For the client located inside the NDN network, the Handle Gateway remains hidden, when using the combination of Handle prefix and suffix. In the case for all prefixes that are registered for gateway-based routing, the requests are forwarded in the NDN network using the Handle prefix as NDN prefix. For routing the information from the Handle Gateway to the Handle data sources within the location-based network, the Handle prefix is resolved into the IP-address of the data source using the resolution information from the GHR. As the transmission unit of the NDN PID push protocol contains the data name of the sending NDN client (cf. Figure 5.11), the routing information can be completed for Handle message forwarding. In Figure 5.23, three different NDN naming scenarios are depicted for using the Handle Gateway from the NDN network employing NDN PID push. In scenario ❶, the Handle Gateway is used in the same way as any other LHS in NDN employing the Handle/NDN prefix as globally routable name (cf. Figure 5.9). When using the NDN prefix in this way, the Handle Gateway acts transparently for the NDN Handle client and serves forwarded requests and responses in the same way as other LHS in the NDN space. In scenario ❷, the Handle Gateway uses an explicit NDN name for forwarding requests. In this case, the usage of the gateway is visible to the client by explicitly using the `gateway_name` in front of the NDN PID push interest. Scenario ❸ shows the explicit selection of a Handle server in the location-dependent network space, using an IP-address.
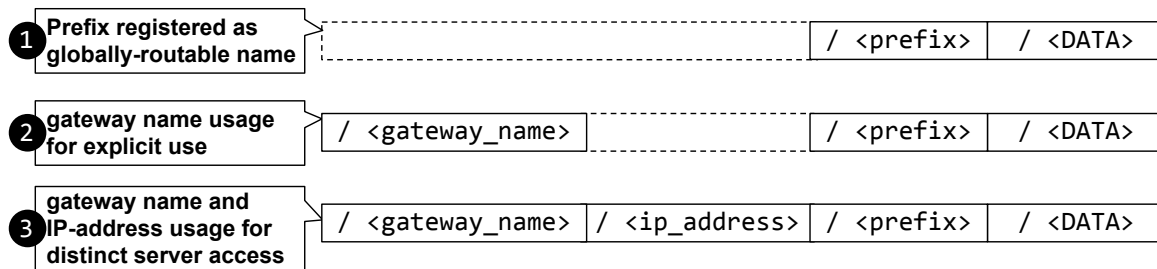
Figure 5.23: NDN Naming Scenarios For Outbound Handle Gateway Usage With NDN PID Push

After addressing the Handle Gateway in different scenarios, using different data names, we continue with the forwarding process of NDN PID push data from the NDN space to the location-dependent space. NDN PID push and the native location-dependent Handle protocol are based on the paradigm of sending Handle messages as segmented payload in transmission units (cf. Figure 5.11) or data packets. As a result, a full reconstruction of Handle messages is not needed before forwarding the request. It is sufficient to determine the location of the data source within the location-dependent network and then forward the payload of the NDN PID push transmission unit in a TCP or UDP data packet. By this, the Handle Gateway and the data source in the location-based network have established a native Handle protocol connection and exchange data to the NDN client using NDN PID push. With a direct forwarding of packet payloads, all requests and response information can be streamed forward and backward between the network areas.

The payload forwarding from NDN PID push to location-based native Handle protocol is depicted as a UML PSM in Figure 5.24. The core of the forwarding process boils down to three major steps:

1. **Determining Target Network Location**
   With the extracted Handle prefix from the NDN PID pull transmission unit data name in the NDN interest, the network location of the LHS needs to be determined. For this, the Handle Gateway queries the GHR to obtain the IP-address of the forwarding target which could be a LHS, a LHS mirror or one of the GHR servers.

2. **Manage Connections & Routing Information**
   After obtaining the network location of the forwarding Handle target, a connection has to be setup using the native Handle protocol. Additionally, the routing information, which consists of a triple of sender NDN name, receiver NDN name and the IP-address, has to be stored in the *Routing Info Table*, to reuse forwarding target location information of all incoming transmission units and data packets.

3. **Forward Payloads**
   The payloads containing the Handle message fragments need to be encoded and decoded into NDN interests, as described in the Section 5.3.3.1.
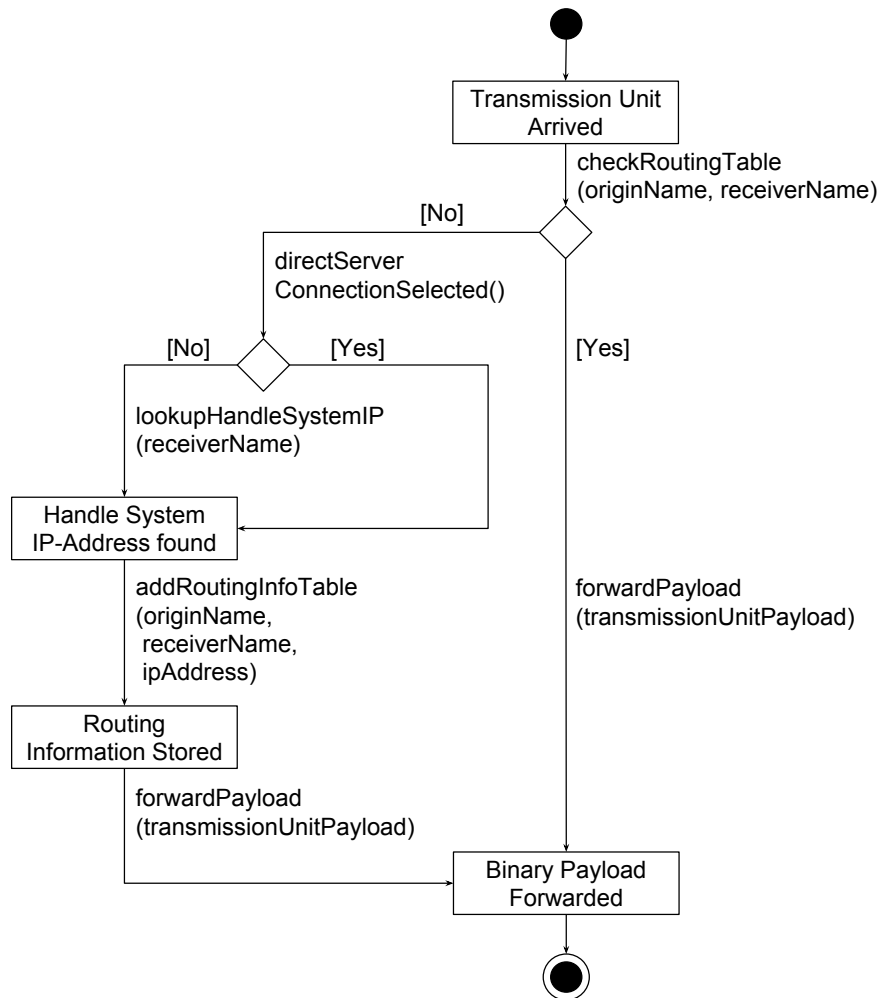
Figure 5.24: UML State Machine for Forwarding Outgoing Binary Payload Delivered Through NDN PID Push To Location-Dependent Handle Systems

We have now a look at forwarding NDN PID push based requests from the NDN network to a location-dependent classic network and the response back in the opposite direction. For each NDN PID pull request, wrapped into a NDN interest, an outbound connection to a Handle data source (LHS or GHR) is established and the data packets are forwarded using the native Handle protocol. If a response is sent back over the established connection to the Handle Gateway, it is wrapped into a NDN PID pull data packet (cf. Figure 5.18). Similar to the previously proposed approach of NDN PID push forwarding, a determination of the network location of the forwarding target in the location-dependent network is needed to create a routing info table.

For using the Handle Gateway in conjunction with the NDN PID pull protocol, different naming scenarios are available to the NDN client, depicted in Figure 5.25. In contrast to the usage of the Handle Gateway through NDN PID push, it provides a slight modification (annotated as gray boxes) concerning the last part of the data name, which contains the PID suffix. It is needed, as NDN PID pull requires the full Handle PID including the prefix *and* the suffix for a globally-routable NDN data access of a PID (cf. Figure 5.17).
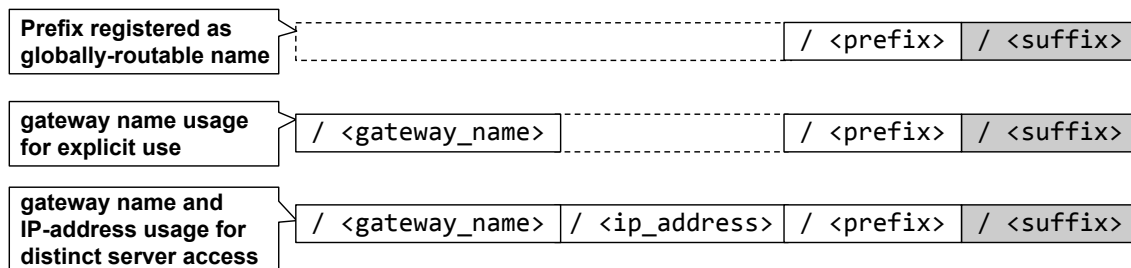
Figure 5.25: NDN Naming Scenarios For Outbound Handle Gateway Usage With NDN PID Pull

Figure 5.26 depicts a UML PSM for NDN PID pull request forwarding that is similar to Figure 5.24. The acquisition and management of the forwarding information in the *Routing Info Table* is identical. However, the request formulated in the NDN data name of the interest needs to be forwarded as a native Handle message. For this, the interest has to be reformulated as a Handle request. As a result, a streaming of request information into Handle messages is not directly possible. The request information has to be stored and forwarded as a complete request using the native location-dependent Handle protocol. On the one hand, this degrades the Handle Gateway to a store-and-forward relay, with decreased forwarding throughput, but on the other hand, NDN PID pull messages are very compact due to their limitation for read-only information retrieval. Thus, only one interest and a low number of data packets (e.g., one to five, cf. Section 5.42) is required for information forwarding. Hence, a store-and-forward architecture provides almost no disadvantage for NDN PID pull forwarding in comparison to a payload streaming approach.
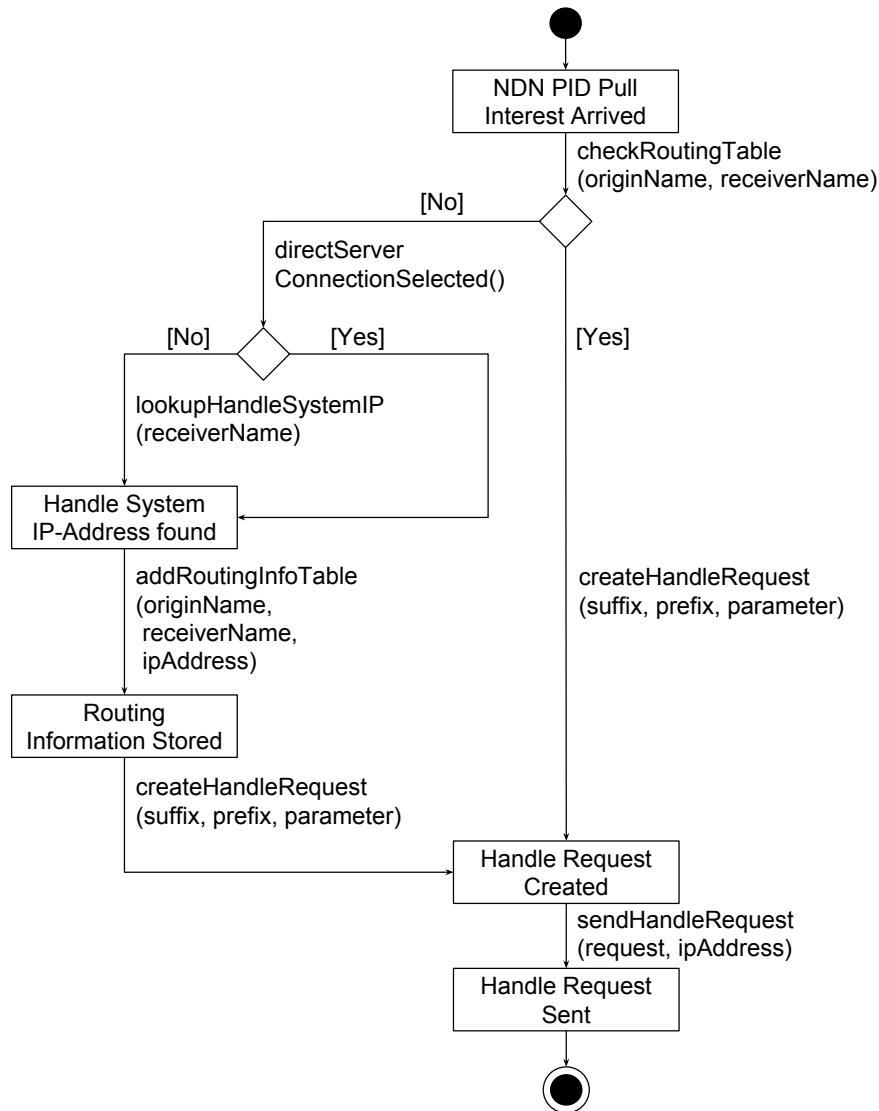
Figure 5.26: UML State Machine for Forwarding Outgoing Binary Payload Delivered Through NDN PID Pull To Location-Dependent Handle Systems

5.4 Implementation

In order to demonstrate the feasibility of a location-independent PID system that is based on NDN, a proof-of-concept implementation is outlined in this section. The implementation uses Java 1.7 for realizing and integrating all user land tools of the NDN and Handle software stack. The NDN components of the network stack are realized in C++ by the NDN research community and serve as a foundation for executing NDN network communication. Major software components are derived from our software stack presented in the IEEE paper

from 2015 [104]. In contrast to the paper, the software components have been updated to more recent versions in order to anticipate the progress of the NDN research project and also to incorporate new techniques in the Handle System. As a foundation, the source code base of Handle Server Version 8.1.1 from CNRI [134] is used in conjunction with the Java NDN application bindings jNDN 0.12 [135]. For operating the NDN network, unmodified versions of the NDN network stack are used in conjunction with a NLSR daemon for NDN data name prefix propagation. NFD in version 0.4.1 is realizing the NDN node with all necessary structures like PIT, FIB, CS and application faces [136]. Name management and name prefix propagation in our NDN network is done by NLSR in version 0.2.2 [137].

5.4.1 NDN-Enabled Handle Server

The Handle Server is responsible for storing and managing the persistent identifiers. It has been designed by CNRI as a modular architecture with defined subsystems that are built for fast and concurrent data access. The Handle Server incorporates the Handle Client library that is responsible for interacting with any existing Handle System using TCP, UDP or HTTP for communication [138]. Besides the client libraries, the Handle server features subsystems for offering a HTTP-based Representational State Transfer (REST) interface based on the Java web container Jetty, a subsystem for database access, and numerous small components responsible for cryptography. To implement the NDN functionality into the Handle server, software parts that are responsible for PID persistency in databases remain untouched, as well as native Handle protocol parts. Hence, instead of changing the technical architecture of Handle, the NDN part is implemented as an addition to the Handle system. In order to provide a *complete* port of the Handle protocol to NDN and thus to allow all operations to a PID server in NDN, even the complex ones for PID maintenance tasks, a split approach is needed. This approach is split into the NDN PID push part and the NDN PID pull part. All implementations are done to the Handle client library that existing software can make use of NDN without major modification using NDN PID push communication. However, to make advantage of NDN augmented PID resolution, an extension of the Handle library is provided with NDN PID pull.
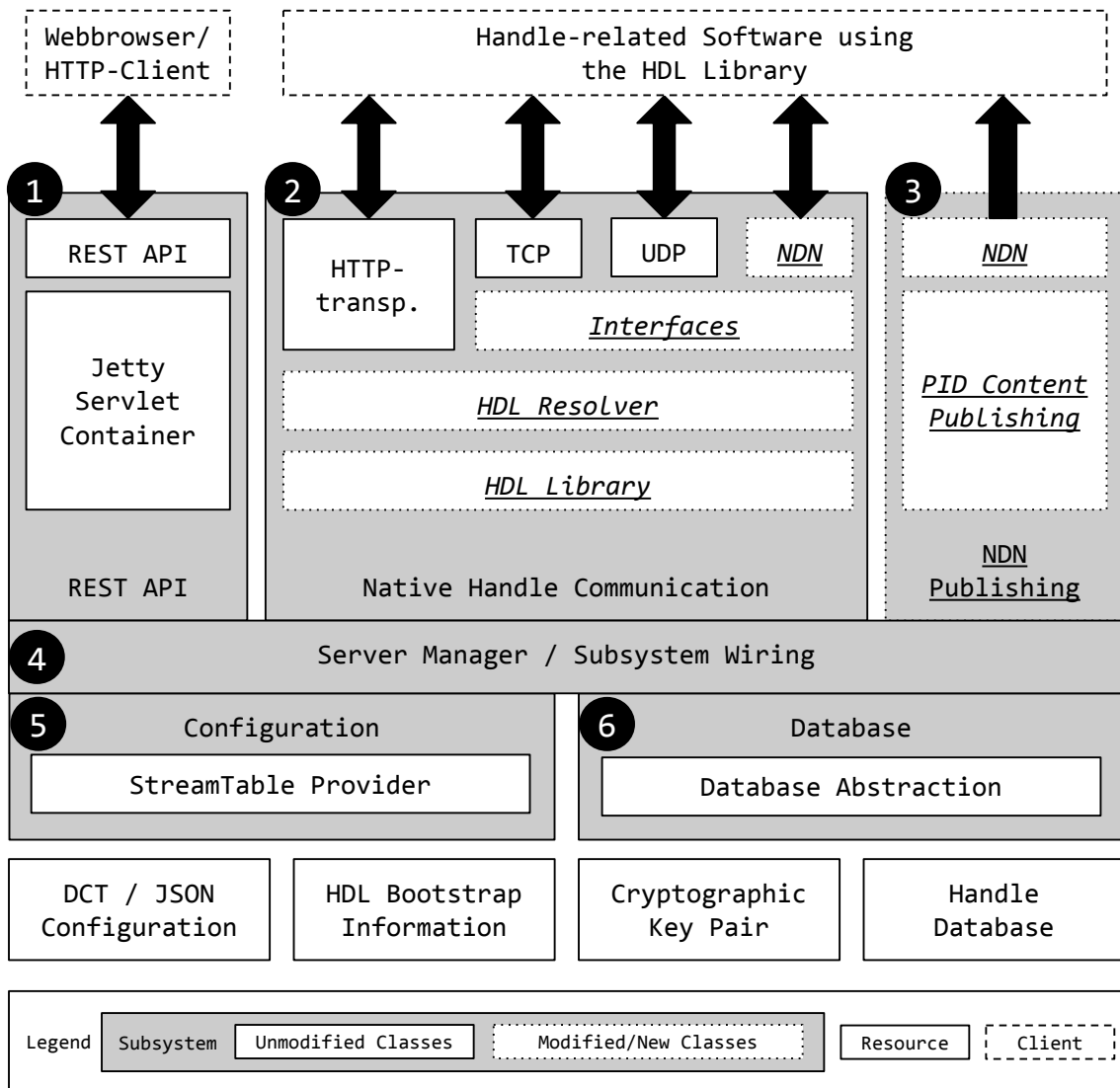
Figure 5.27: Overview of the Handle Server Software Architecture with NDN Additions

We now give a brief explanation of the Handle Server software architecture. An overview that only contains major subsystems and systems that are needed for an understanding of the NDN integration into the Handle System is provided in Figure 5.27. The subsystem for accessing the Handle system as a regular REST-enabled web service is depicted in ❶. The REST-API is a recent development and allows interaction with Handle servers from third-party software [138]. The REST subsystem contains an own Jetty servlet container that allows executing the Handle service as a standalone web service. This subsystem should not be mixed up with the Handle proxy server, which is a separate software product using parts of the server code base. In contrast to the REST-API, the Handle proxy server is only able to read PIDs and Handle values, while the REST-API allows

full administrative and write access. Subsystem ❷ contains all parts for native Handle protocol communication. It is responsible for serving incoming requests from Handle clients and other Handle servers. It implements Handle's native authentication, session management, transport encryption and an own system of data queues for message flow control. This subsystem is in most parts very similar to the software components that can be found in Handle client software. The core component of this subsystem is the HDL library for Java. All additions and modifications of the HDL library are explained in Subsection 5.4.2. Subsystem ❷ features an own resolver service that is used to query other servers and the Handle root infrastructure. As clients feature different kinds of connectivity, the subsystem provides a very aggressive and redundant connectivity, in order to serve request on the fastest connection available. The primary communication is done through an *interface* abstraction that provides communication either through UDP or TCP. Those interfaces connect to the operating system network interfaces through network sockets provided by the Java Virtual Machine (VM). As a fall back communication, the Handle subsystem can also to communicate via HTTP with other Handle servers. This allows running a Handle server behind a restricted firewall through a HTTP-proxy server. Subsystem ❸ is responsible for accessing Handle values and thus also to resolve Handle PIDs using NDN. It implements the NDN PID pull functionality. Subsection 5.4.4 explains the subsystem in detail. The functionality for starting, stopping and initializing the Handle server is provided in ❹. This subsystem is also responsible for daemon threads that perform housekeeping tasks. The subsystem for configuration management ❺ provides all configurations parameters for the other subsystems. These configuration items are stored in key-value files encoded as DCT or JavaScript Object Notation (JSON). The configuration contains the IP-addresses and ports for the UDP and TCP interfaces, the role that the Handle server possesses (primary/secondary site), the database configuration and all homed prefixes. The subsystem also provides access to the bootstrap information of the Handle System and to the cryptographic key pair of the server. Subsystem ❻ provides database access to read and write Handle values. It features a transaction management all operations and provides an abstraction layer to use a Java Berkley Database or SQL-like databases like MySQL or Postgres SQL.

5.4.2 Handle Library Modification for NDN Connectivity

In this subsection, we take a closer look at the native Handle communication subsystem, in order to give an insight into the necessary extensions and changes that were made to the Handle server. While Figure 5.27 gives a full software architecture overview, Figure 5.28 provides a detailed view of the native Handle communication subsystem. This subsystem is responsible for all inbound and outbound network traffic of the Handle server and it is capable of handling arbitrary Handle commands in the native Handle protocol specified in RFC 3652 [38].
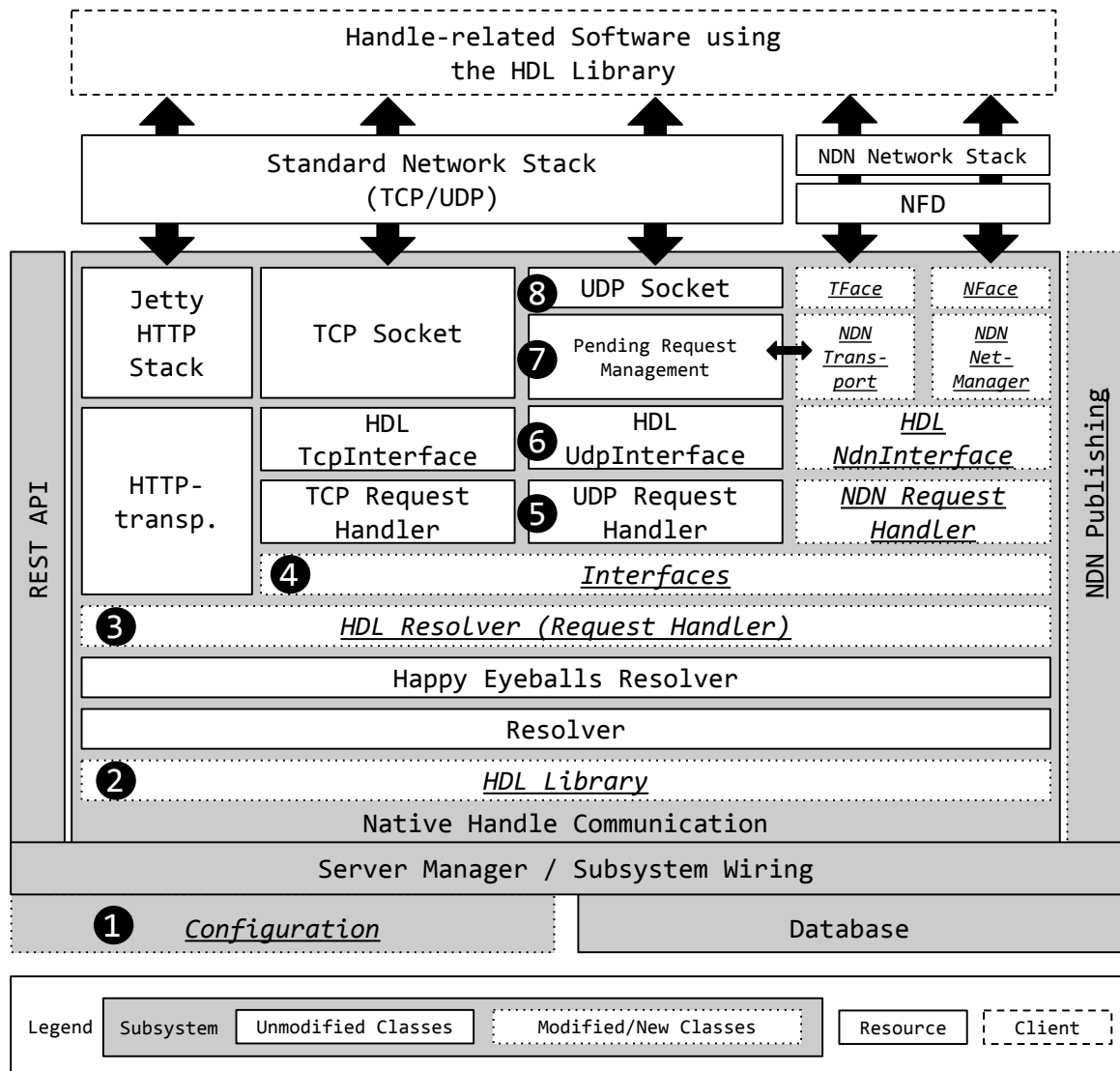
Figure 5.28: Details of the NDN-enabled Native Communication Subsystem

For a better understanding of the native Handle protocol port to NDN, we first explain how UDP request handling in the Handle server is realized. As we will see in the following, the handling of Handle protocol messages transported via UDP has shared principles with NDN PID push that uses NDN interests for data transport. For incoming connections, the Handle server acquires an UDP socket that listens per default on port 2641 ❽. The socket receives all incoming UDP datagrams that contain binary Handle data. As the size of UDP packets is limited and Handle messages may exceed the size limitation of a UDP datagram, it can be split into several packets. Thus, the Handle server needs to keep track of incoming packets and has to assign incoming packets to specific Handle messages and requests. This is done by a pending request management ❼ using the `request id` of the Handle Message and IP-addresses of incoming datagrams. It is also important that, in contrast to TCP, UDP

datagrams are not available in a specific order to the application using the socket [139]. Thus, ❼ also needs to sort all incoming datagrams into a valid order such that it forms a valid Handle request for the application. Moreover, the server may simultaneously have contact to multiple clients and therefore, ❼ also has to attribute datagrams to specific clients. This is done by handling the datagrams using the sender IP address encoded in the packet. If the Handle request is fully reconstructed from a fragmented Handle message, the request is forwarded by the HDLUdpInterface ❻ to the UDP Request Handler ❺. The UDP request Handler is now passing the request to the server logic of the Handle Server that is contained in the HDL library ❷. It is also responsible for handling the generated Handle response. The Handle response that contains the answer for the client is decomposed into UDP datagrams and then sent back to the client using the IP-address and the `request id`. The UDP data grams are sent back to the client using the UDP socket ❽.

In the following, we have a look at the modifications and extensions that have been made to the Handle server to provide a NDN-based connectivity. Afterwards, in the next subsubsection, we describe the implementation of PID push that forms NDN interface layer ❻. The Goal of the NDN native protocol implementation is to provide an additional connectivity method for the Handle System that is equal to the existing location-based connection methods. By this, we enable a full dual connectivity of Handle servers between location-based and location-independent networking. We realize this by providing new NDN request handlers that are responsible for processing incoming client requests and a NDN interface abstraction that provides data transport and client-server connection negotiation and establishment. The first component that needs adaption is the configuration subsystem ❶ that is extended to provide NDN-related configuration items. These configuration items are NFD access data, the global NDN name of the server and the possibility to configure the NDN interface. Next, the HDL library ❷ needs a small modification to provide NDN network connectivity understanding. For this, a new connection type `HDL_SP_NDN` is added to the HDL Resolver ❷. The Interface abstraction ❹ also needs an understanding of the newly integrated interface types. They provide an instance of the NDN interface for all connections that require `HDL_SP_NDN` as protocol for their connection. The effort for changing the Handle server up to now is very minimalistic, as only the native communication subsystem has been extended so far. In Appendix A.2.1, we provide the complete patch set of the existing Handle server code base. It underlines our minimalistic changes to the existing code base that are necessary to integrate the new NDN connectivity, which is described in Section 5.4.3.

5.4.3 Native Handle Protocol Transport With the NDNInterface

In this section, we describe the implementation of the NDN interface. Our NDN Handle interface implementation is designed to build an end-to-end connection using the NDN PID push principle, meaning that a Handle client can connect and access resources on a Handle

server using a NDN-enabled library. Important for NDN PID push access is the fact that a specific server is accessed independent from its location. In contrast to that PIDs can be access independently from a specific server and a network location using NDN PID pull.

For realizing end-to-end connections in NDN, we have to take into account two facts. First, we want to eliminate NDN round trips in the network transmission to provide fast data transfer. Secondly, we want to use all available routes from the client to the server to ensure optimal packet transport. For the first part, we use a single interest-data pair round trip as minimal NDN node interaction. Thus, we transport data payload from the client to the server in an interest, which is forwarded by all intermediate NDN nodes without further processing or caching. This also allows us to directly initiate a connection from the client to the server. To clear the PITs of intermediate nodes, a response data packet is sent from the server back to the client. This response server packet has an empty body section as we will see later on. For the second part, we have to anticipate the fact that interests and thus also data packets may take different routes from the client to the server. Thus, data payloads encoded in interests may arrive at the other peer in a different order as they were sent out.

Additionally, it is important that the Handle protocol and thus the Handle server implementation heavily rely on IP addresses for managing their connection, attributing incoming and outgoing Handle messages and managing Handle locations. Hence, in order to use the code of the original Handle implementation, we have to find parts of the code base that are able to handle random packet orders and attribute packets based on their origins. The Handle UDP interface implementation is generally providing these features and has an own pending request management. This is helpful, as NDN PID push and UDP-based transport share these main properties. Thus, we follow the approaches from literature that propose a light-weight adaption of TCP for NDN (cf. Subsection 4.5.1) but provide a UDP-like adaption for NDN, in order to transport native Handle messages through NDN. In contrast to literature [112] that is looking at stateful end-to-end connections with TCP, we build a stateless connection in UDP manner for the Handle protocol. However, implementing UDP over NDN is still very complex and to our knowledge, no publications exist in this area. Luckily, a full implementation of UDP is not needed for Handle message protocol over NDN, as the NDN integration for NDN PID push is done on application level. Thus, we focus on the implementation of UDP-like Java API that uses NDN for data transport and end-to-end connection management, in order to satisfy the needs of the Handle library. As a result, we can offer NDN connectivity to the client and the server. And as we have full insight into the Handle protocol specification and the Handle library source code, we possess the necessary knowledge to manage a NDN connection. By this, the implementation of UDP-like API for Handle is possible in contrast to a universal UDP adaption that provides data transport of black box data packets. To start with the implementation, we patch the Pending Request of the UDP Handle connectivity, in order to manage incoming data packets on all Handle interfaces (cf. Figure 5.28, ❼ and Appendix A.2.1). Then, a native NDN Interface is derived from the UDP interface code but instead of Java UDP sockets, a custom NDN implementation is provided that features NDN

data transport and management using NDN PID push (cf. Appendix A.2.2). In Figure 5.29, the details of the HDLInterface for NDN PID push are lined out. The figure is used as overview map for the following explanations and provides more details than Figure 5.28 in terms of client-server Handle protocol communication over NDN.
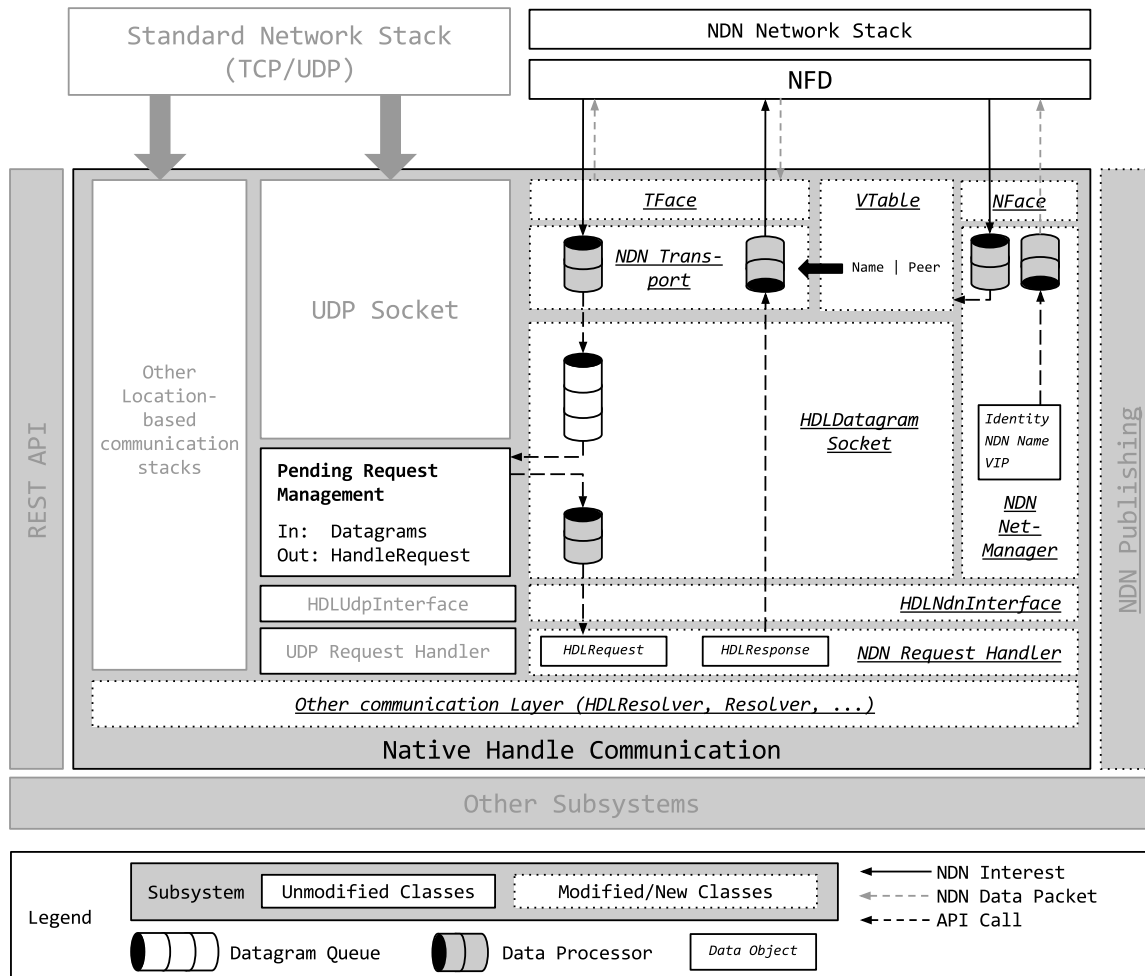

Figure 5.29: Details of the HDLNDNInterface

The Handle Library uses a Java 7 `DatagramSocket` for providing UDP connection [140]. In general, the method `send(DatagramPacket p)` is used for sending datagrams and `receive(DatagramPacket p)` is used for receiving datagrams. To provide server functionality, the `DatagramSocket` binds to UDP port 2641. The `DatagramPacket` contains the IP-address which is set to the receiver IP-address before the datagram is sent with `send(DatagramPacket p)`. If a `DatagramPacket` is received, the IP-address is set to the IP-address of the sender. These semantics allows attributing outgoing and incoming packets to server connections and clients in the application logic. Thus, `DatagramSocket` outlines the semantic that has to be implemented as a drop-in replacement

for `DatagramSocket` with NDN. We call our drop-in replacement `HdlDatagramSocket`, which is able to transport `DatagramPacket`-encoded Handle Messages, but no generic UDP network payloads. To fit with the IP-address semantic, a mapping between a (virtual) IP-addresses and NDN end-nodes is established called `VTable`. It contains a HashMap which provides a mapping between the NDN name and a Virtual Internet Protocol Address (VIP). The VIP is only used as an internal identifier by the Handle library program logic for organizing the Datagram transport through NDN. By this, dualism of NDN name and a virtual IP number, both semantics of IP and NDN can work together in a closed and application specific system. The `NDN Transport` component (cf. Figure 5.29) translates between the semantics using the `VTable`.
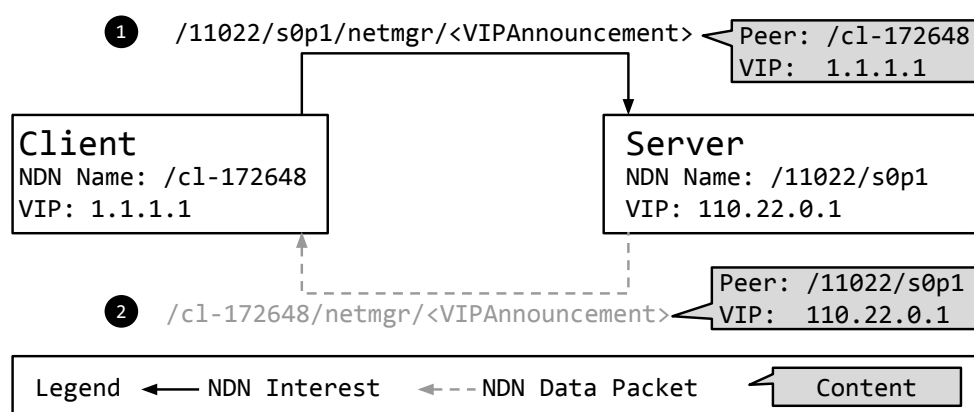
Figure 5.30: NDN PID Push Peer Announcement

The `VTable` semantic uses randomly generated IP addresses that are generated at each instantiation of the Handle library in a software. They do not contain any semantic or represent a real network location – they are just an opaque random identifier that is bound to the NDN name of the server. To have a working translation between VIP and a NDN name, the mapping has to be synchronized between the client and the server node before a successful communication over NDN is possible and Datagrams can be attributed from VIP to a NDN name. For this, a separate component called `NDN NetManager` (cf. Figure 5.29) is implemented that synchronizes `VTable` entries between NDN nodes before a NDN PID push communication is established. This component has an own NDN communication called `NFace`, which is used for sending out `VTable` mappings called `VIPAnnouncements`. As the communication of `NFace` is taking place on the transport face (`TFace`) outside the NDN data transport, an out-of-band synchronization is implemented. Using Figure 5.30, we explain the `VTable` synchronization. Before a client can establish a connection to a Handle Server, the VIP of the server needs to be acquired. For this, the function `addPeer(Name ndn_server_name)` is called using the NDN data name of the server. This name can be derived from the site information and the Handle prefix using NDN PID pull. In this example, the server `s0p1` managing the Handle prefix `11022` is contacted on the `netmgr` `NFace`. The client sends a `VIPAnnouncement` with its own NDN name and its VIP to

the server using an NDN interest ❶. For the transport of VIPAnnouncements in NDN interests, the representing Java Object is base64-encoded and NDN URL are encoded as part of the interest name. This is necessary, as the NDN interest packet has no data section (cf. Subsection 2.7.4). The server receives the VIPAnnouncement and adds the entry to its VTable. Then, the server returns a response with a NDN data packet to the client containing its own NDN data name and VIP as VIPAnnouncement ❷. After the response has been processed by the client and added to the client VTable, the synchronization is finished – both VTables contain the same client-server VIP-NDN mapping. This means that the communication between the client and the server can start using the VIPs, which are mapped into NDN data names for interest generation and data packet transport.
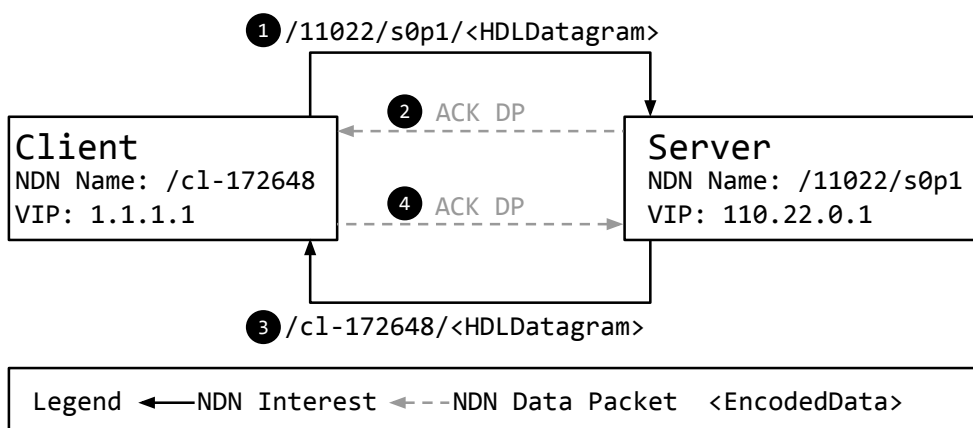
Figure 5.31: HDLDatagram Transport with NDN PID Push

To transport Handle protocol data through the HdlDatagramSocket, a NDN transport socket needs to be implemented. It uses NDN interest that are acknowledged through empty NDN response packets for a synchronous communication. To achieve asynchronous response handling in the client and the server, responses are sent back with NDN interests that are directed to the opposite direction. To transport messages of the Handle protocol, the Handle library decomposes Handle messages into a native packet format that is suitable for the selected transport. For UDP, the decomposition is done UDP datagrams. Due to the similarities of NDN PID push and UDP transport, we use the UDP decomposition functionality of the Handle library to feed our NDN transportation pipeline that is part of the native Handle communication subsystem. Figure 5.32 depicts the functionality of the pipeline. The pipeline takes the UDP datagram and wraps its content into HDLDatagram structures that contain the VIP, the binary payload and the payload size ❹. Afterwards, the HDLDatagram is encoded into a NDN data name (❺ and ❻) that is part of the interest containing the payload data. By using the VTable, the receiver name is determined by the VIP and then the interest is sent out by the NDN face responsible for transport (TFace ❼).
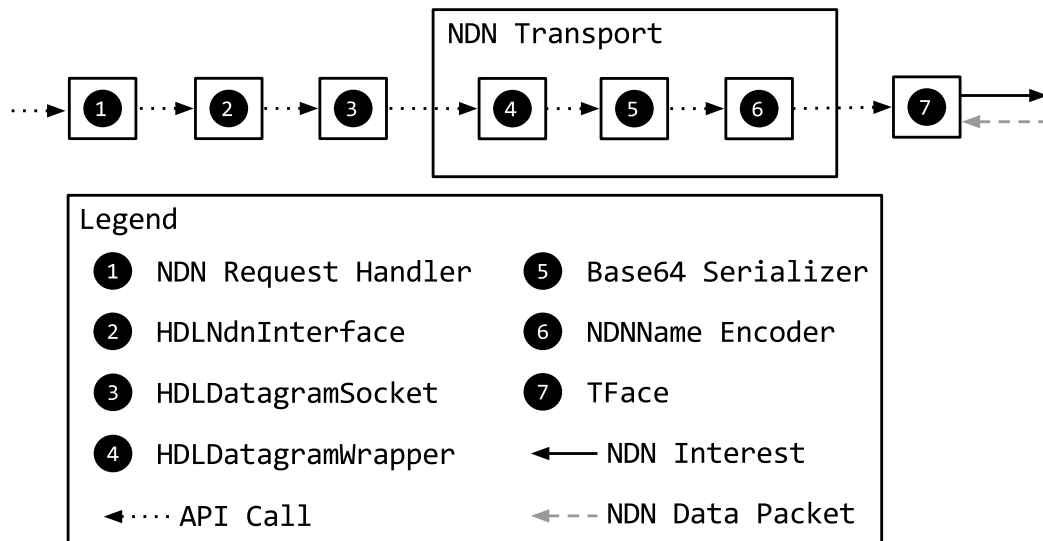
Figure 5.32: NDN PID Push Pipeline for Decomposing, Encoding and Sending Handle Messages Over NDN

For receiving Handle protocol messages with NDN PID push, a receiving pipeline is necessary that is depicted in Figure 5.33. While the sending pipeline works synchronously, where an API request triggers a series of NDN interests send outs, the receiving pipeline works event-driven. Therefore, it features a consumer-producer architecture with a `BlockingQueue` as buffer between the NDN data receiving, decoding and deserialization parts (❶ to ❹) and the request processing parts (❺ to ❽) [141]. The data receiving works in the opposite direction of the sending pipeline. When a NDN interest is received, then payload is decoded and then deserialized back into a Handle message. To follow the semantic of the UDP datagram socket, the HDLDatagramSocket replaces the VIP of the incoming datagrams with the VIP of the sender using the `VTable`. By this, the Handle application logic can attribute data to a specific client, although multiple client connections take place at the same NDN face. The Pending Request Management ❺ joins together the Datagrams into Handle protocol messages using the request id encoded in the Handle protocol header. At the end of the receiving pipeline, the Handle Software is able to process the message as if it would originate from any other interface. To clear the PITs of all intermediate NDN nodes, an empty NDN data packet is sent back. This answering packet is not processed by the sender but it cleans up the PITs in the network NDN network. If no answering packet would be sent out, the PIT entries automatically expire in the NDN nodes.
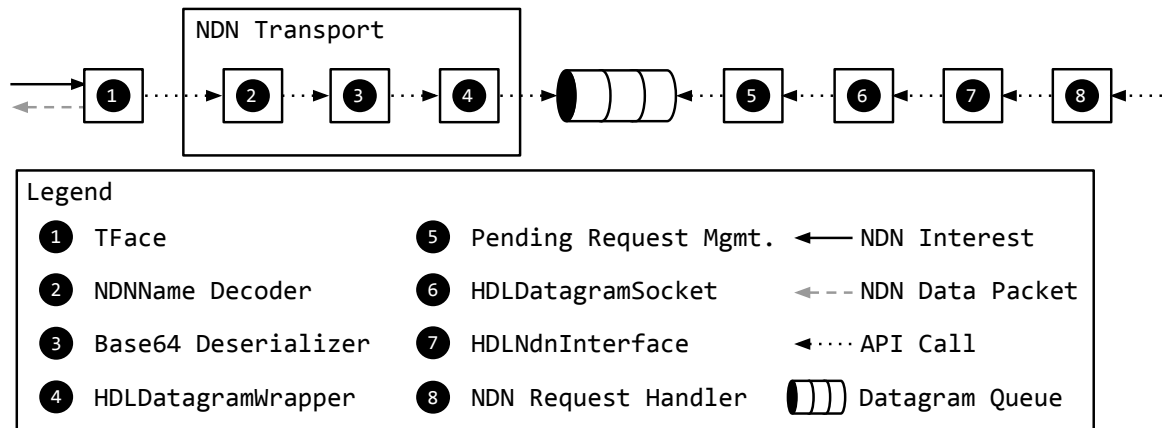
Figure 5.33: NDN PID Push Pipeline for Receiving, Decoding and Composing Handle Messages Over NDN

5.4.4 PID Publishing Subsystem

The PID Publishing Subsystem realizes the NDN PID push functionality. It accompanies the existing and new Handle subsystems that realize the native Handle protocol with an own independent subsystem. By this, the server and the Java Handle library are extended with new functionality that allows read-only access to Handle values through NDN. This read-only access allows resolving Handles very fast and efficiently over NDN and it also permits getting general information on the Handle prefix such as available servers or authorization information. In Figure 5.34, we provide an overview of the integration of the NDN publishing subsystem in the existing Handle server software stack. As NDN PID pull uses interests for Handle information acquisition, it uses an own NDN face for receiving interest ❶, which is independent from the faces used for NDN PID push communication. The interest is forwarded to the *InterestHandler*, which decodes the data name of the interest ❷. As PID pull and PID push may use the Handle prefix as their NDN name prefix for running their NDN faces, the NDN PID push face checks the length of data name in order to determine if the data packet is subject of a PID push communication. If the interest data name does not contain encoded native Handle message protocol packages, it is a PID pull interest, which needs to be processed by the NDN publishing subsystem. Then, the Handle value is resolved using the *HDLLibrary* of the server in ❸ and a NDN packet is generated that matches the interest and is later consumed the PIT in the NFD. To ensure a long-living availability of the Handle Packets in the NDN network, the data packets have to be compact. Thus, the *PayloadEncoder* encodes the complete Handle value into a two-byte padded binary array forming a native NDN binary payload (cf. Figure 5.35). The *EventProcessor* ❹ is responsible for dispatching interests from the NFD to the *InterestHandler* and dispatches NDN data packets to the NFD.
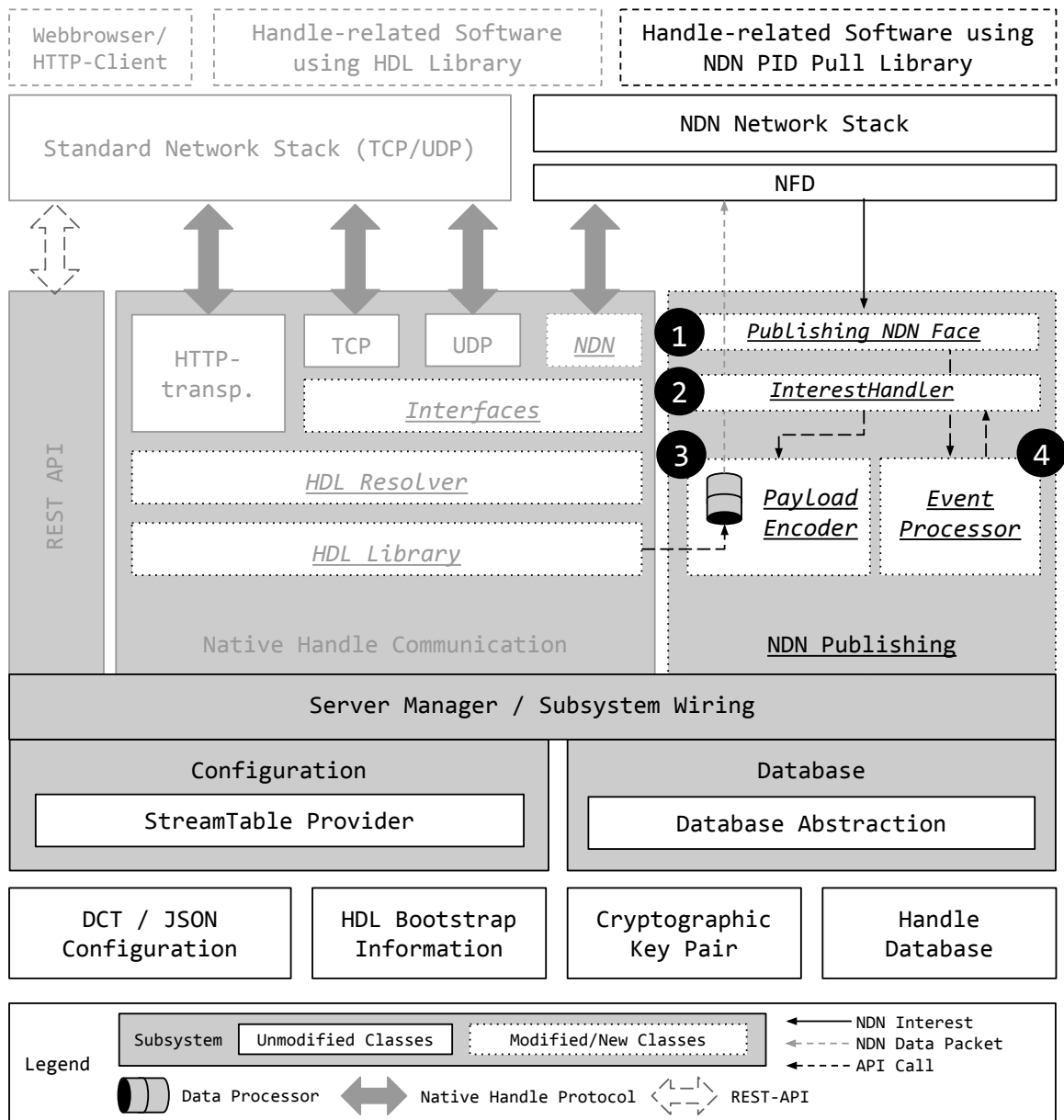
Figure 5.34: NDN Publishing Subsystem in Context of the Handle Server Architecture

In Figure 5.35, the realization of the NDN PID pull communication protocol is depicted. For retrieving a Handle with all its associated Handle values, the client can use the NDN PID pull Java library that mimics the API of the Handle library for Handle value retrieval and has been created for this proof-of-concept implementation. By this, the client cannot only retrieve Handles that store valid PID information such as target URLs but also administrative information stored in the LHS. Therefore, the server has registered the Handle prefix as NDN prefix at its local NFD, in order to receive all interests that have related hosted Handle

prefixes. For retrieving a Handle value, the access library decomposes a Handle into its prefix and suffix. The prefix is used as first part of the NDN data name for the interest. The Handle suffix is used as second part of the data name in the interest. To avoid a collision of Handle suffixes with NDN control directives, that are encoded as part of the data name, the Handle suffix needs to be encoded or escaped before (cf. Figure 5.17). In our proof-of-concept implementation, we use a base64-encoding that encapsulates the Handle suffix. With the completion of the interest, the client sends the interest to the server using the NDN network in step ❶. The interest is now passed from one intermediate NDN node to the other and reaches the server. We assume that no CS hit has taken place in the meantime at any intermediate NFD. The Handle is then resolved by the server. If a Handle has been found in the local database and its access permissions are set to *public read*, a NDN data packet is generated as an answer for the interest. The Handle and its public values are written into the data packet payload as binary data using Java object serialization for our proof-of-concept implementation. This allows storing Handle value information very compactly in a documented format that does not need any additional, time-consuming mapping. Hence, NDN PID pull data packets are very space efficient in order to satisfy NDN data structures and CS network caches. To ensure the validity and origin of the data packet, the private NFD key can be used for signing the data packet before submitting it to the client in step ❷. The data packet containing the Handle values is transported back to the client by the routes created in the PIT by the intermediate NDN nodes. The clients receive the data packet and deserializes the binary information into a Handle object suitable for further processing by Handle-related software. This mechanism allows transporting complete Handles from the server to the client with a simple binary-only mapping. As we can see in the following evaluation section, this fast packet composition and data encoding is very beneficial for the performance and stability of NDN-powered Handle networks relying on NDN PID pull.
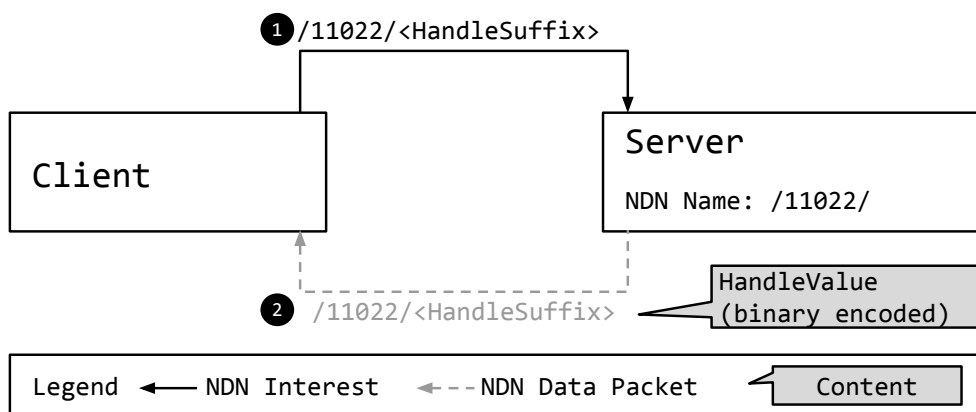

Figure 5.35: Handle Value Retrieval with NDN PID Pull

5.5 Evaluation

In this section, we provide an evaluation for our approaches on location-independent persistent identifiers. First, we describe the simulation environment used for the evaluation in the subsequent section. Secondly, we provide an evaluation of NDN PID push in Section 5.5.3. Finally, we provide an evaluation for NDN PID pull in Section 5.5.4.

5.5.1 Simulator Environment

For evaluating the approaches presented in the preceding sections, we use a dedicated NDN network evaluation and simulation environment. Currently, two major simulation environments are available. A simulation for low-level NDN network activities is *ndnSIM* 2.0 [142], which is based on ns-3, a discrete-event network simulator for Internet systems [143]. NdnSIM implements all basic operations of NDN and allows simulating NDN networks with NFDs, network components, as well as generators for different data request distributions. NdnSIM is implemented in C++ and simulates all NFD components like PIT, FIB and Faces as modular components in the simulation. For this, ndnSIM contains C++-APIs that are compatible with the native C++ NDN development libraries ndn-cxx (NDN C++ library with eXperimental eXtensions) [144]. For using Python applications in ndnSIM, pyNDN is available as C++-Python-Bindings for ndn-cxx [145]. By using the ndnSIM simulation environment, C++ and Python applications can be used as native application in simulated network environments. For this, the application make use of the compatible simulator APIs, instead of the regular APIs provided by the NFD, which are handling the communication to the NDN network in real deployments.

Although ndnSIM has a large impact on the research community, it limits researchers to using C++ compatible applications, as compatible APIs are not available for other programming languages. Furthermore, the simulation environment is simplified in comparison to NDN deployments on real computers running an operating system as intermediate layer between applications for providing hardware-abstraction. In ndnSIM, there are no intermediate layers integrated that simulate inter-process communication which is used to connect the NDN-applications to the NFD through a Unix domain socket / Inter-Process Communication (IPC) socket. Hence, for assessing our NDN-enabled Handle stack, we use a different simulator called *Mini-NDN* [146] that also has a large impact on the NDN research community and focus on the simulation of networks on a higher perspective using the virtualization abilities of the Linux operating systems. Mini-NDN is based on Mininet [147] [148], which emulates an entire NDN network using all NDN libraries and tools like NFD and NLSR on a single Linux system. For this, Mini-NDN uses process-based and network namespace virtualization of the Linux kernel and provides for each virtual host an own private network interface and virtual network switches based on SDN technology using the reference implementation of Open vSwitch and OpenFlow.

Virtual hosts can be connected through a configurable network topology that includes setups of connection speeds, packet loss percentages and transmission delays. By this, entire NDN networks can be emulated that are running the Handle client and LHS on different virtual hosts. This allows drawing conclusions on the expected performance of the location-independent PID system and to provide an analysis of each layer (Handle system, NDN stack, operating system overhead and residuals) and its performance impacts. Additionally, this approach allows integrating NDN-enabled software that is using an arbitrary NDN-library and programming languages. Hence, we can employ our NDN Handle stack implemented in Java using jNDN [135] on Mini-NDN.

To eliminate the overhead of full virtualization, we use a physical computer to execute the evaluation runs with Mini-NDN. As Mini-NDN uses Mininet as backend, we rely on Linux operating system namespace virtualization, which is similar to Docker. It provides process and network isolation on the kernel side without introducing significant overhead to simulate multiple hosts and a virtual network [149]. For the evaluation, an Intel i5-2400 processor with an Intel Q65 Express chipset is used with 16GB RAM running Ubuntu 14.04.1 64bit with Linux Kernel 3.16.0-53-generic. The system is equipped with a Samsung EVO 850 Solid State Disk (SSD) drive that delivers 520MB/s read and 500MB/s write performance. Mini-NDN in version 0.1.1. [146] was used in conjunction with NFD version 0.4.1 [136].

5.5.2 Evaluation Input Data Preparation

After selecting the simulation software, we have to make realistic assumptions on the network topology. This is important for a comparison of our approaches with the classic location-based Handle System. For a realistic Handle network topology, we want to simulate an average network path between a Handle client and a primary LHS. As Handle servers are located all over the world, we have to choose servers that are frequently used by researchers. To get a real-world sample of PID resolutions, we employ anonymized telemetric data from one of the official Handle HTTP-proxies operated by GWDG. The data collection process that we accomplished in 2014 is depicted in Figure 5.36. The official HTTP-Handle proxy operates on the domain `http(s)://hdl.handle.net`. To balance the user load, a round-robin DNS, hosting the domain, is distributing each request to one of three Handle HTTP-proxy servers with a probability of 33% ❶. One of these HTTP-proxies performing PID resolutions is located at the GWDG data center ❷ and produces telemetric data including the Handle prefix of the user request. The HTTP-proxy performs the resolution in behalf of the HTTP-client (cf. Figure 2.8) ❸. The telemetric data is strictly anonymized in a way that a time stamp, the Handle prefix, the character count of the PID target URL and a salted hash of the Handle suffix are stored ❹. In order to determine the character count (URL length) of a PID target URL, a mining tool ❺ is used, which is explained in detail in Section 6.5.1.1. The target URL of a PID is deleted after determining its character count, as we are not interested in the URL and it may contain sensitive user

data. The hashed version of the suffix is stored in the sample to offer the possibility to count unique PID resolutions, while not exposing the full PID request for data analytics. By this, we can draw a sample that is based on 33% of all Handle PID resolution requests performed by hdl.handle.net. Furthermore, any identifying data like User-Agent String, user IP-address or the full HTTP-request has been deleted in the anonymization process to ensure confidentiality of user log data and to comply with German and EU data protection regulations. Thus, we only use anonymized telemetric data as input and present aggregated numbers that do not allow identification of users from hdl.handle.net. Our data sample contains 22,757,503 PID resolution requests from hdl.handle.net in a time span of 100 days (June 2014 - August 2014). We create a Python program for classifying all PID resolution requests towards their Handle prefixes (cf. Appendix A.3.1). This aggregation was also used for our analysis in the NAS 2015 paper [104].
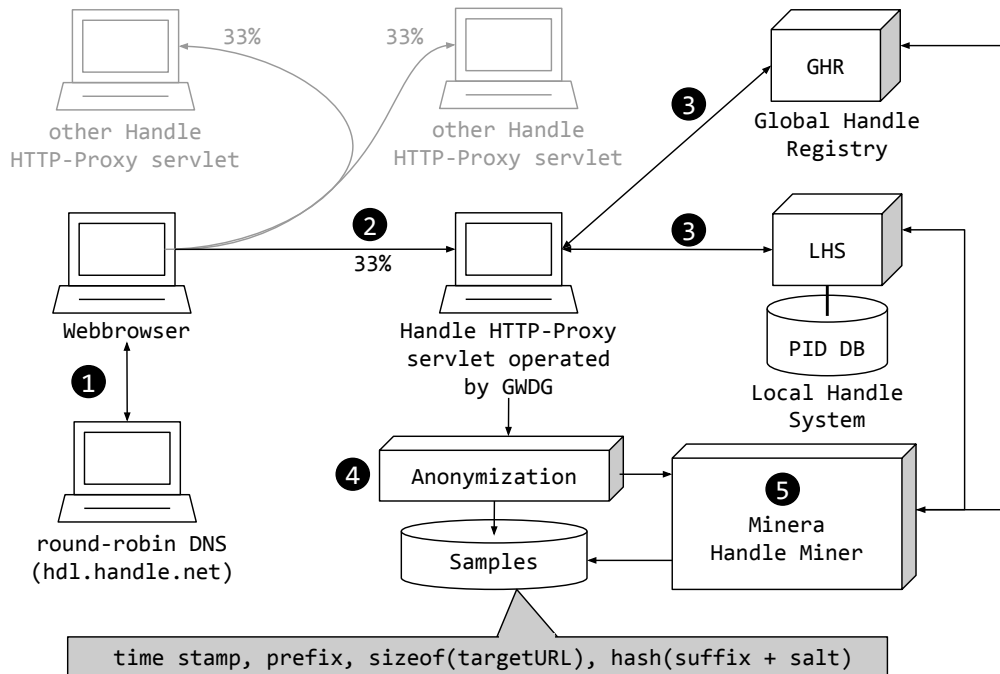

Figure 5.36: Acquisition of Data Samples for Real-World PID Resolution

5.5.2.1 Network Hop Calculation

For network hop count, we use the definition by [150], which donates that a *hop* is every router a network packet (or in case of UDP a datagram) passes while travelling from its origin to the source. We follow the extended definition of a network hop for NDN as proposed by the NDN research community [11] [151] and add a NDN NFD as a counting occurrence identically to a location-based network router. Thus, a hop count of two means that a network packet has traveled through two routers in the location-based case

or through two NFDs in the NDN case. By this, extension, we provide a comparability between location-based and NDN network scenarios. After the classification of the requests according to their PID prefixes, we measure the network hops between a measurement server, and all primary LHS associated to the respective Handle prefixes that we extracted from the anonymized telemetric data from 2014. For this, we first need to extract the IP-addresses of all primary Handle LHS by using the public REST API of the GHR. The script for extracting the IP-addresses of the primary site is available in appendix A.3.2. Then, we use another script for counting the hops between the primary LHS and our server located at the same data center network segment at the GWDG as the official Handle HTTP-proxy operated by GWDG. The script for calculating the hop count is available in appendix A.3.2. We can estimate the hop count between the official Handle HTTP-proxy and the primary LHS, which is serving Handles for a specific Handle prefix. Our measurement server is located at the GWDG datacenter in the same network segment as the official Handle HTTP-proxy operated by GWDG. With this setup in place, we have the identical network topology for determining the network counts, as the official Handle installation uses for their operative HTTP resolution services. Using the acquired network hop counts, we can set up a network topology in the Mini-NDN simulator that is very close the real-world topology concerning hop count and packet latency for average resolution cases. As we are interested in the average performance, we have to condense the observed wide variety of network hop counts to an average network hop count number. For calculating the average hop count as parameter for our simulation, we use the weighted mean, which includes the number of successful PID resolutions per prefix. Thus, we can determine the average network path length with:

$$\bar{x}_w = \frac{\sum_{i=1}^{n} w_i x_i}{\sum_{i=1}^{n} w_i},$$

where $\bar{x}_w$ is the weighted arithmetic mean of hops and w_i is the percentage divided by 100 of all collected measurements of successful PID resolutions. The collected measurements are attributed to a Handle prefix with the rank i. The variable x_i is the number of hops between the primary LHS, hosting a Handle prefix and our measurements server.

Rank (i)	Prefix	Hits	Percent (w_i)	Hops (x_i)
1	Prefix-A	3,848,567	16.9112	10
2	Prefix-B	1,882,874	8.2736	12
3	**Prefix-C**	**1,571,502**	**6.9054**	**0** *corr. 14*
4	Prefix-D	985,922	4.3323	10
5	Prefix-E	862,209	3.7887	13
6	Prefix-F	855,999	3.7614	9
7	Prefix-G	344,398	1.5133	15
8	Prefix-H	301,685	1.3257	21
9	Prefix-I	301,667	1.3256	12
10	Prefix-J	264,584	1.1626	21
11	Prefix-K	256,810	1.1285	16
12	Prefix-L	232,642	1.0223	27
13	Prefix-M	230,000	1.0107	12

Table 5.3: Network Hop Calculation for the Top-10 Handle Prefixes

In Table 5.3, we see an excerpt of the first 13 ranks from the classification of Handle PID resolution requests according to their Handle prefixes, based on the data that was collected as depicted in Figure 5.36. After rank 13, the amount of PID resolutions attributed to a specific Handle prefix drops below one percent (cf. Section 6.5.1.1 and Figure 6.12). The full data is available in appendix A.3.3.

We observe for Prefix-C, located on rank three, a hop count of zero. The hop count of zero is correct, as the GWDG operates own Handle prefixes in the same data center than the official HTTP Handle proxy. Furthermore, our measurement server is located at the same data center and in the same network segment. Thus, the traffic between the servers is not routed between network segments and as a result, the hop count is zero. The observation that a primary LHS is located in the same network segment as one of the official Handle HTTP proxies is very unlikely if we take into consideration as over 12,000 Handle prefixes running on LHS already exist in 2014 (cf. Table 2.1). Hence, we replace the hop length of the occurrence of Prefix-C on rank three with the (unweighted) arithmetic mean $\bar{x} = 14.28 \approx 14$ of all hop lengths.

Beside this observation, we can see in the data that the Prefix-A and Prefix-C, which are operated by the GWDG, are located in the top ten although we would expect a different distribution for round-robin DNS assignment that was deployed for the domain name hdl.handle.net at the time of data collection. We assume that the organizations using both prefixes for PID tagging did not use the domain name hdl.handle.net and thus a round-robin assignment of the HTTP Handle proxies, but rather circumvent round-robin assignment by

directly using the IP-address of the GWDG PID-HTTP-proxy server. This observation is not unusual, as data repository administrators use this strategy for reducing the response time due to shorter data transmission paths. We can assume that at the time of data collection in summer 2014, repository administrators located closer to the other official HTTP Handle proxies (mainly beyond the Atlantic) applied these strategies as well.

As a result, we can compute the weighted mean of the hop counts $\bar{x}_w = 12.8626 \approx 13$. Hence, $\bar{x}_w = 13$ is used as a standard network hop count in Mini-NDN simulation for evaluating our approach.

5.5.3 Native Handle Communication Using NDN PID Push

In this section, we evaluate the native Handle protocol communication using NDN with NDN PID push communication. For the evaluation, we setup a serial chain of network hosts with a length of n nodes within the Mini-NDN. The complete setup is depicted in Figure 5.37 and the source codes for setting up the evaluation environment are available in Appendix A.3.4 and A.3.5. In the upper part of the figure, the NDN simulation scenario is depicted, while the lower part shows the location-based simulation scenario using TCP transport. By this, we simulate the transport network between a Handle HTTP proxy and a LHS. Within the network, the native Handle application protocol is used, employing TCP for the location-based scenarios or NDN PID push for the location-independent scenarios. The chain starts with a client node that performs Handle typical operations on a server which is connected to the end of the network chain. The client measures the time between sending the first packet of the request and receiving the last packet from the server completing the request. Each host in the chain runs an instance of a NFD for the location-independent case or a TCP packet forwarder implemented in Java for TCP communication in the location-dependent scenario. The source code of the TCP packet forwarder is available in Appendix A.3.6. The usage of TCP user land packet forwarding is necessary, as there are no kernel space implementations for NDN yet. By this, we compare NDN- and TCP-based networks with implementations in the Linux user space to provide a comparison on the same operating system level. Each chain host features two network cards connected for the previous and successor node of the chain forming a Classless Inter-Domain Routing (CIDR) /30 network. The simulation is executed for each case with zero to 13 hops for NDN PID push, UDP and TCP transport of the native handle protocols.
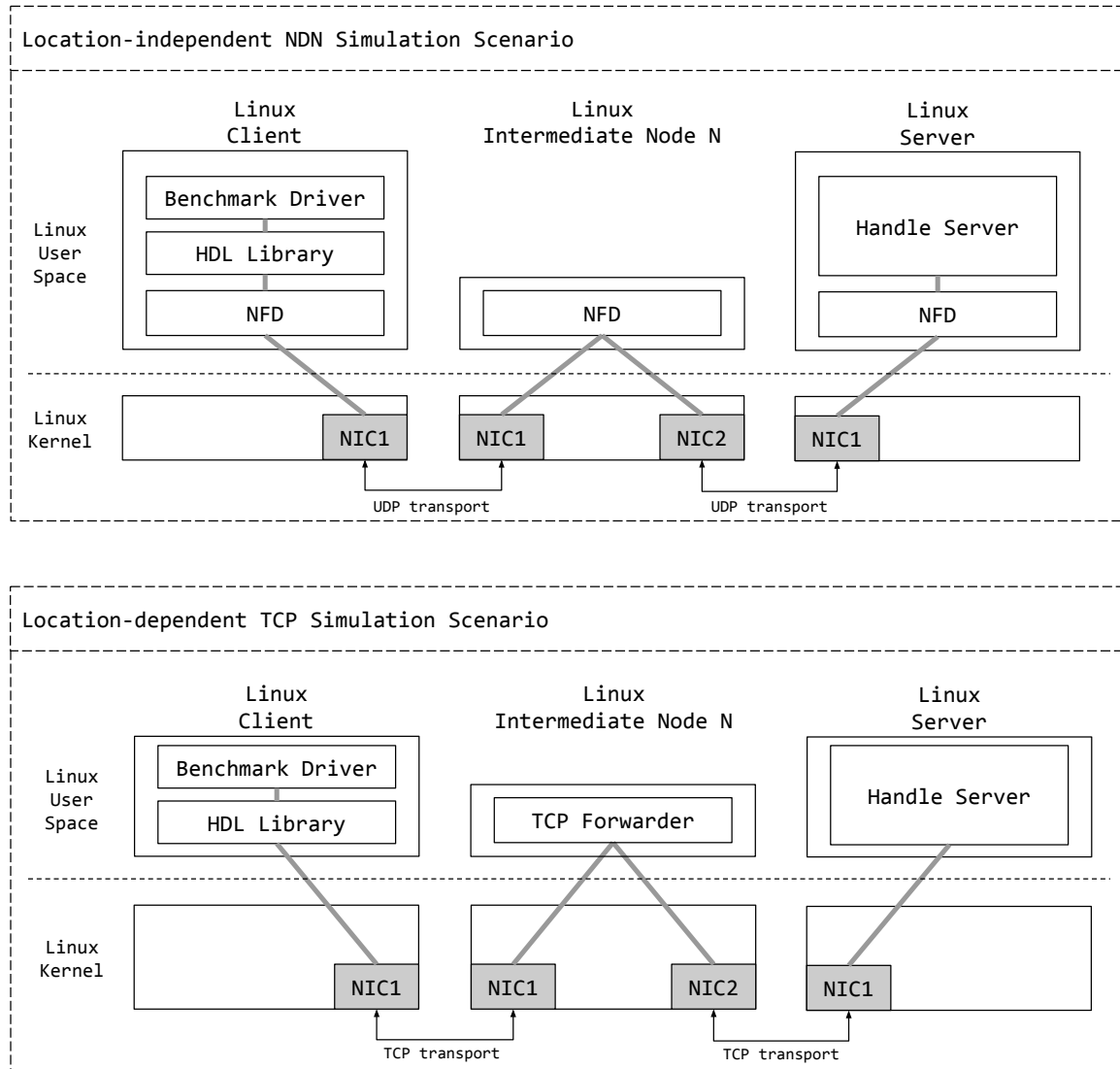
Figure 5.37: NDN PID Push Evaluation Simulator Setups

For the evaluation, we have three major types of scenarios that we present in the following. Together with the setups of NDN-based transport and native Handle transport using TCP, we have in total six evaluations to perform (two for each scenario). We perform each evaluation with zero to 13 nodes, so we have in total

2 types $*$ 3 scenarios $*$ 13 hop setups $=$ 78 simulator runs, where each run performs at least 10,000 requests. We describe the results of the following simulation runs and provide an evaluation in Section 5.5.3.4:

- Create PIDs Authenticated over Encrypted Transport (cf. Section 5.5.3.1)
- Resolve PIDs Authenticated over Encrypted Transport (cf. Section 5.5.3.2)
- Resolve PIDs without Authentication over Plaintext Transport (cf. Section 5.5.3.3)

5.5.3.1 Create PIDs Authenticated over Encrypted Transport

In this scenario, the goal is to create a Handle PID with a target URL using client authentication and transport encryption. First, the Handle client uses its private key to authenticate against the Handle server. Secondly, after the successful authentication, an encrypted Handle session is set up and a Handle value is created. In order to acquire target URLs for the PID creation evaluation, a realistic set of target URL data is required, which mimics the structure and length of the target URLs found in real-world PID populations. In order to generate a target URL data set of 100,000 items with the required attributes, we extracted randomly selected PIDs from the 2014 telemetric data set that belong to the prefix 11858, which is owned by the GWDG. Then, the target URLs were extracted with public APIs using the Minera target URL mining tool (cf. Section 6.5.1.1). To protect the target URL data of GWDG PID customers, while preserving the length and the structure of the target URL sample data, all alphabetic characters were permuted randomly with other alphabetic letters for each target URL.

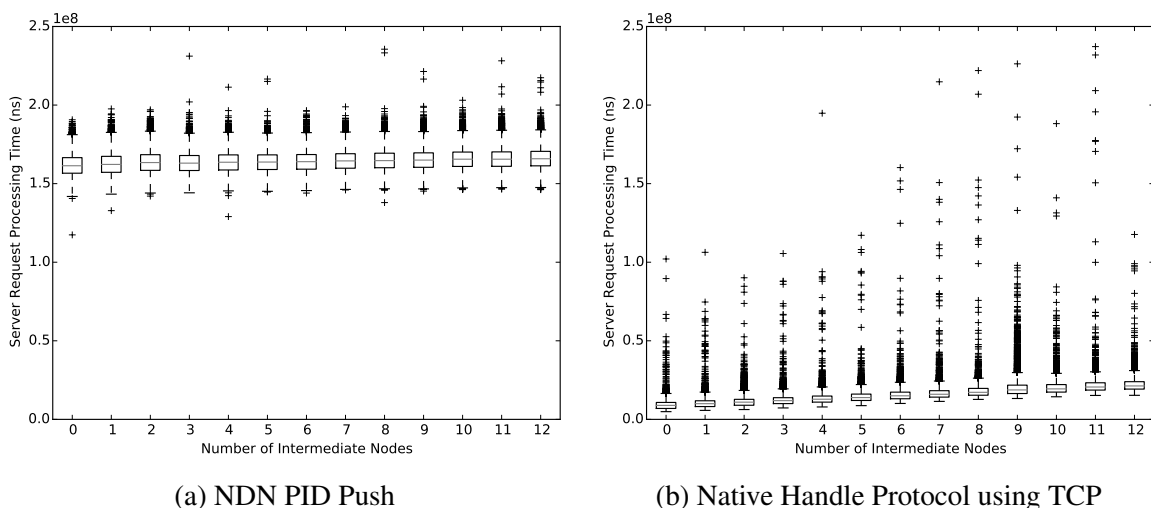| (a) NDN PID Push | (b) Native Handle Protocol using TCP |

Figure 5.38: Server Request Processing Time for Authenticated PID Creation

Figure 5.38 shows the processing time for an authenticated PID creation request measured at the client node. On the left side, we see the result for NDN PID push and on the right side using the native Handle protocol over TCP. As we can derive from the figure, the native Handle protocol over TCP outperforms NDN PID push approximately by the factor of 15. The overhead introduced by every additional node is almost identical in both scenarios, while TCP has a large slope in comparison to the overhead created by NDN. Furthermore, the variance on the TCP reference scenario is larger. Both effects of larger slopes and larger variance can be explained by the connection's persistent character of TCP that involves an establishment of connections between nodes using a hand-shake procedure, while NDN

relies on UDP traffic that does apply connection management and thus outperforms TCP in the dimension of latency and thus application's responsiveness.

5.5.3.2 Resolve PIDs Authenticated over Encrypted Transport

The goal of this scenario is to resolve an existing PID using an encrypted and authenticated session. First, the Handle client uses its private key to authenticate against the Handle server. Secondly, after the successful authentication, an encrypted Handle session is set up and a Handle value is resolved against its target URL. The PID database is preloaded with 10,000 unique PIDs containing our generated sample target URLs.

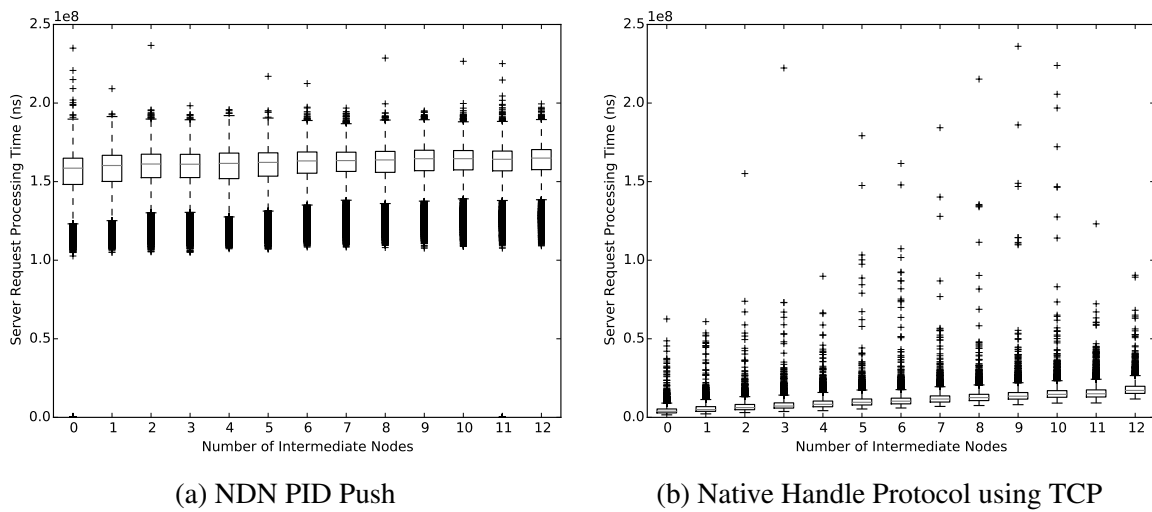(a) NDN PID Push (b) Native Handle Protocol using TCP

Figure 5.39: Server Request Processing Time for Authenticated PID Resolution

Following Figure 5.39, we can see that the performance is almost identical to the case of authenticated PID creation (cf. previous section). This is no surprise as the establishment of an encrypted session after client authentication takes 38 round-trips in the case of TCP, while the actual resolution request only uses 8 round trips between the client and the server. This ratio is similar for the NDN PID push use case. Thus, $\approx 75\%$ of the resolution time is spent for client authentication and setting up an encrypted communication channel, while the minority of the response time is actually different from the previous scenario. As we can see, PID resolution is faster than PID creation in the NDN PID push case. The reason behind this is that the Handle server waits for the confirmation of a durable database write/update before sending an acknowledgment to the client in the PID creation scenario (cf. Figure 5.27, subsystem ❻).

5.5.3.3 Resolve PIDs without Authentication over Plaintext Transport

The goal of this scenario is similar to the previous one. But in contrast, it does not use authentication or encryption for the PID resolution. Resolving PIDs without authentication over plaintext transport is applied in the average PID authentication that is performed mostly between the Handle client and the Handle server and is employed in almost all public PID resolutions. The PID database is again preloaded with sample PIDs as described in the previous scenario.

(a) NDN PID Push   (b) Native Handle Protocol using TCP
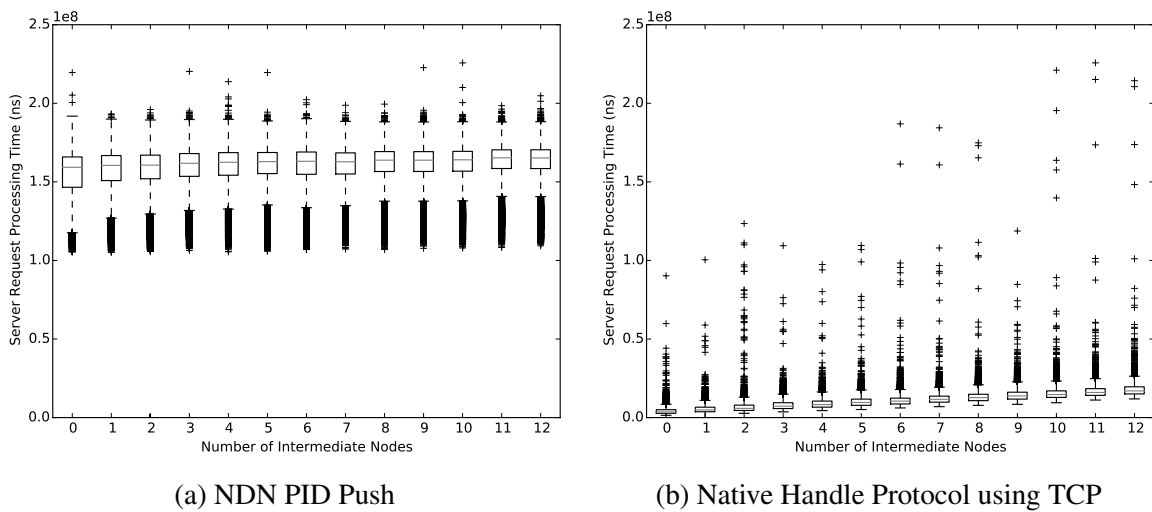
Figure 5.40: Server Request Processing Time for Plain Text PID Resolution I

As we can see in Figure 5.40, unencrypted PID resolution provides a similar outcome in comparison. In contrast to the two previous scenarios, the response times are faster, which is due to the absence of authentication and encryption. The difference between the native Handle protocol using location-based TCP and NDN PID push is persistent at a constant factor.

5.5.3.4 Comparison of NDN PID Push and TCP-based Native Handle Communication

In this subsection, we compare the results for NDN PID push and TCP-based native Handle communication. In Figure 5.41, we provide a direct comparison between NDN PID push on the left side and the native Handle protocol on the right side. Lower values imply a better responsiveness for server requests and a higher performance regarding the number of requests processed in a fixed time range. The figure shows the average processing times for all three scenarios explained before as a summary. As we can derive from all evaluation

figures above, NDN PID push is slower than the location-based TCP-based implementation by an approximate factor of 15.

(a) NDN PID Push
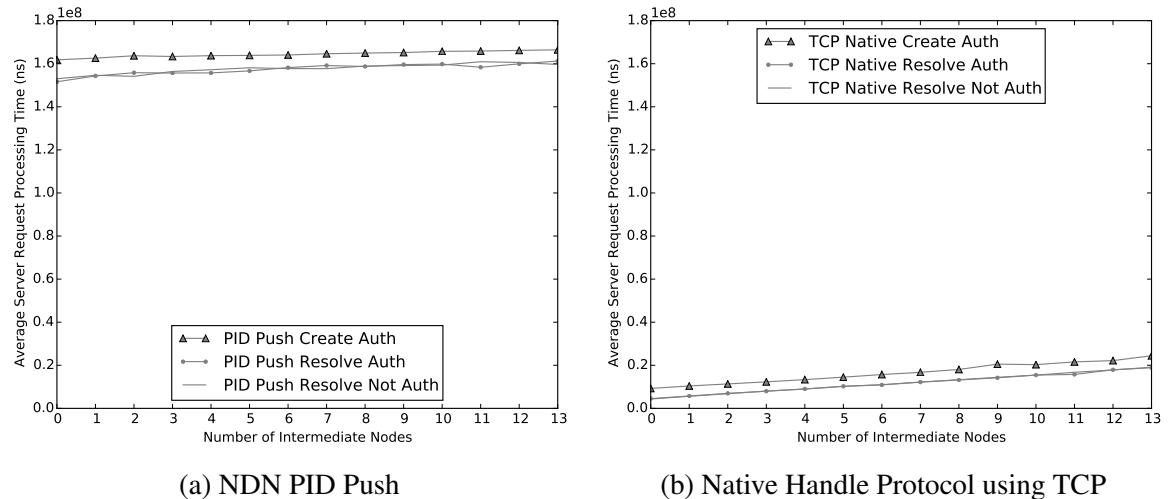

(b) Native Handle Protocol using TCP

Figure 5.41: Average Server Request Processing Times

This can be explained by the fact that NDN PID push is designed as a client-to-server protocol that allows direct end-to-end communication without requiring previous knowledge of the network location of a server. In contrast to our alternative NDN Handle implementation of NDN PID pull, it does not involve any NDN network caching capabilities. This is caused by the fact that NDN-based caching is only subject of NDN data packets but not of NDN interests, the foundation of NDN PID push. Thus, NDN PID push uses NFDs as a store-and-forward network to transmit native Handle protocol packets from one NDN node to another. Additionally, it is important to mention that there is an overhead caused by NDN which leads to a performance penalty and yields worse results in direct comparison with location-based protocols like TCP or UDP for the NDN PID push case.

5.5.4 PID Publishing using NDN PID Pull

In this section, we evaluate the performance of the NDN PID pull approach. As NDN PID pull is only capable to perform retrieve operations on Handles and, in contrast to NDN PID push, cannot perform create, update or delete operations, we only consider PID resolutions in the evaluation. For this, we start with the scenario presented in Section 5.5.3 and compare NDN PID pull with an unauthenticated PID resolution using the TCP-based native Handle protocol. The plots are provided in Figure 5.42.

(a) NDN PID Pull   (b) Native Handle Protocol using TCP
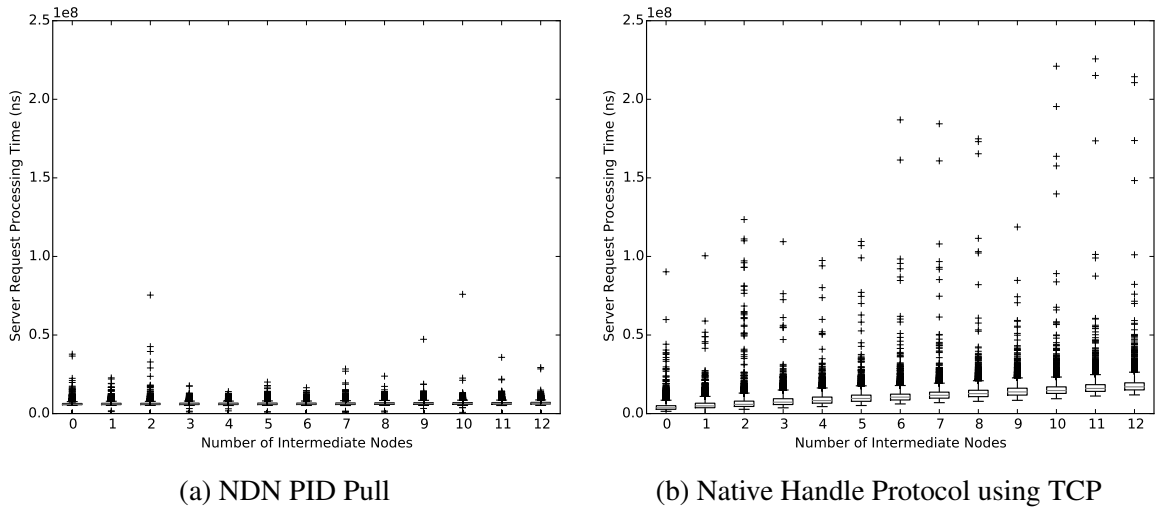
Figure 5.42: Server Request Processing Time for Plain Text PID Resolution II

As we can see in Figure 5.42, NDN PID push is able to outperform the native Handle protocol using location-based TCP when looking at the average resolution times for a serial network path with variable length. The results are according to our approach, as NDN PID push matches the semantic level of Handles with the semantic level of object granularity in NDN. By this, the transition between the logic data structure and the NDN network structure is very lean and efficient. In contrast, the native Handle protocol breaks down the Handle request into protocol messages that are further decomposed into TCP data packets which are sent over an established end-to-end network connection. The end-to-end connection setup, the data decomposition at the server and the data composition at the client requires more Central Processing Unit (CPU) time, data structures and states which in the end lead to a slower response time. This observation highlights an advantage of NDN that offers a high-level semantics of complex and named data structures in low-level network operations, leading to faster reaction in cases where simple network semantics requires a more complex data handling.

While the performance gap between a native Handle protocol stack using TCP and NDN PID pull is rather small, it is more significant in network scenarios that suffer from unreliable network connections with TCP packet loss. In order to simulate this, we use the packet loss feature of Mininet that is behind Mini-NDN. This feature allows dropping a fixed percentage of data packets in a network connection using *netem*. Netem is a framework in the Linux kernel which is capable of simulating faulty network connections with variable packet delay, loss, corruption and bandwidth impairments. While netem allows different distribution patterns such as Bernoulli or Gilbert-Elliot, Mininet uses a random distribution for packet loss. For our scenario, we assume a packet loss of five percent at a random link between two TCP forwarders or NDN nodes. By this, the connection between the client and the server is working faulty. Under these conditions, we repeat the experiment depicted in

Figure 5.42 and perform a non-authorized PID resolving on 10,000 PIDs using NDN PID pull and native Handle protocol with TCP connectivity.
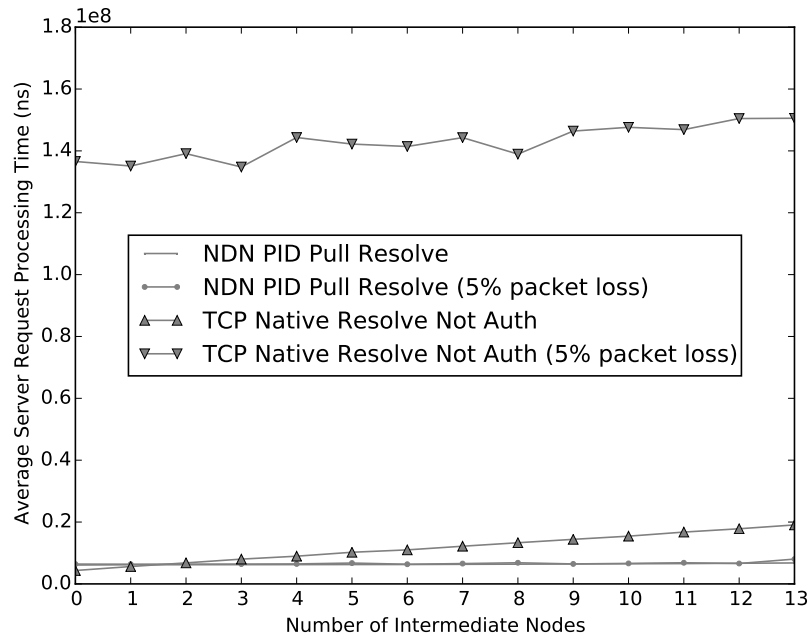

Figure 5.43: Comparison of Average Server Request Processing Times in Packet Loss Scenarios

In Figure 5.43, we find that NDN provides a significantly shorter response time for PID resolution than native Handle PID resolution using TCP. This is caused by the better error correction provided by NDN that is able to anticipate network errors between intermediate NFDs in contrast to location-based networks that are only able to deliver error correction on end-to-end base. If a packet loss or network error is detected, the intermediate NFDs can request a retransmission of data packets from their neighbor node that is in closer network distance than the data source. By this, the transmission distance of network control parameters and retransmitted data is shorter and thus transmission is accelerated in networks with packet loss. Benefiting from this advantage of NDN, NDN PID pull is able to achieve significantly higher performance in faulty networks.

Another important aspect in the development of NDN is reduction of network load through implementing network cache mechanisms. In this paragraph, we describe the impact of the content store-based caching implemented in each NFD on the performance of NDN PID pull and highlight its importance for parallel connection speed up. For assessing the parallel connection capability of NDN PID push, we compare the PID resolution time with the PID resolution using the native location-based Handle protocol over TCP. For the evaluation of parallel connections, we use a pool of 100.000 PIDs generated with our sample target URLs and draw a random sample of 10% for each client. Hence, some PIDs in the

samples for each client are overlapping at each other in a small percentage. This leads to multiple resolutions of a small number of PIDs in the evaluation, which is needed to trigger cache hits in the NDN network. Each client resolves its PID sample independently at the server. The resolution of all PIDs is done in parallel and the number of clients resolving their PID is increased in each experiment.

Following to the design of NDN PID pull, valid resolution results remain in the CS of the intermediate NFD. If a PID resolution request is sent to the server and has been stated by a parallel connection before, the resolution response is directly delivered by the intermediate node from its CS. This design provides two benefits for handling parallel connections faster in PID resolution. First, efficient network caching answers resolution requests very fast as the number of network hops is reduced in comparison to a full direct-server connection. Additionally, CS in NFDs are organized as key-value pair storage backed by memory databases that retrieve data significantly faster than the databases used in the Handle server (Berkeley DB, MySQL and Maria DB). Moreover, the caching capacity provided by all intermediate NDN nodes exceeds the capacity of the built-in memory cache of the Handle server, leading to a higher cache hit rate and a better coverage of resolution requests for Last Recently Used (LRU) caching designs. Secondly, the PID resolution requests that are satisfied through caching are not reaching the Handle server in the case of NDN PID pull, as the NDN-friendly design is able to fully complete resolution requests from the vanilla NFDs not running any PID-related software. Thus, as a second effect, the system load of the LHS caused by the PID resolution requests is reduced and allows faster processing the resolution requests. This accelerated resolution helps to feed the caches faster, which leads to a better cache hit rate and faster resolution request processing. This effect is self-amplifying and grows with the number of parallel connections. As a combination of cache-based PID resolution responding and Handle server load-reducing side-effects, NDN PID pull delivers significantly higher performance in parallel connection processing compared to TCP-based native Handle resolution. Figure 5.44 depicts the accumulated time for all PID resolutions measured at all involved clients using 10,000 PID for the resolution benchmarking out of shared PID pool with a size of 100,000 on the Y-axis. For providing a comparison for different numbers of parallel connections, additional benchmarks are stated on the X-axis for an increased number of parallel connections. In Figure 5.44, we see that NDN PID pull is able to fulfill parallel requests for all clients faster than the native not-authenticated Handle protocol over TCP. Moreover, we observe that NDN-based Handle resolution scales better than location-based native Handle connectivity with an increase of parallel connections. This property of superior outscaling for content-consumation from single-instance sources is a major NDN benefit that is provided by NDN PID pull. In contrast to NDN PID push, no negative impact causes by an increased network hop count can be observed and the scale out behaviour of NDN PID pull is similar to the native location-based Handle protocol.
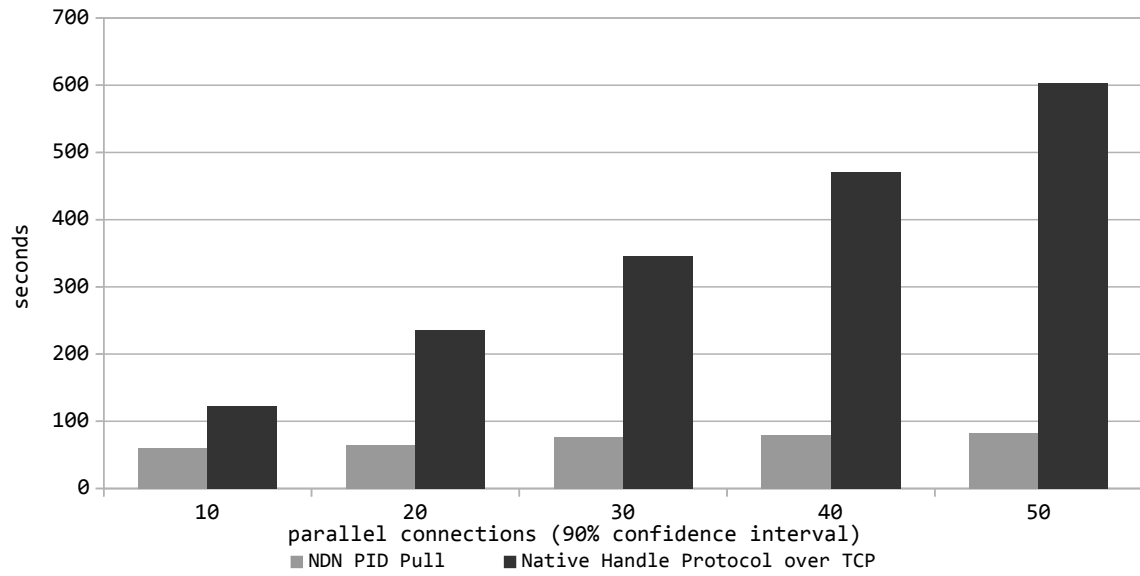
Figure 5.44: Accumulated Resolution Time vs. Number of Clients for 10,000 PID resolutions per Client

Chapter 6

Location-Independent Data Access using Persistent Identifiers

While the last chapter provided a solution to make PIDs independent from network locations through NDN, this chapter aims at accessing location-independent research data through PIDs. As explained in Definition 2.8, PIDs are simple maintainable identifiers that refer to digital objects. In their operational realization, PIDs contain metadata and a resolution target that points to the current location of the digital object. Resolution targets are often stated as URLs (cf. Figure 2.3) and frequently change over time. Thus, PIDs need continuous maintenance and supervision. While the effort for every PID is comparatively small, the effort will increase tremendously with the advancement of e-science and Internet of Things (IoT), where the number of datasets is rising by magnitudes, as it is already the case today, where hundreds billions of sensors and large research experiments are subject of PID assignment. To create maintenance-free PIDs, we propose a new approach for accessing research data through location-independent technology that is integrated into PIDs. By this, we eliminate the need for resolution target adjustment by employing location-independent network technologies such as content-centric and overlay network technology. To show the validity of our approach, we use the Handle PID system in conjunction with Magnet Link (cf. Section 2.5) scheme that we extend for the use in NDN. Furthermore, we extend the Magnet Link scheme with the possibility of storing trust items to secure location-independent access through cryptographic signatures and certificates. Inside the Magnet Link, we encode access information for state-of-the-art location-independent access technology using BitTorrent and NDN that is subject of current research. In contrast to existing approaches in literature, our approach does not require major modifications of the Handle PID system and does not suggest a green-field implementation of a next generation PID system. Thus, our approach embraces the slow change momentum of research data management and can be integrated into existing PID infrastructure for location-independent research data dissemination.

Aspects of our approach were presented at the 2016 Federated Conference on Computer Science and Information Systems (FedCSIS) [51]. In this thesis, we present an improved and extended version of our approach.

6.1 Improvements and Benefits

Let us now have a look at the benefits and improvements of our approach for location-independent data access using persistent identifiers:

1.) Maintenance-free PIDs is an improvement to a PID system that we describe in this chapter. We create an approach for persistent PID resolution targets which is based on the content that a PID is linking to and not on its network location. With persistent content-based resolution targets, an adjustment of PID target URLs is not needed anymore. This addresses the challenges named in *Problem Statement 3* (cf. Chapter 3). Figure 6.1 visualizes the approach of location-independent persistent resolution targets. As we can see, the prerequisites for accessing data after a successful PID resolution (cf. Figure 6.1, ❶) are reduced to the fact that data has to be connected (online) to the Internet. The ephemeral target URL is replaced by a persistent location-independent access information data set.
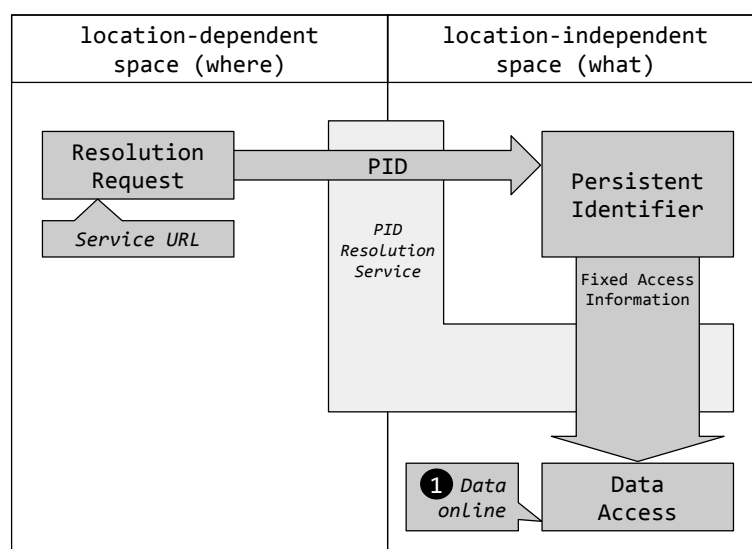

Figure 6.1: PIDs with Location-Independent Resolution Targets

2.) Trusted data dissemination with PIDs is an additional benefit that we can provide with the approaches we present in this chapter. For this, we add cryptographic verification items to the location-independent access information of a PID that allows the verification of datasets to which the PID is pointing to. This addresses *Problem Statement 4*. We create a cryptographic-aided chain of trusted data access that consists of a signed PID and verification information for PID-tagged downloaded data. This allows attributing PIDs and disseminated data to trusted parties and disconnecting the authenticity of PIDs and research data from their network location. Our approach for trusted data access is described in Section 6.3.2.

3.) Long-term access for resources accessible with location-independent networks is a new benefit we add to the current state-of-the-art. The focus of location-independent data access technologies, such as BitTorrent or NDN, is currently not on providing long-term data access. Using the approaches we present in this chapter, data sets that are (only) reachable through location-independent network technologies can be tagged with a PID. By this, (large) datasets from the research community stored in next-generation data repositories using BitTorrent or NDN for research data dissemination can be retrieved in conjunction with Handle PID (cf. Section 4.6, *Academic Torrents*). This allows citing of research data sets in publications applying this new dissemination technology.

6.2 Distribution of PID Maintenance Efforts

As described in Section 2.3, the PID durability and persistency is an effort of the PID infrastructure operators which use advanced software systems, policies and social ecosystems to achieve this goal. However, these efforts are only the minor part to have a working PID functionality. The other part of the PID maintenance effort is on the side of the data owners, as they have tagged research data with PIDs in the past. In contrast to the PID infrastructure providers, they have the deep view on the research data. In the best case, they know the content of the PID-tagged data and have expert knowledge on the data and the associated metadata, but most important, they know the current location of the data set in the network. Thus, the data owners or a succeeding organization are responsible for keeping the PIDs up to date. Hence, the verification and adjustment of the target URL is the most important part of the PID curation process. As a result, data owners have to accept the burden of updating and checking the PID regularly. These are the resource consuming parts of research data curation and dissemination using PIDs, as they are staff-intensive and require individual processes for each organization. At this point, it is important to mention that non-technical factors decide on the success of research data curation such as sufficient founding, qualified staffing and well-designed infrastructure strategies [4]. Hence, the PID curation efforts and the resources spent for PID checks and updates may become the limiting factors in the future. Therefore, it is interesting to see whether the PID related processes and infrastructures for research data management are scaling out sufficiently on the side of the data owners.

In the following, we have a look at the user statistics for the DOI service in order to estimate the current effort for PID maintenance and its future development. As an example for the usage and growth of a real-world PID infrastructure, we consider the DOI system and evaluate the statistics from DataCite, one of the largest PID infrastructure providers for DOIs [152]. In Figure 6.2, the numbers of PID assignments and unique successful resolutions for the DataCite infrastructure are visualized for the time span between 11/2011 and 11/2015. In the line chart of Figure 6.2, a massive increase of PID assignments can be observed which follows a super-linear pattern. Thus, we can conclude that the usage

of PIDs is rapidly increasing and hence the effort for PID maintenance is growing, too. With the digital transformation of science, the importance of persistent identification will grow, leading to a stronger increase of PID assignment. For underlining our argument, we added the aggregation of the successful DOI PID resolutions to Figure 6.2 [153] [154]. This figure shows a massive increase in PID-tagged data sets that have been requested by users. The increases indicate that the role of PIDs becomes more important and follows the trend line of research data volume increase. Thus, efforts on PID maintenance will follow the similar patterns and thus, the contributions of this thesis will become even more relevant to practitioners who are looking for intelligent alternatives to error-prone target URLs.
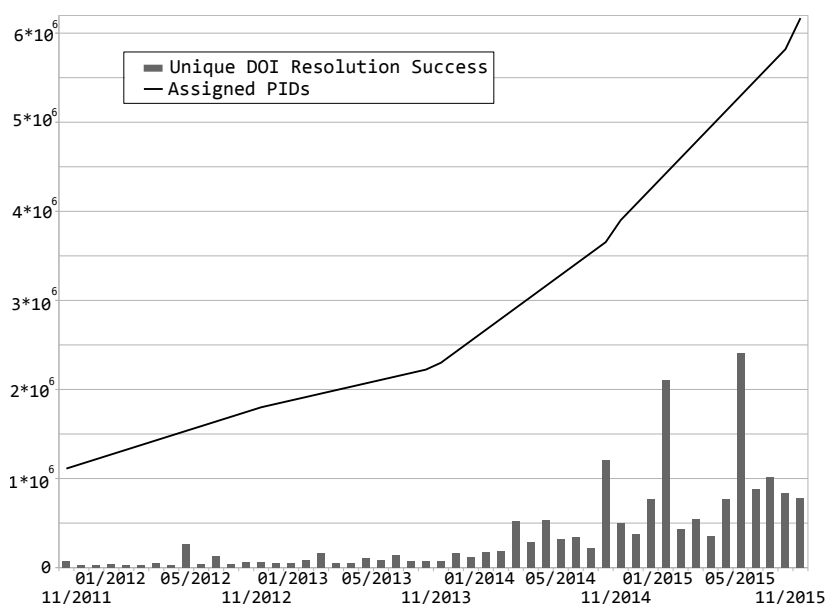

Figure 6.2: PID assignment and unique successful resolution for the DataCite DOI infrastructure between 11/2011 and 11/2015 [152] [153].

6.3 Approach

Our approach combines the very stable concept of Handle PID with the advantage of location-independent data access by using a stable access information scheme. In this way, we reduce the effort for PID maintenance for the data owners by taking advantage of persistent resolution targets that do not require adjustment as they are independent from the data locations. Hence, our approach reduces the effort for PID maintenance as regular checks on the side of the data owners concerning data localization and availability become obsolete. PIDs that are extended by location-independent access information remain valid as long as data is online through a location-independent technology like BitTorrent or NDN. Thus, physical data availability in the network (which is the foundation for data access) is the only prerequisite for persistent data access and needs to be checked, while

the content of PID remains untouched. For data owners, checking the online availability of data is a standard software for service monitoring and is part of a monitoring solution that can be found in almost every (large) IT-installation. For storing location-independent access information in existing Handle PIDs without impairing existing PIDs and their infrastructure, we extend the non-standardized approach of the Magnet Link scheme. Magnet Links have already become very popular within the file sharing community, but need further extension into the domain of research data dissemination and the NDN domain [47]. To emphasize the advantage of our approach, we do not have to modify the Handle PID system, which allows a practical integration into the existing Handle system, but our approach only adds little overhead to PID resolution, as we show in Section 6.5. Additionally, our approach also contains an extension of the Magnet Link scheme into the domain of NDN. By this, we provide a novel transport container format for NDN data access that does not only contain the NDN data name but also cryptographic access information which supports the verification of data authenticity from arbitrary NDN data sources like mirror servers or CS in NFDs. As a result, Magnet Links with NDN access information can be used in other media like websites or E-Mails.

6.3.1 Magnet URI Scheme Extension for NDN

In Section 2.5, we already introduced the Magnet Link URI scheme as being capable of handling peer-to-peer network technologies like BitTorrent for location-independent access. In this subsection, we extend the Magnet URI scheme int the domain of NDN. In contrast to the existing simple URI-based NDN name conventions [63], our contribution to the Magnet URI scheme is not limited to the identity of digital objects in a NDN network through a name but it is also able to store auxiliary content information such as alternative data names, object size and content checksums. Due to missing standardization and centralized development (e.g., through a technical board), it lacks in adoption from the latest developments in URN. Hence, we treat the Magnet URI scheme as a variant of URN, but not as a URN-compliant adaption.
By looking at Section 2.5, we recall that a Magnet URI has the following form:

```
magnet:?xt=urn:<System>:<Access Information>,
```

where `magnet:` is the URI scheme for a Magnet Link and all subsequent keys after the "?" contain information about the digital object in the form of a key-value dictionary concatenated by the "&" character. These keys may contain location-based access URLs or descriptive information that is needed to access a digital object independently from its location.

For encoding location-independent access information using NDN, we provide a xt-key extension that is announcing a Magnet Link URI against a scheme handler for retrieving

data over a NDN network. It provides NDN access information for the digital object within the NDN space.

The access information xt-key extension we provide for NDN is

```
xt=urn:ndn:<DATANAME>,
```

where `<DATANAME>` is a NDN data name according to the NDN namespace conventions [63] pointing to the digital object.

In Table 6.1, we provide an overview of different Magnet URI scheme xt-keys that are assigned to different content exchange platforms. Our contributions for NDN data access in row five is printed in bold fonts. In the following subsections, we will also in work with the BitTorrent xt-keys (4) and provide more xt-key extensions for realizing a trusted data dissemination through PID.

Nr.	System	URN	Value
1	Gnutella2	`sha1`	file hash (SHA-1)
2	Gnutella2	`tiger`	file hash (Tiger Tree Hash)
3	Kazaa	`kzhash`	file hash (proprietary)
4	BitTorrent	`btih`	unique file identifier (info hash)
5	**NDN Access**	`ndn`	**data name**

Table 6.1: Magnet URI Scheme Extension (in bold letters) [51]

6.3.2 Magnet URI Scheme Extension for Trusted Data Access

For trusted access in location-independent networks such as BitTorrent and NDN, the principle that trusted network locations serve trusted data does not hold any longer. Hence, we have to add cryptographic data to the access information which allows a direct verification of the data. For this, we propose an approach for a long-term data access trust chain that uses PIDs. The PIDs are used in the trust chain for the following purposes:

1. Bootstrapping the trust chain by using the PKI of the Handle system that provides a set of pre-installed certificates from trusted parties.
2. Transporting the access information of the actual research data.
3. Transporting the cryptographic signature of the research data.
4. Transporting the access information for obtaining the certificate for research data signature verification.
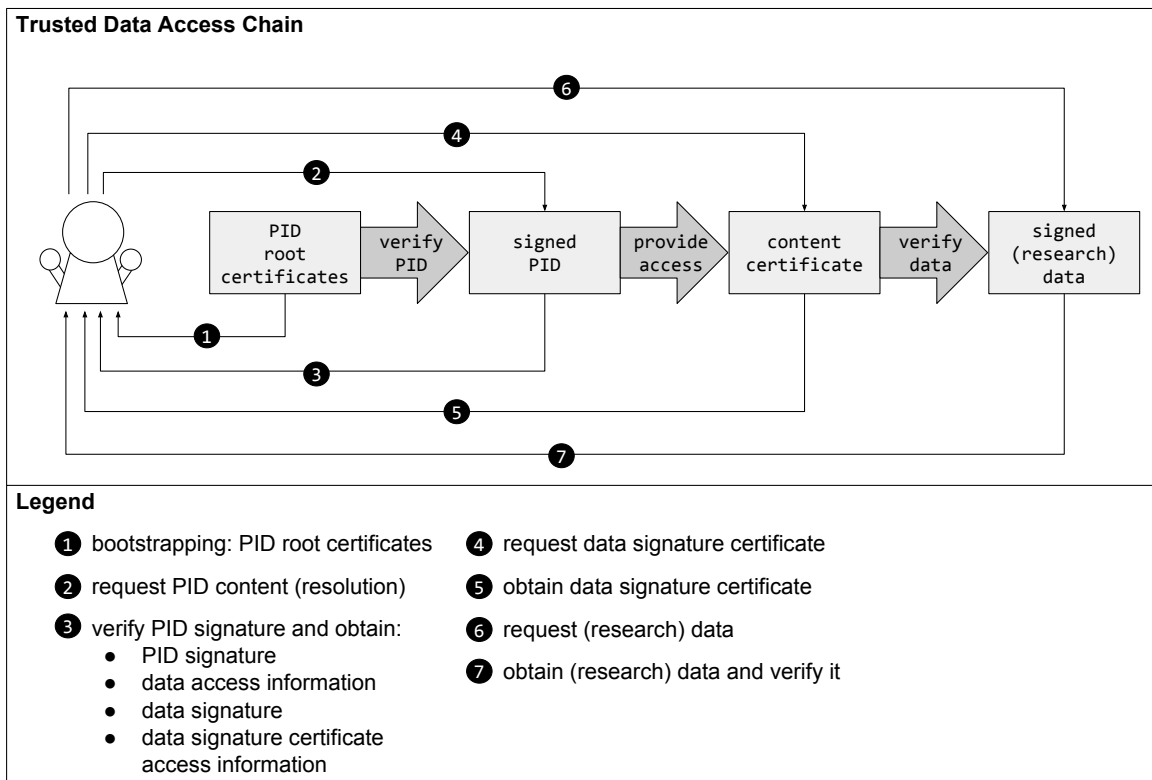
Figure 6.3: Trusted Long-Term Location-Independent Access Through using PIDs

In Figure 6.3, the trusted access chain is depicted. It assures that data has been created or published for dissemination only by (known) trusted parties. In the step ❶, the user obtains a set of pre-installed certificates belonging to trusted PID infrastructure providers. This set of trusted certificates can be included into the PID software package. In step ❷, the PID is requested for resolution.

The result of the PID resolution process is returned in step ❸. Now, the user obtains the Magnet Link stored in the PID together with the PID signatures which has been created by the PID creator. The certificate for verifying the PID signature can be obtained from the LHS (e.g. through NDN PID pull) and is signed by one of the owners of the pre-installed certificate set. With this information, the genuineness of the PID content can be verified. The Magnet Link in the PID contains:

- data access information
- data signature
- data signature certificate access information

In step ❹, the certificate for verifying the data signature is requested from the location-independent network using the access information in the Magnet Link. After obtaining the certificate in step ❺, the validity of the certificate can be verified either using

the pre-installed Handle certificates or any other established Internet PKI. In step ❻, the user requests the (research) data from the location-independent network using the access information in the Magnet Link. After obtaining the data in step ❼, it can be verified using the signature stored in the Magnet Link and the data verification certificate obtained in step ❹ and ❺.

The Magnet Link data plays a significant role and hence, we provide an extension that adds xt-keys to store the data signatures and the access information for obtaining the certificate. Table 6.2 shows the extensions in bold fonts:

Nr.	System	URN	Value
1	BitTorrent Access Information	`btih`	unique file identifier (info hash)
2	NDN Access Information	`ndn`	data name
3	**Content Certificate NDN Access Information**	`ndncert`	data name
4	**Content Certificate BitTorrent Access Informations**	`btihcert`	unique file identifier (info hash)

Table 6.2: Magnet URI Scheme Extension for Trusted Data Access

For trusted PID access, we need to encode the cryptographic signature into the Magnet Link as well. For this, we propose a new key type `cs` (`content signature`). Within this cs-key, a URL-encoded representation of a base64-encoded cryptographic signature is stored. In Figure 6.4, we provide examples of Magnet Links that use URL-encoded cryptographic signature computed with OpenSSL Link [155]. The signature adds 256 bytes [75] with an overhead of estimated 15% for the base64- and URL-encoding to the Magnet Link. Our experiments in Appendix A.5.4 show that we can expect a growth of Magnet Link size by 741 characters for adding a signature (737 characters average signature size and 4 characters for the "`&cs=`" type key preamble). We can also formulate a Magnet Link for trusted BitTorrent access:

```
magnet:?xt=urn:btih:b415c913643e5ff49fe37d304bbb5e6e11ad5101\
&dn=Ubuntu+14.10+desktop++x64\
&xt=urn:btihcert:13d22ec551069369502a3100a99b991dd56389d4\
&cs=Xis9OH1azSR0bJSKiJpPdrMSINVQdVA4AQ5IxSnV2iTe088D3Tgq1bMA\
SP5czqQDeQd%2ByLtPOtz4%0Ai5oAv%2B8HLSopD71N8HFTPyVONQZeabIGT\
IvNEIWQREicuAKA7tBZF%2BTpcY8dWtUBVGAceM%2FXPyZ4%0AdCJPehuHdC\
SyWMSesHLHfWsm5gGLGTVJ5zkjRcCVCpmnBxQ%2B0fbXLVYMVLw%2BYNA8Jz\
NzIeA%2Bx9M1%0Abe4mF%2BFFUev%2F01FxaqZCsA8bQHXrxaDgxrwC5W0JZ\
ggC5r0vLALR78bU0LpG2alp%2FYwTLziRBX1z%0AyrW%2BSGHmcSoWFy131L\
hNgZAULDGdnND1uvZXsvalAHGb070qXs0bnBabSgLtYBv4dwa42%2BUHDmit\
%0A%2BJWz4AuuOb3%2FOu2qGnfhBHc5%2BKtn1R1N2%2Fxy0PbOiFYSaYqWE\
GuHgA6zgGyP4k9vnylIZWnEDTnR%0A3kFuYik4bgoaPoptZQs0I%2FbZrqKp\
sbvb2vW%2BBQwUcT49uQSPkXEX0EKdXWpBmbr3Qj0bUosiwyzP%0ADWdBB0z\
MuDyJnK8PMW7gDkkfdDFNconsWnbC2GmMDWDV%2BnW14yc3plw6NMlqcmCVB\
viQk4ISqFfr%0Avzqt4GdJcvY65nEhzEfiF%2BG1ZZuZx8JN4WEfgNN07CBY\
G9uJcQCC7zKBWtWRgyHeAJ51%2FhsRHC8%3D
```

Figure 6.4: BitTorrent Magnet Link with Verification Information for Trusted Access

Furthermore, we can formulate a Magnet Link for trusted NDN access:

```
magnet:?xt=urn:ndn:/com/ubuntu/current/Ubuntu1410desktop.iso\
&xt=urn:ndncert:/com/ubuntu/certificates/isocert.pem\
&cs=Xis9OH1azSR0bJSKiJpPdrMSINVQdVA4AQ5IxSnV2iTe088D3Tgq1bMA\
SP5czqQDeQd%2ByLtPOtz4%0Ai5oAv%2B8HLSopD71N8HFTPyVONQZeabIGT\
IvNEIWQREicuAKA7tBZF%2BTpcY8dWtUBVGAceM%2FXPyZ4%0AdCJPehuHdC\
SyWMSesHLHfWsm5gGLGTVJ5zkjRcCVCpmnBxQ%2B0fbXLVYMVLw%2BYNA8Jz\
NzIeA%2Bx9M1%0Abe4mF%2BFFUev%2F01FxaqZCsA8bQHXrxaDgxrwC5W0JZ\
ggC5r0vLALR78bU0LpG2alp%2FYwTLziRBX1z%0AyrW%2BSGHmcSoWFy131L\
hNgZAULDGdnND1uvZXsvalAHGb070qXs0bnBabSgLtYBv4dwa42%2BUHDmit\
%0A%2BJWz4AuuOb3%2FOu2qGnfhBHc5%2BKtn1R1N2%2Fxy0PbOiFYSaYqWE\
GuHgA6zgGyP4k9vnylIZWnEDTnR%0A3kFuYik4bgoaPoptZQs0I%2FbZrqKp\
sbvb2vW%2BBQwUcT49uQSPkXEX0EKdXWpBmbr3Qj0bUosiwyzP%0ADWdBB0z\
MuDyJnK8PMW7gDkkfdDFNconsWnbC2GmMDWDV%2BnW14yc3plw6NMlqcmCVB\
viQk4ISqFfr%0Avzqt4GdJcvY65nEhzEfiF%2BG1ZZuZx8JN4WEfgNN07CBY\
G9uJcQCC7zKBWtWRgyHeAJ51%2FhsRHC8%3D
```

Figure 6.5: NDN Magnet Link with Verification Information for Trusted Access

6.3.3 Embedding Magnet Links into Handle PID

For storing location-independent access information in Handle PIDs, Magnet Links need to be embedded with a maximum compatibility. This is necessary, as research data dissemination has a very slow change momentum, owed to billions of existing PID-tagged data sets. Therefore, we investigate the impact of embedding Magnet Links into the Handle system. The Handle PID format stores data records hierarchically with indexed typed key-value pairs (cf. Section 2.4 and Figure 2.5). As Magnet URI can be encoded into a valid UTF-8 string, they can be placed into a Handle record without any conversion or encoding [38]. The Handle System contains hard-coded (also known as "registered") data types starting with the prefix `0.TYPE/` like `HS_SITE` for the records containing LHS access information. The full list of hard-coded data types is available in Appendix A.1.1. All `0.TYPE/` data types are needed to operate the Handle System or are data types that have a strong impact on use cases of the Handle system like `0.TYPE/URL`. The native support of URLs as data type is a property that is in common with other PID systems like PURL [156]. For implementing the Magnet URI scheme, a possible hard-coded data type in the Handle code base could be the data type `0.TYPE/URN`. But although CNRI claims support for it [157], the current Handle Library 8.1.1 has removed native URN support from the code base for unknown reasons (cf. Appendix A.1.1).

However, besides those hard-coded data types, user-defined data types can be assigned at any Handle record. By this, the LHS can be extended to store more data types than the hard-coded types. As Handle systems replicate on data record base and each record contains its type, custom type definitions are spread within all LHS hosting PID data for a Handle prefix. For embedding Magnet URIs using a Handle record field with the type `MAGNET`, we can only stick to conventions in the LHS and employ a user-defined data type. The application of user-defined data types and a machine-readable description may be improved in the future, as PID data types are a current subject of research [158] [159]. As a result, using Magnet Links as Handle record data type `MAGNET` has to follow the rule of convention over configuration and we use the data type `MAGNET` for all resolution targets of a PID containing a Magnet Link.

6.3.4 Data Access Service Chain

Before looking at the details of maintenance and resolution for PIDs with persistent resolution targets, we first compare the data access service chains of our approach with the classic data access service chain of PIDs using location-dependent resolution targets. By this, we motivate the usage of Magnet Link encoded resolution targets and emphasize the advantages of location-independent data access through PID. For PID resolution in the data access service chains, we can either use the native Handle protocol or NDN PID push. Magnet Link-enabled PID resolutions using location-based network connectivity will be

discussed in detail in the next section. For this section, we abstract from the PID resolution details.
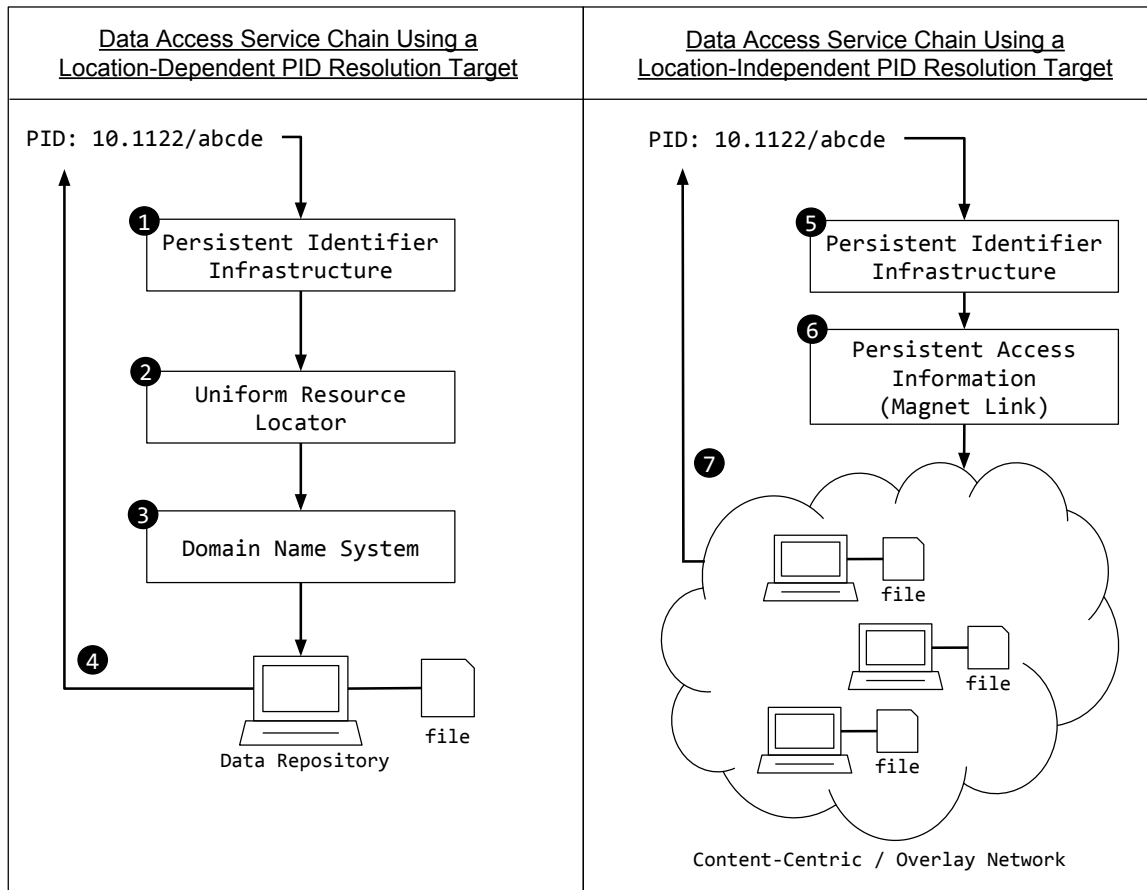

Figure 6.6: Comparison of PID Data Access Service Chains (based on [51])

In Figure 6.6, a comparison of the Data Access Service Chains is provided. On the left side, the classic Data Access Chain of location-based PID systems is presented. In this chain, a PID ❶ is sent from a *Client* to the *Persistent Identifier Infrastructure*, in order to be resolved into a target URL ❷. Then, the *Domain Name System* ❸ is resolving the Fully-Qualified Domain Name (FQDN) of the URL into the IP-address of the *Data Repository*. By connecting to the *Data Reposity*, the *Client* can retrieve the file. This location-based service chain features a double look up of PID to target URL and URL to IP-address. As a result, the file can be downloaded from a single source if the PID has not been equipped with multiple target URLs.

In contrast, the right side of Figure 6.6 depicts the data access service chain for our approach of PIDs with persistent resolution targets. The data access initially works identically for the *Client*, which sends an obtained PID to the *Persistent Identifier*

Infrastructure, in order to be resolved into a target URL ❺. Identically to ❶, the *Persistent Identifier Infrastructure* ❺ queries the GHR to determine the LHS. Then, the PID is sent to the responsible LHS for resolution. Now, in the case of this novel data access service chain, the resolution process does not return a target URL, but rather a Magnet Link in the Magnet URI format. With the access information acquired through the Magnet Link ❻, the localization and data access is now handled by the location-independent network part in ❼. The process of PID resolution works as a single look-up from the PID to the Magnet Link, which leads to a direct data localization and access after the resolution, instead of multiple redirections in a chain of services administered and operated by different parties. As the data access relies on peer-to-peer connections in the case of BitTorrent or on a multi-node connection in the case of NDN, no centralized infrastructure is involved after the *Persistent Identifier Infrastructure*, which is distributed in the case of the GHR and decentralized in the case of the LHS systems. Hence, the data access chain could be reduced in length and becomes more resilient, due to redundant decentralized infrastructure. As data access can be simultaneously sourced from multiple peers or NDN nodes, the advantages of redundant data access such as parallel downloading and failover are beneficial as well.

6.3.5 Creation and Maintenance of PIDs

Creating, maintaining and resolving PIDs that contain Magnet Links can be done like any other PID-record in the Handle PID system using the native Handle protocol for location-dependent access or NDN PID push for location-independent access. In these cases, creating and updating Handle records in PIDs requires the explicit statement of the data type for each Handle record [38]. Thus, there is no difference from the side of the Handle system between a Handle record containing a hard-coded or registered data type and a user-defined data type like `MAGNET`. To create and maintain the access information in Magnet Links stored in the PID, an additional service is needed that is aware of the location-independent access technology. This additional service works outside the Handle system and manages the PIDs with their resolution targets inside the LHS. The service consumes access information such as NDN data names, NDN data signatures or BitTorrent metadata and transforms them into our extended Magnet Link scheme for PID embedding. In the next two paragraphs, we explain the creation and update of Magnet Link-enabled PIDs in detail.

In Figure 6.7, the publication workflow for NDN Magnet Links is explained. It starts with the input research data and results in a Magnet Link-enabled PID, published data sets and a certificate available through NDN. In input stage ❶, there are the *Research Data* sets, the *Private Signing Key* and the *Certificate*, which contains the identifying data of the data creator (or publisher depending on the scenario) and the public key for content verification. In the next step, the workflow is then split into two stages ❷ and ❸. Stage ❷ generates the *Verification Items*, which allow verifying the integrity and origin of data obtained through

NDN. The operations *Sign Research Data* is IO- and compute intensive, as the whole research data set has to pass cryptographic algorithms. Hence, the generation of *Verification Items* is time and energy consuming, but only necessary if the research data is changed. As research data dissemination focuses on statically derived data, changes of data sets are not expected to happen regularly (cf. Definition 2.7). Stage ❸ makes all data available through NDN. This includes making the research data and the certificate available through a distinct NDN data name and reachable through an application protocol over NDN. It is comparable to publishing data on a web server in a location-based network. Stage ❹ contains now all items for the Magnet Link generation which are the cryptographic signature of the research data and the NDN data names. In stage ❺, the access data from stage ❹ is aggregated into the *Magnet Link* (cf. Figure 6.5). As a result, in the last stage ❻, the NDN access information can be included into a PID. Now, the research data and the certificate for validation are available for persistent access through the PID (cf. Figure 6.6, right side).
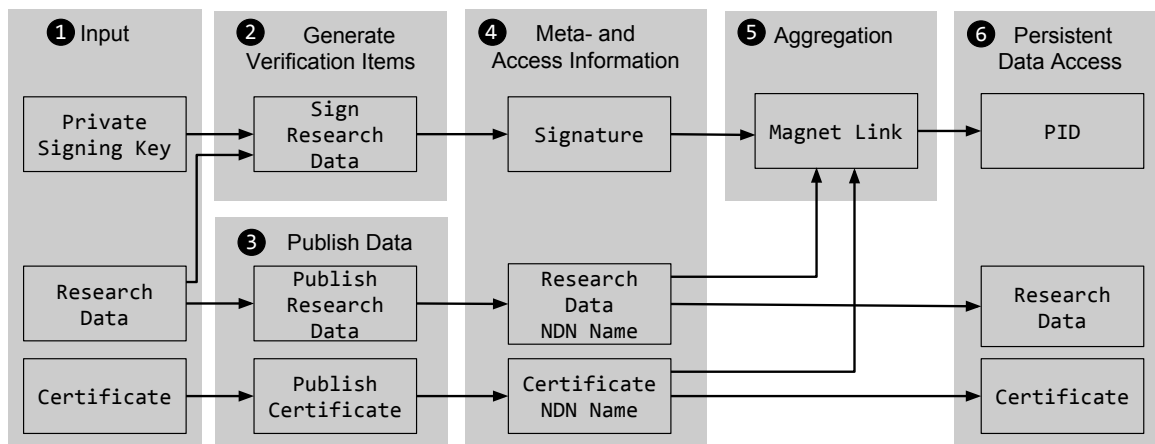

Figure 6.7: PID Publication Workflow for Magnet Link-enabled PIDs using NDN

The publication workflow for the location-independent overlay network BitTorrent is similar to the NDN workflow. For the details, we look closer at the publication workflow for BitTorrent in Figure 6.8. The input stage ❶ is identical to the NDN workflow. In stage ❷, the cryptographic *Signature* is computed over the *Research Data* using the *Private Signing Key*. After the completion of the signature, stage ❸ publishes the data. For this, the *Research Data* and the *Certificate* containing the public key and signed identity information of the data owner is made available through a BitTorrent software participating in a DHT with PEX. The access information of the research data set and signature are computed and stored in the *Bencoded Dictionary for Research Data* and the *Bencoded Dictionary for Certificate Access* (cf. Section 2.6.3) [48]. The dictionary information is published through the BitTorrent software in the DHT, in order to give other clients the possibility to obtain the access information using the DHT from other peers. In stage ❹, the info hashes of bencoded dictionaries are computed. Afterwards, in stage ❺, the info hash of the research data, the info hash of the certificate and the encoded signature are aggregated into a Magnet

Link, which is transferred into a PID. Data that needs to be online on the BitTorrent overlay network is depicted in stage ❻. This is the case for research data, the certificate and the bencoded dictionaries containing the access information for the research data and the certificate. Furthermore, the PID needs to be available in the Handle network space.
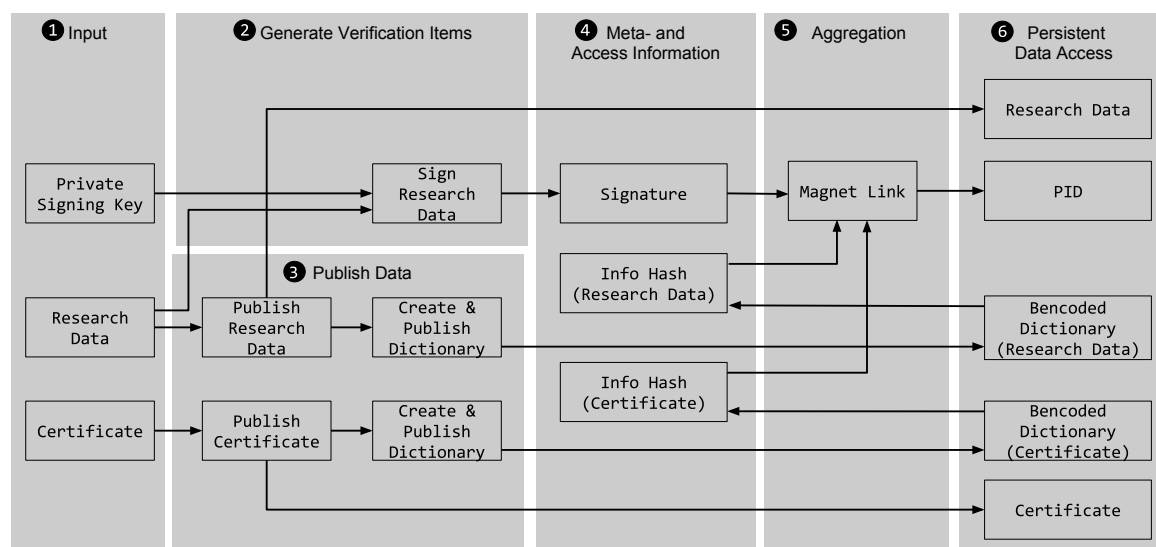

Figure 6.8: PID Publication Workflow for Magnet Link-enabled PIDs using BitTorrent

6.3.6 Data Access from Location-Dependent Networks

The resolution of PIDs with location-independent resolution targets does not pose a challenge when using the native Handle protocol or our suggested approaches based on NDN PID push and NDN PID pull. However, when using web-based resolution with Handle HTTP-proxies, new challenges arise for the resolution of PIDs with location-independent resolution targets. An overview of the web-based PID resolution process is depicted in Figure 2.8 and explained in Section 2.4. In this section, we provide an approach for resolving this kind of PIDs using web technology in order to grant access to research data from location-dependent networks.

By default, the Handle system and PID-related software rely on Handle data type `0.TYPE/URL` for resolving PIDs using web-technology. For resolving a PID using a HTTP client (e.g. a web browser), Handle's HTTP-proxy takes a PID and resolves it at the LHS [160]. Then, the Handle value with the lowest index and the data type `0.TYPE/URL` is chosen by the software as resolution target and returned to the web browser as a HTTP-redirection response with the HTTP status code `"303 - See other"`.

When using Magnet Links with the data type `MAGNET` in a PID without providing a `0.TYPE/URL` field, the HTTP-proxy software shows the content of the PID as a raw data

structure or as a landing page. In this case, the HTTP client (or web browser) does not know how to forward the Magnet Links to the application responsible for location-indepdent data access. To solve this problem two steps have to be added:

1. **Follow 0.TYPE/URL Handle values** as a PID HTTP proxy software and return the resolution target as a HTTP-redirection to the client using the HTTP status code `"303 - See other"`.
2. **Parse the Magnet Link** at the client side and invoke an application capable of accessing the research data through a location-independent network. Optionally coordinate the data verification using the signature and certificate information.

For solving the first problem, the PID HTTP-proxy software has to be extended to support `MAGNET`-typed Handle records as a HTTP forwarding target to the web clients. This does not require no modification to the core Handle system (LHS or GHR software).

For solving the second problem, we have to turn our attention to the client side. Web browsers as most popular web clients also need to support certain properties to pass Magnet Link information to BitTorrent or NDN client software. For this, the web browser must be able to support different protocol handlers that determine the behavior of the web browser when accessing different types of resources. These resources may even point outside the sphere of resources available through HTTP. As an example, we look at the protocol handler inside the browser that is responsible for handling E-Mail addresses embedded into websites as `mailto:` resources. If a `mailto:` URI resource is selected by the user, the information in the `mailto:` scheme is passed to the E-Mail protocol handler, which launches the E-Mail application and is able to preset the E-Mail receivers, the subject and the mail body [161]. For other applications, further URI schemes and respective handlers can be added to the web browser such as `tel:` for phone numbers [162]. As a result, a scheme handler for `magnet:` URIs needs to be present in the web browser. If a system should only download data from BitTorrent using the information in a Magnet Link, modern BitTorrent software offers a scheme handler for `magnet:`. But if more than one access technology is used within the Magnet Link, the `magnet:` scheme handler has to evaluate the Magnet Link content first using the URN content and then decide what application should be invoked and fed with the access information. After the `magnet:` scheme handler has passed the information to the location-independent access program, the download of the research data is done through location-independent network. The scheme handler can also invoke the download of the certificate and coordinate the verification of the obtained data at the client system.
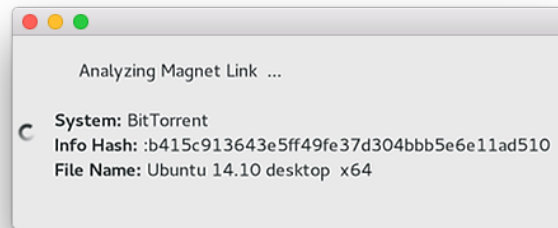
Figure 6.9: Magnet Link Protocol Handler

After clarifying the necessary additions at the server and client side, we look at the details of the data access to location-independent data using PIDs. In Figure 6.10, a simplified example for resolving a Magnet Link-enabled PID with a web browser is depicted as a sequence diagram.
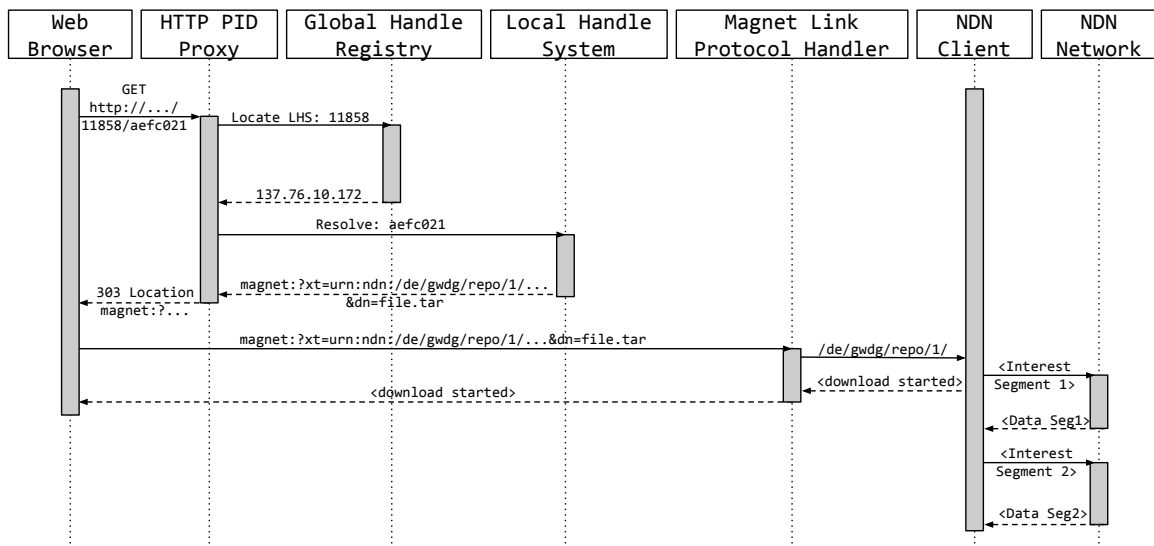

Figure 6.10: Web-based Data Access through PID using Magnet Links and NDN

The figure above depicts the invocation of a PID resolution from a click on a website link that contains the URL to the HTTP-based resolution service and a PID. After the click, following steps take place. First, PID is sent as part of a URL using a HTTP GET request to the HTTP-Handle proxy software. After this, it is resolved by the proxy software using the GHR and the LHS. This is done employing the native Handle, NDN PID push or NDN PID pull protocols. The HTTP-Handle proxy software extracts the Magnet Link access information from the PID and sends a HTTP-forwarding response "See Other" with the status code 303 to the web browser. By the 303 status code response, the HTTP-Handle proxy indicates to the web browser that the server does not own a (PID) target representation

that can be transferred over HTTP, as the data is available in the location-independent network space. Sending status code 303 is in line with the HTTP standard [163]. The HTTP status response contains the Magnet Link as HTTP-header, which is processed by the `magnet:` scheme handler which is invoked by the web browser. After analyzing the Magnet Link content, the scheme handler launches the NDN application and passes the NDN access information as NDN name for the digital object `file.tar` to the NDN client. The NDN client starts downloading the segments of `file.tar` form the NDN network. If the Magnet Link would contain `ndncert:` *xt-keys* pointing to a certificate, the NDN network can be used for obtaining the certificate as well. For Magnet Links containing BitTorrent access information, the data access as depicted in Figure 6.10 looks similar but the last stages of obtaining research data and optional certificates for content verification are backed by the BitTorrent system.

6.4 Implementation

In order to evaluate our approach of PIDs with persistent resolution targets, we implement an entire stack of software that is able to realize our approach as a proof-of-concept. The software stack is capable of the following tasks:

- Create and Update PIDs with NDN and BitTorrent access information
- Resolve Magnet Link-enabled PIDs using HTTP

In the following sections, we briefly describe the implementation for the main components. The screenshots of the user interface can be found in Appendix A.5.1, as well as the source code in Appendix A.5.2.

6.4.1 Server Side

First, we describe out the implementation on the server side that is necessary to realize our approach of Magnet Link-enabled PIDs and afterwards, we outline the implementation and prerequisites on the client side in the subsequent section.

1. Local Handle System: As pointed out in Section 6.3.3, the Handle PID system does not require any changes to store Magnet Link-enabled PIDs. Hence, the LHS hosting the Handle prefix and storing the PIDs does not need any modifications in its source code. By this, it remains compatible to entire Handle stack. It is also compatible with legacy LHS running an older version of the Handle software stack.

2. Web Service (PID-Burner): For creating, maintaining and resolving PIDs with persistent resolution target, we implement a web service that provides a user interface for human interaction employing a web browser. We refer this service as *PID-Burner* in the following in order to have a distinct name for it. The full architecture is depicted in Figure 6.11 and described in the following. The service is implemented from scratch but incorporates third-party libraries (*libtorrent Python bindings)*) for BitTorrent access information extraction from *.torrent* files [164]. With libtorrent, bencoded info dictionaries can be extracted for creating Magnet Links, as well as file checksums and file names that are helpful for creating a human-readable description of research data access information. PID-Burner also features a Python library created by us to pack NDN access information according to our approach (cf. Figure 6.7). We built a further Python library that allows parsing and creating Magnet URIs according to our extension proposed in Section 6.3.1. The PID-Burner web application is implemented in Python using the Bottle Framework for realizing the web application [165]. For interacting with the Handle LHS, we make use of the EPIC-API v2 web service from EPIC that exposes the Handle API as REST interface [166].
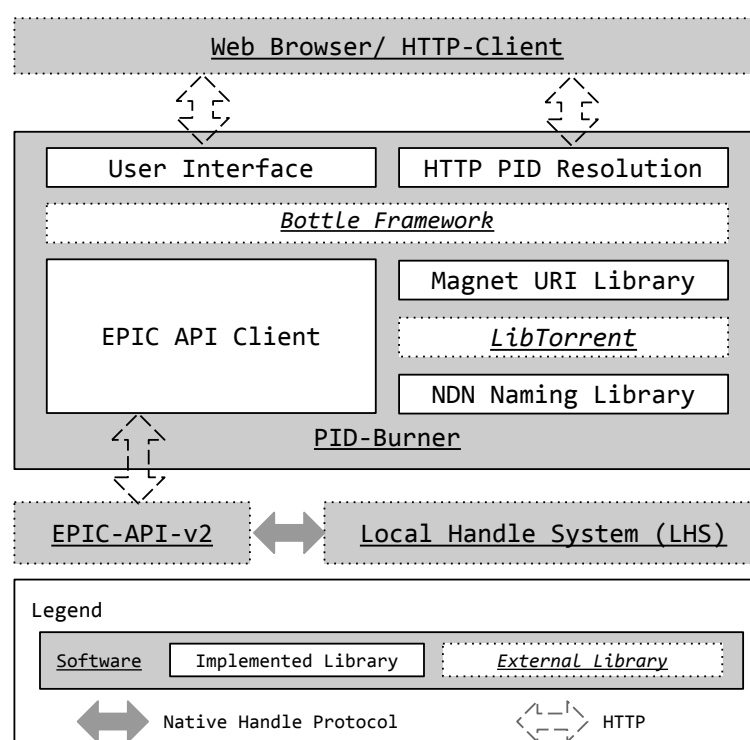
Figure 6.11: PID-Burner Software Architecture

For updating and creating PIDs with BitTorrent access information, the user has to generate a signature of the research data. This can be done with OpenSSL [155]. Then, two torrent files with a BitTorrent software capable of exporting info dictionaries into torrent

files have to be generated. One of these files belongs to the research data sets and a second torrent file belongs to the certificate. A BitTorrent software that is suitable for publishings file and export access information as torrent file is *Transmission* [167]. After exporting the torrent files, the signature file and the torrent files can be uploaded using the web interface of PID-Burner (cf. Figure A.1 in the Appendix). In Figure A.2, a screenshot of PID-Burner is provided that shows the user interface for the PID management. Furthermore, Figure A.3 and Figure A.4 show PIDs with Magnet Links containing BitTorrent access information (without trust information) in the official HTTP-Handle proxy running at http://hdl.handle net and on the official DOI HTTP resolver at http://dx.doi.org.

For uploading NDN access information, the information can be added via a web form in the user interface. In this web form, the NDN data name of the research data set, the data name of the certificate and the signature file can be uploaded. While the generation of BitTorrent access information does not require user interaction, the NDN data names have to be determined by the user depending on the NDN topology information. The topology information can be found out by the NFD running on the host that is responsible for serving the research data and/or the certificate through NDN.

For the PID resolution using HTTP and a web browser, we add a HTTP resolution interface to PID-Burner that is almost identical to the PID resolution capabilities of the HTTP-Handle proxy by CNRI. It implements the approach described in Section 6.3.6 and resolves PID against the presence of a Handle record with the `MAGNET` data type. For PIDs only containing `*URL` data types, the resolution is done against web-based location-dependent resolution targets, in order to render full functionality for resolving PIDs with (advanced) persistent and (classic) non-persistent resolution targets. The behavior enables the important back-compatibility to existing Handle services.

6.4.2 Client Side

For using PID with persistent resolution targets, a software is needed that is able to invoke PID resolution and to access the data employing a location-independent network technology. A typical end-user environment on a desktop or mobile device can consist of a modern web browser for browsing PID-containing media such as websites or PDF-documents containing scientific publications, a BitTorrent and/or a NDN client software for accessing data sets. For simulating a common end-user desktop environment, OSX version 10.11.6 running on a MacBook Pro 12,1 (Intel i5-5287U CPU and 16GB RAM) is used in conjunction with a Google Chrome 52.0.2743.116 web browser and Transmission 2.92 as BitTorrent software [167]. For dispatching Magnet Links to suitable data access applications, a Magnet Link Handler is implemented in Python that dispatches the download information either to the BitTorrent software or to the NDN download client. It incorporates our proposed Magnet URI scheme extension. A screenshot is provided in Figure 6.9. As a private NDN network installation is not available yet, a public testbed is running at the GWDG consisting

of six nodes. For the NDN data hosting and downloading, the NDN Repo NG tool set is used that consists of file server and client programs, which also define a NDN file transfer protocol [168].

6.5 Evaluation

After describing the implementation, we evaluate our approach in this section. The evaluation is done from two different view points, in order to provide a complete analysis covering the interests of infrastructure operators, software designers and end-users.

The first evaluation aims at the assessment of the data access using a location-independent PID resolution target. It is done from the view point of the infrastructure side and includes the interests of individual Handle server operators, as well as the goals of the Handle software designers. For this, we will have a closer look at the expected PID size for Magnet Link-enabled Handle PIDs and its impact on the Handle PID infrastructure performance (cf. Section 6.5.1).

The second part of the evaluation is focused on an assessment of expected data access duration in the location-independent setting. For this, we take the view point of the Handle PID system end-users, the researchers, in order to understand the impact on user experience introduced by our solution. For this, we look at the PID resolution performance and evaluate the impact of PID size increase (cf. Section 6.5.2).

The approach of Magnet Links as location-independent access information uses (multiple) key-value pairs for storing all necessary access data. PIDs with Magnet Link persistent resolution targets store more information than classic PIDs only containing URLs as resolution targets and thus they differ in size. The Evaluation Questions (EQ) aim at the assessment of difference between those two kinds of PID regarding their size characteristics and the impact regarding PID resolution. Hence, we formulate two questions for the evaluation:

EQ 1: Is there a significant deviation between the size of PID resolution target URLs and the size of real-world Magnet Link collections?

EQ 2: Do PIDs with an increased number of characters have an impact on the PID resolution duration?

6.5.1 PID Size Increase

For answering EQ 1, we collect real-world data from Handle PID systems (LHS) and perform a calculation on all target URLs focusing on the character counts. By this, we can identify typical characteristics of large PID collections. We use again our evaluation

data set described in Section 5.5.2 and particularly look at the determination of the PID target URLs for large PID populations. The extraction of the PID target URLs is described in the next section. For answering EQ 2, we perform load tests measuring the PID resolution behavior on large PID populations with stepwise increased target URL sizes. By this, we can estimate the impact of PID growth on the overall behavior of the Handle PID system. The evaluation is provided in Section 6.5.2.

6.5.1.1 PID Target URL Determinitation

In Section 5.5.2, we described the data preparation of the input data for the evaluation. One step, which was only discussed briefly, was the determination of the target URL size for each recorded PID resolution (cf. Figure 5.36, ❺). As we need the target URL size of recorded PID populations for estimating the deviation between today's PIDs and Magnet Link-enabled PIDs, we have a closer look at this process. Figure 6.13 depicts the process of target URL determination of a PID. In step ❶, we obtained the PID resolution log data from the GWDG-operated Handle HTTP-proxy. While the log data contains the PID requested by the user, it does not contain the target URL. The target URL is transported as part of the HTTP payload and is therefore no part of the log data, which is only recording the user interactions, but not the data obtained by the user. Hence, we have to re-enact the PID resolution with a Handle Miner, which grabs the PID as input data (step ❷) and uses the GHR and LHS for extracting the target URL (step ❸ and ❹). After the PID has been resolved by the Handle Miner, the length of the target URL is calculated and written into the data sample.

In order gather a collection of meaningful PIDs, we first have to select a relevant PID collection. Without filtering, the complete set of observed PID contains 22,757,503 resolution requests. The complete re-enactment of all requests would take over 26 days if we assume a PID resolution rate of ten PIDs/second. Hence, we first have a look at the relation between the PID resolution requests and the associated Handle prefixes for selecting a PID collection. In Figure 6.12, the Handle resolution requests are grouped by their Handle prefix and sorted by the access number. The chart contains the top 200 prefixes and covers 86.76% of the 22,757,503 resolution requests. It follows a Zipf distribution similar to other content on the Internet [169]. As we can see from the chart, 49.30% of all requests belong to the top ten prefixes. For our further investigation concerning the determination of the target URL size, we focus on the top ten PIDs.

resolution
requests
4m

3m

2m

1m

↓1%

0

Rang 1   50   100   150   200
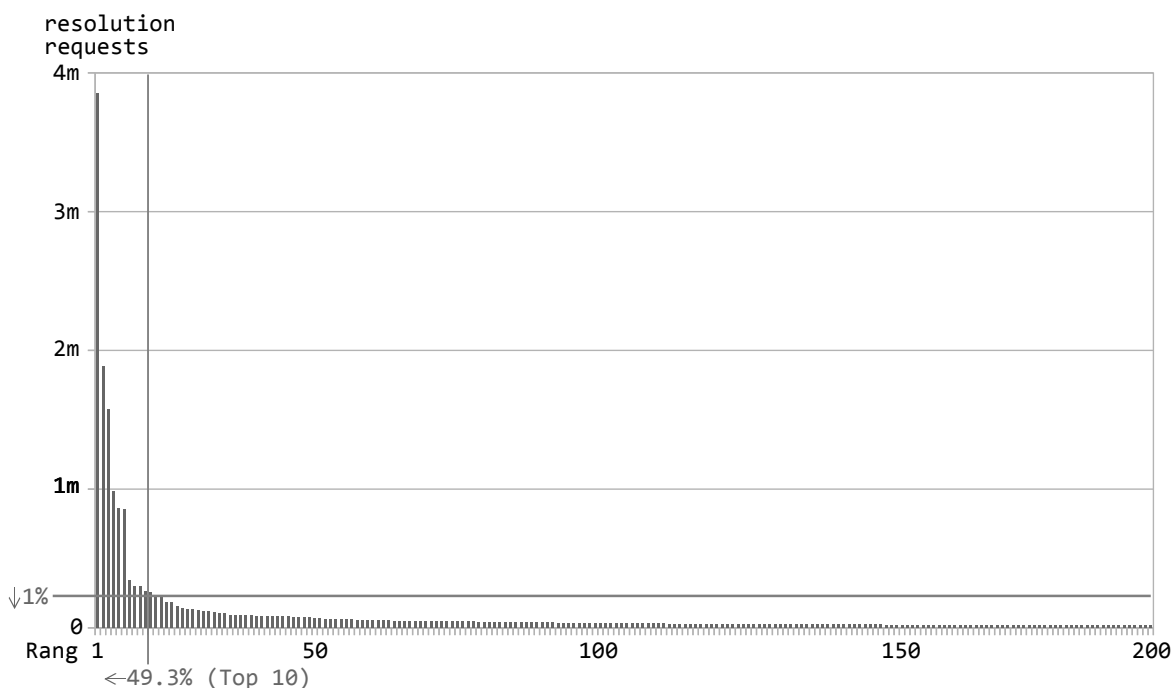   ←49.3% (Top 10)

Figure 6.12: Fragmentation of Handle PID Resolutions Grouped by Handle Prefix

For realizing the concept of PID resolution re-enactment, a specialized Handle Miner, called *Minera*, has been created (cf. Appendix A.4.1). Minera allows a high speed gathering of target PID URLs by employing a massive multi-threaded architecture. By this, up to 20 PIDs per second can be resolved and checked for a valid PID resolution. However, gathering the sizes of the target URLs from the top ten Handle prefixes presented in Table 6.3 took 6.5 days to complete which shows that the PID verification for large PID collection is not practical for billions of PIDs. Although this mining tool returns the full target URL, we truncate the target URL data using the script in Appendix A.4.2 into the target URL size by counting the characters. Afterwards, the target URL data is deleted and the Handle suffix is replaced by a salted hash to protect the user data (cf. Section 5.5.2).
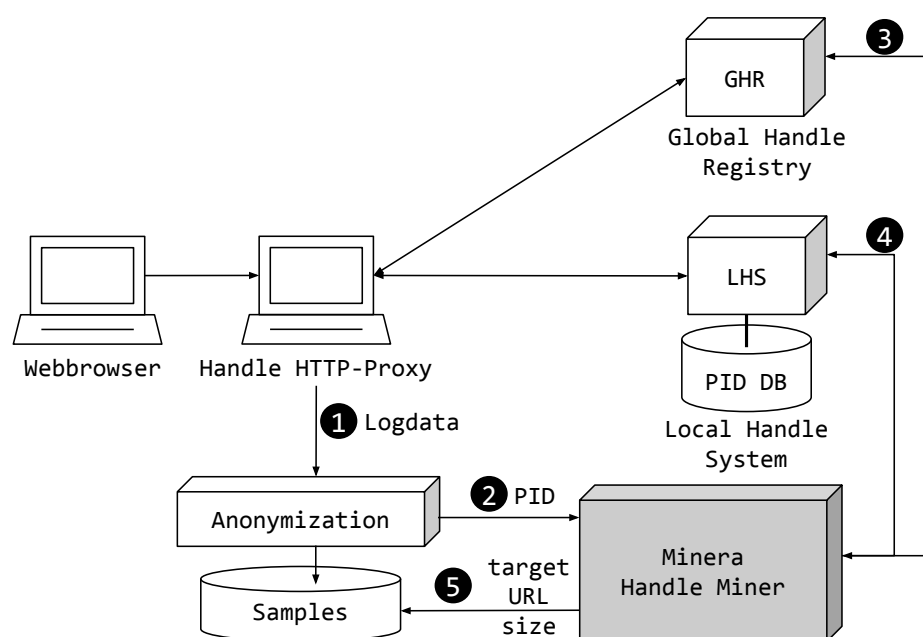
Figure 6.13: Determination of PID Target URL Size with Mining Software

6.5.1.2 Magnet Link Collections

In order to compare the target URL length of the existing PID populations with real-world samples of Magnet Link data, we a need large Magnet Link population created by humans.

Collection I – The Pirate Bay

As Magnet Links have their origins in the file sharing community, we take the Magnet Link population from the largest BitTorrent file sharing website, called "The Pirate Bay" that is publicly available on the Internet. From February 2012, a full dump of the Pirate Bay Magnet Link collection has been released on the Internet which contains 1,643,194 Magnet Links created by the users of this file sharing website [170]. For the analysis of the Pirate Bay data, we remove the tracker information from the Magnet Links, in order to have clean and comparable Magnet Links populations. Furthermore, the tracker informations are not needed, when using DHT for accessing files in BitTorrent.

Collection II – Academic Torrents

Another data source for comparison is provided by the website "Academic Torrents" which runs a distributed research data repository (cf. Section 4.1) [96] [97]. From this website, we have extracted all Torrent files using the publicly available REST API and converted them into Magnet Links with the same structure used for the Pirate Bay collection (cf. Appendix A.4.3). Potential tracker information has been removed from the Magnet Links in order to have identical structures. Thus, the Magnet Link structure for our comparison is as follows:

`magnet:?xt=urn:btih:<INFOHASH>&dn=<URL_ENCODED_FILENAME>`

Collection III – Synthetic Collections based on our Magnet Scheme Extensions
From the novel Magnet Link structures we proposed in this thesis, we can set up a collection
of minimal examples that shows potential impact on PID resolution. From the Magnet
Link structure above, we can conclude that the size of a Magnet Link only containing
untrusted information is determined by the content of the `dn` key (cf. Table 2.3), as the
info hashes have a fixed size of 40 characters (160bit hex representation). Additionally,
the requirements of URL encoding for every special character in the `dn` key extends the
size, too. The Magnet Link preamble has a fixed size of 20 characters. Hence, we can
build a minimal Magnet Link structure for BitTorrent access information using DHT, which
consists of 20 + 40 = 60 characters:

```
magnet:?xt=urn:btih:<INFOHASH>
```

A Magnet Link containing untrusted BitTorrent access information has the following
minimal structure:

```
magnet:?xt=urn:btih:<INFOHASH>&xt=urn:btihcert:<INFOHASH>&cs=<SIGNATURE>
```

The character count of this minimal example is
20 + 60 + 17 + 60 + 4 + 737 = 898

   As we can see from Table 6.3, the mean character counter of the target URLs is in the
range between 51.02 and 102.55. From the target URL collections, we can compute a
mean character count 78.21. For Magnet Links with NDN access information, we expect
a similar character count as we can see at the normal location-dependent URLs. This is
caused by the fact that the future NDN name space will have a hierarchical structure and
contain human-readable data names. As a result, we take the average character count
of $\lfloor 78.21 \rfloor = 78$ of the PID top ten target URLs as an estimation for the NDN data name
size. Hence, the structure of an untrusted NDN Magnet Link is:

```
magnet:?xt=<NDN_DATA_NAME>
```

The character count of this minimal example is
10 + 78 = 88

For a Magnet Link with trusted NDN access information, we have the following structure:

```
magnet:?xt=<NDN_DATA_NAME>&xt=urn:ndncert:<NDN_CERT_NAME>&cs=<SIGNATURE>
```

The character count of this minimal example is
10 + 78 + 16 + 78 + 4 + 737 = 923

6.5.1.3 Comparisons of PID Target URLs and Magnet Link Collections

After the aggregation of the data, we now estimate the impact on the PID system using Magnet Links within PIDs. For this, we answer the first evaluation question (EQ 1) and check if there is a significant deviation between the size of state-of-the-art PIDs with target URLs and real-world Magnet Link collections. In Table 6.3, the median, mean and variance target URL sizes is calculated of the PIDs from the top ten prefixes. We can see from the table that the median and mean for target URLs is below 100 characters in most cases. When looking at the variance, we can conclude that some PID owners use almost identical target URL structures for their PIDs leading to a very small variance. This is mainly caused by URLs that have a fixed structure with an alternating key pointing to IDs of research data sets or publications. For prefixes with a large variance in their target URL length, the structure differs. A reason for this may be a high amount of user generated URL parts like file names or the fact that the Handle prefix is used for different services at the same time operated under the responsibility of a single organization.

Rank	Prefix	Percentage in Sample	med. target URL	mean target URL size $\bar{x}_t$	var. target URL size σ
1	Prefix-A	16.91%	102	102.55	180.32
2	Prefix-B	8.27%	56	57.45	44.15
3	Prefix-C	6.91%	81	80.41	17.89
4	Prefix-D	4.33%	85	84.85	34.73
5	Prefix-E	3.79%	66	66.25	2.21
6	Prefix-F	3.76%	68	67.21	39.30
7	Prefix-G	1.51%	51	51.02	0.03
8	Prefix-H	1.33%	87	87.45	44.93
9	Prefix-I	1.33%	75	86.94	619.28
10	Prefix-J	1.16%	92	96.61	35.46

Table 6.3: Target URL Length of the Top Ten Handle Prefixes

We now look at the size of the Magnet Links as alternative PID resolution targets. In Table 6.4, the sizes of the Magnet Link collection are presented together with their calculated median, mean and variance. By looking at the median and mean, we realize that Magnet Links converted to our normalized structure of info hash and `dn` key are significantly larger than target URLs. When comparing the Academic Torrent Magnet Link collection, with the Handle top ten prefixes, the Magnet Links exceed the median by 183.49% and exceed the mean by 183.40%. When looking at the Pirate Bay Magnet Link collection the Magnet Links exceed the median by 125.82% and the mean by 123.25%. Moreover, the variance of Magnet Link collections is larger, which is caused by the user-dominated `dn` key containing user content such as file names or descriptions. Hence, we can conclude that

Magnet Links exceed the target URL size by a factor 1.2 to 1.8 for BitTorrent Magnet Links containing an info hash and a human-readable identifier encoded into a `dn` key.

Source	med. Magnet Link size	mean Magnet Link size $\bar{x}_t$	var. Magnet Link size σ
Academic Torrents	140	143.18	1324.62
The Pirate Bay	96	96.22	217.06

Table 6.4: Magnet Link Length of Academic Torrents and The Pirate Bay

In the next Table 6.5, the estimated Magnet Link sizes for the synthetic Magnet Link collections are depicted.

Collection	Estimated mean Magnet Link size $\bar{x}_{t'}$
Minimal BitTorrent Magnet Link	60
Minimal BitTorrent Magnet Link with Signature and Certificate Access Information	898
Minimal NDN Magnet Link	88
Minimal NDN Magnet Link with Signature and Certificate Access Information	923

Table 6.5: Estimated Magnet Link Length of Synthetic Collections

To provide a direct comparison between the existing collections of PID target URLs, existing Magnet Link collections and the synthetic Magnet Links collections that follow our contributions on NDN and secure content access, we show character counts in Figure 6.14. Estimated values are marked with an asterisk (∗). As we can see from the figure, Magnet Links without access verification information (minimal BT and minimal NDN) show similar size as the original PID target URLs. Furthermore, real-world Magnet Links collection (The Pirate Bay and Academic Torrents) provide an increase of the character count by 20% respectively 70%. The inclusion of content security measures with digital signatures and access information (minimal secure BT and minimal secure NDN) increases the character count by a factor of over nine. Despite this increase, we will see in the following that this has no impact on the PID resolution.
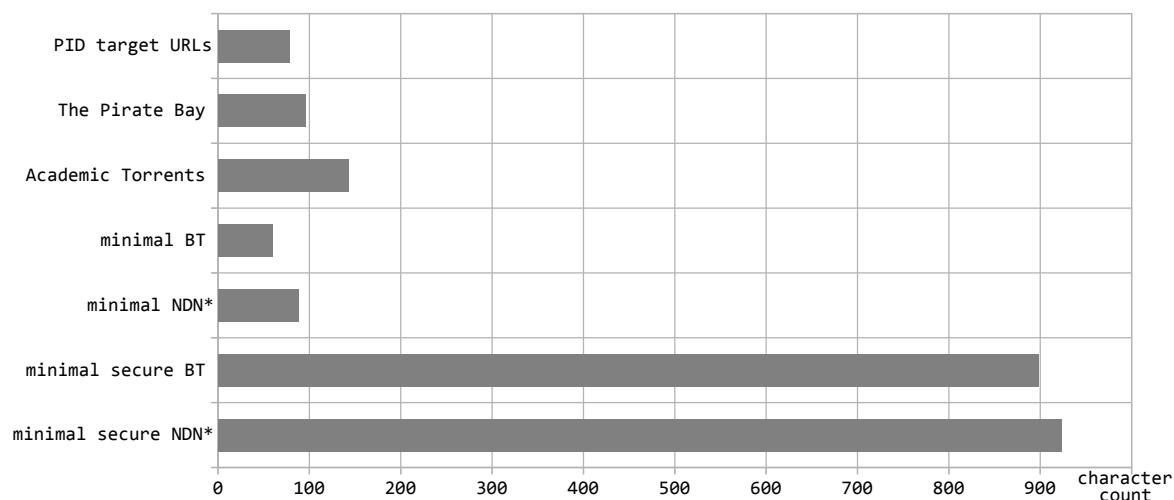
Figure 6.14: Comparison of Character Counts For PID Target URLs and Magnet Link Collections

In order to depict the distribution of the gathered datasets, we provide box plots in Figure 6.15. The figure shows the size of the equally structured Magnet Links and target URLs on the x-axis. The median for each data set is provided as a gray bar.
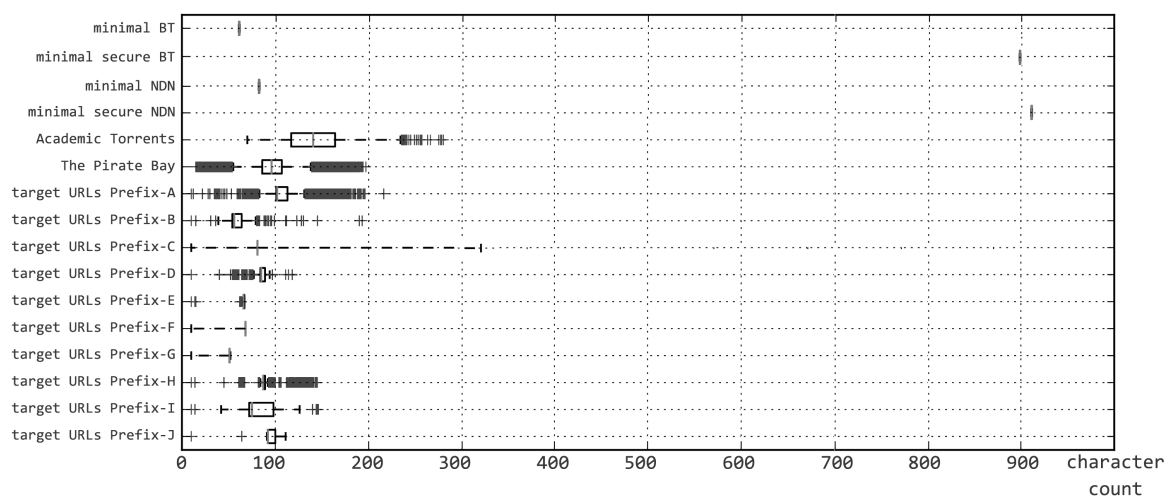

Figure 6.15: Distribution of Magnet Links and PID Target URLs Character Counts

6.5.2 Data Access Duration

Since we now know the expected size increase for PIDs with location-independent access information, we are able to investigate the impact on the Handle PID system. For this, we perform a sensitivity analysis to improve the understanding of the relationships between the PID size and the PID resolution performance in the Handle system. This allows us to answer the second evaluation question (EQ 2), which deals with the impact of increased PID sizes on the PID resolution duration.

For the evaluation, we use the LHS located at the GWDG data center that is responsible for serving PIDs which are located under the Handle prefix 11022. It is important for the evaluation to ensure that no data from fast caches is used, as this provides very short PID resolution times that are not expectable for average PID resolutions. For this, we have to assure that potential cached data in the Handle resolution stack is not employed for PID resolution. Hence, we explicitly suppress caches in the evaluation PID resolution requests by using an authoritative PID query type. Now, we can formulate the evaluation steps as follows:

1. Creation of 10,000 PIDs with a target URL of n random characters.
2. Record the time for the resolution of the PIDs created in step one by using one selected official Handle HTTP resolution proxy. For the resolution request, authoritative request types are selected, where the HTTP proxy has to resolve the PID using the LHS and is not allowed to use cached versions of the PID record.
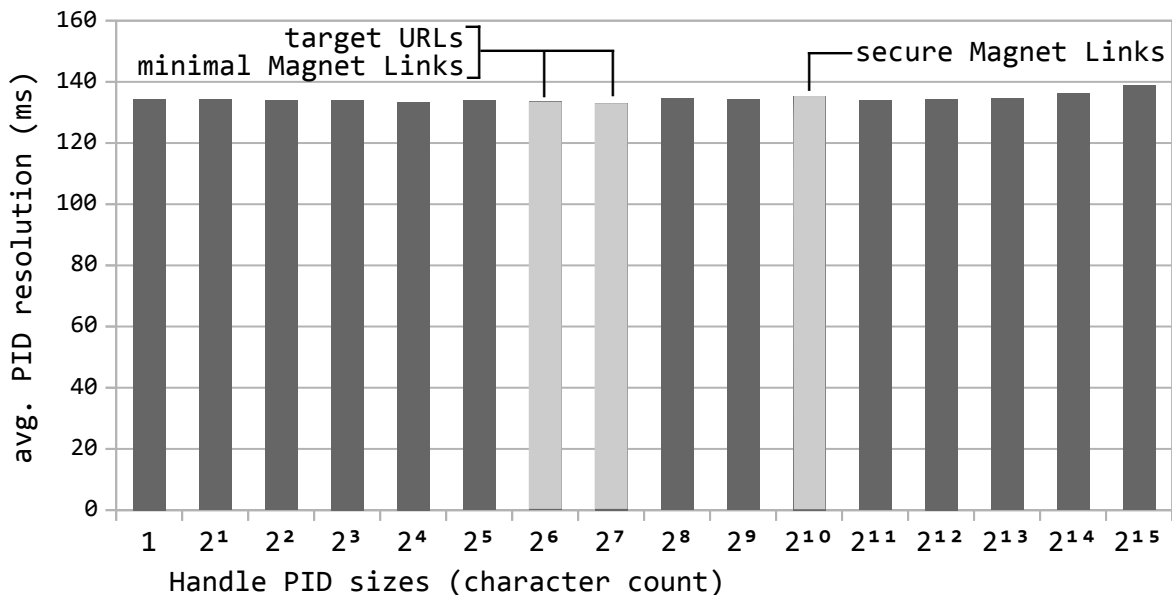3. Repeat the steps for a new PID set with an increased target URL size of 2^{n+1}.

Figure 6.16: Average Resolution Time of Handle PIDs With Different Target URL Lengths [51]

The complete source code for the evaluation can be found in Appendix A.5.3. In Figure 6.16, we see the averages of the PID resolution times for various Handle PID sizes. The measurements were conducted for each PID resolution sequentially using a server at the GWDG data center for executing the measurement script. The selected official Handle HTTP-proxy PID resolver was located at the Amazon EU cloud center in Ireland (Amazon Web Services Region eu-west-1), while the LHS is located at the GWDG data center. The x-axis shows the PID target URL sizes used for the evaluation with a maximum of 32,768 random characters. The y-axis shows the average resolution time for the PID using a HTTP PID resolution service. In the figure, we highlighted the bars for the mean PID target URL size and the minimal Magnet Link examples in light grey (2^6-bar and 2^7-bar). Furthermore, we highlighted the 2^{10}-bar that represents the character count of secure Magnet Links with digital signatures and certificate access information. For PIDs with an extreme target URL size of 2^{15} characters, we increase the PID resolution time in comparison to a short target URL of one character by only 5.5ms. Moreover, the figure shows that for some target URL lengths, the PID resolution time even decreases. This is caused by the reaction time of the database of the LHS and also by the underlying infrastructure of the LHS software, where we can expect an impact of the Java Runtime Environment and the operating system schedulers. Hence, if we summarize all results depicted in Figure 6.16, we can draw the conclusion that the PID size has no significant impact on the PID resolution. The effort for retrieving a PID value from the database in order to get the target URL is almost identical for different value sizes. This is clear as the data organization of the PID model is not changed, but only the amount data that needs to be transferred from the database to the Handle client is increased. Hence, we can conclude that the changes that we propose to PID for integrating complex access information do not have a significant impact on the PID resolution time. The difference of 10 ms seconds is not perceivable for end-users when resolving a PID. Hence, the answer the second evaluation question (EQ 2) is that PIDs with an increased size of the factor 1.2 to 1.8 has no perceivable impact for PID resolution. Hence, the integration of Magnet Links can be considered as a stable concept for referring to research data and publication stored location-independent networks.

Chapter 7

Discussion

In this section, we discuss the novel approaches proposed in this thesis. First, we discuss the results of the first approach on location-independent PID infrastructure using the evaluations conducted in Chapter 5. Afterwards, we discuss the results of our second approach on accessing location-independent research data access through PIDs using the evaluations of Chapter 6. In Section 7.1 and 7.3, we answer the research questions introduced at the beginning of this thesis. We conclude with the limitations of our approach in two separate Sections 7.2 and 7.4.

7.1 Answers to Research Questions Concerning Location-Independent Persistent Identifiers

In Section 1.2, we introduced the first research question (RQ 1) together with three subquestions. The answers are based on the results of proceeding chapters. RQ 1 queries if persistent identifier systems benefit in performance and resilience from integrating location-independent data access. We answered this question by re-enacting PID creation, maintenance and resolution operations in a simulated NDN test bed employing real-world data from EPIC and by comparing the results with the current state-of-the-art location-dependent equivalent applying a Handle infrastructure in the same simulation testbed. By comparing our NDN PID push approach with the native location-based Handle protocol, we learned first that a general operation of PID infrastructure is possible on top of NDN. From the evaluation of location-independent PID realization, we can draw the following conclusion regarding RQ 1. All Handle PID-related activities including the full Handle server administration can be performed with a location-independent NDN-based protocol. For a complete support of the native Handle protocol over NDN, a full adaption of the underlying application protocol layer is provided through our NDN PID push approach. While it has the advantage of connecting Handle servers and clients without having an a priori knowledge of their network location, it comes with the disadvantage of providing

less performance than the native location-based Handle protocol. However, it shifts the Handle PID system to a real identifier system without predetermined location for certain servers or vital parts of the infrastructure. When only focusing on location-independent PID resolution, the situation for performance and resilience is different. Our approach of NDN PID pull is able to provide location-independent plaint-text PID resolution and PID information retrieval that makes full advantages of the NDN design. In the domain of PID resolution and PID information retrieval, the strength of NDN provides superior performance, over the native Handle protocol. This is the case for intact networks with error-free connection, where the PID resolution and access times decrease only slightly, but especially for faulty network connections with random packet loss. For faulty network connections, the performance of NDN PID pull is much better than the native Handle protocol. For parallel PID resolution and access, the performance is further increased, as parallel PID resolutions are executed faster with a decreased load on the Handle server due to the network caching capabilities of the NDN, when compared to the native Handle protocols. Additionally, NDN PID pull provides a better outscaling behavior with an increased size of parallel connections. Thus, we can answer RQ 1 with *yes*, as NDN PID pull is able to deliver better performance in faulty network conditions. Limitations only arise in cases where PIDs have to be created, updated or maintained over NDN and thus are related to maintenance use cases. Here, the native location-based Handle protocol performs better. But, as the vast majority of the PID traffic between clients and server is related to PID resolution and PID information access requests, our contributions do not only release the Handle system from its location-based application design but also provide a significant performance increase with a better network error resilience. Thus, the Handle system could be improved for its most-prominent use cases of PID resolution and PID information access which cause almost all traffic on public Handle systems.

The subsequent research question RQ 1.1 focuses on the requirements for extending a PID system towards location-independent data access and operation. Concerning this research question, we found that a namespace convergence between the PID and the NDN space is necessary, in order to make PID and its attached data directly accessible through NDN as a global data name without a priori knowledge on the user side. Then, a Handle PID request can be directly routed to a server within the NDN network by only using the combination of Handle prefix and suffix. In order to make the conversion possible, restrictions for the naming of entities in NDN and Handle have to be considered. Additionally, the PIDs built on the application of this principle have to be human understandable for a convenient usage.

Research question RQ 1.2 considers the necessary end-to-end connection principle required by PID systems within NDN. We proposed a new end-to-end communication approach that employs NDN interests for data transport. Thus, we are able to connect arbitrary NDN endpoints by a data push mechanism and exchange data without a prior NDN connection establishment or using resource-consuming polling patterns. Using this approach, we are able to connect Handle peers directly through NDN, in order to perform synchronous communication that is important for maintenance operations, where changes

on PIDs and infrastructure topologies have to be immediately published to all responsible LHS servers.

Research question RQ 1.3 is related to the problem of providing operational and semantic interoperability between the location-dependent as-is PID infrastructure and our new approaches for location-independent PID infrastructure using NDN. To answer this research question, we provided an interoperability model that forwards PID-related information exchange between both worlds using a gateway. The gateway approach employs different mechanisms for selecting a forwarding protocol (NDN PID push or pull) for incoming and outgoing Handle requests. The protocol selection uses state machines for decisions and takes Handle-specific metadata from the application layer as decision input. Employing the conversion of the namespaces presented in this thesis, PID requests can be mapped to NDN resources and vice versa applying simple conversion rules.

7.2 Limitations Of Location-Independent Persistent Identifiers

The approaches for location-independent persistent identifiers presented in this thesis have limitations that are subject of future work. Some of them are caused by the design principles of the Handle PID system and the NDN technology. Hence, these limitations also occur for NDN adaption in various areas, as presented in the related work section (cf. Section 4.5). In the following, we discuss the limitations in detail.

1.) Namespace Convergence between the PID and NDN namespace is essential for the approaches presented in this thesis (cf. Section 5.3.2). Currently, no global NDN network exists besides larger test bed installations and as a result, no global namespace exists yet. This means that we can only formulate recommendations on a global NDN namespace that contains a PID system with its specific namespace. Therefore, we look at two scenarios.

In the first scenario, all Handle prefixes become NDN root names in the global NDN space (`/<HANDLE_PREFIX/`). In this case, the access of PIDs through NDN is easy and straight forward (cf. Section 5.3.2).

In the second scenario, PID prefixes are not part of the NDN root namespace. This case is not creating a problem either if we stick to the principles of PID-friendly NDN namespace design as formulated in this thesis and find an alternative root namespace (example: `/handle/<HANDLE_PREFIX/`) that includes all Handle prefixes (unified NDN PID namespace). If the Handle system operators are not able to agree on a unified NDN namespace, it is comparable to the present day situation, where end-users have to memorize different URLs (or NDN namespace prefixes in this case) to resolve a PID maintained by different PID infrastructure operators. As we perform a direct resolution and maintenance of Handle PIDs using NDN, a non-unified NDN namespace for the Handle PID system may result in a fragmented PID system. From this point, we can confirm our assumption that operating and evolving a PID system is not only a technological challenge, but also an organizational one.

2.) Lower data throughput for the NDN PID push protocol adaption can be observed in direct comparison to the native location-dependent Handle protocol (cf. Section 5.5.3). This is the case for PID creation (cf. Figure 5.38 and Figure 5.40), where NDN PID push is outperformed by the native Handle protocol. It is also the case for PID resolution, when comparing NDN PID push to the native Handle protocol (cf. Figure 5.41).

However, when looking at the approach of NDN PID push, we see that this protocol adaption for NDN does not make extensive use of the NDN features such as network caching and multi-sourcing. Thus, the overhead introduced by NDN is not compensated by the NDN advantages. But when considering the literature provided in the related work section 4.5.2, we see that this outcome is not surprising with regard to the current realization of NDN network technologies. A similar outcome can be observed for HTTP NDN adaptions, where the performance was behind location-dependent original implementations (cf. Table 4.1). We can also confirm with our approaches NDN PID pull and push that the impact of the overhead caused by NDN network technology is larger for small data portions transported over the NDN network. Thus, NDN PID pull that transports complete Handle values at once is outperforming NDN PID push that is transporting multiple smaller Handle message fragment. We expect better performance concerning network throughput and latency for future generations of NDN implementations. They might leave the domain of user land tools (cf. Figure 5.37) and will instead run as optimized kernel modules or as a native network component using a SDN approach.

3.) Real-world PID resolution patterns show the limitations to the advantageous feature of NDN's network caching capabilities. As we can see in Figure 6.12, a large fragmentation of the PID resolution requests grouped by Handle prefixes exists at the official Handle PID resolution HTTP proxies operated by GWDG. This pattern follows a Zipf distribution [169], which can also be assumed for other linked resources on the Internet, such as websites, downloads or PDF documents. This means that a small amount of Handle LHS that is responsible for all PIDs under a specific Handle prefix gets the majority of PID resolution requests. In contrast to this, the long-tail of LHS with all other PIDs related to less popular prefixes, have to perform very few PID resolutions. Consequently, a small percentage of PIDs is frequently requested, while a long-tail of PIDs is rarely or never resolved. If we assume such an access pattern is used for resolving PIDs employing NDN PID pull, we can conclude that a large percentage of PID resolution requests is not completed by a network cache hit, due to its low resolution frequency. As a result, the majority of PIDs cannot take advantage of NDN network caching. This affects all PIDs whose expected meantime of resolution requests by clients is higher than the longest cache expiration limit on the network path between the client and the LHS. Thus, we can assume that PIDs that are resolved less than once a month do not benefit from NDN network caching at all, as the NFD will remove the cached NDN PID pull resolution responses from its CS, before the PID is resolved again by another client. However, this observation for NDN is transferable to all NDN applications which are used to distribute content that is rarely accessed. It is definitely a disadvantage of NDN that needs better attention in the NDN community and is highlighted now by our approaches in the PID area.

7.3 Answers to Research Questions Concerning Location-Independent Data Access Using Persistent Identifiers

In the second part of this thesis, we looked in Chapter 6 at the integration of location-independent access information into PIDs. For this, we formulated the second major research question (RQ 2) of the thesis (cf. Section 1.2). Concerning the possible improvements of research data dissemination through location-independent networks, we can give following answers.

1.) Maintenance Free PIDs can be created using the approaches for PID improvement, we proposed in Chapter 6. As the PIDs do not point to changing data locations but rather describe the data that should be accessed through a smart connectivity, checking and adjusting the PIDs is not necessary anymore.

2.) Location-Independent Research Data and Publication Citation is enabled by our approach using existing Handle-based PID systems such as DOI. Even today, large research data sets from the National Aeronautics and Space Administration (NASA) and from high-energy physics community are distributed in next-generation research data repositories that use location-independent network technologies. With projects such as Academic Torrents, platforms for this kind of research data dissemination already exist and thus, our contributions add the missing piece for providing an approach to persistent citation of the shared research data. Additionally, this point answers the first subquestion (RQ 2.1) that asked whether a construction of a PID target scheme is possible that can coexist with the existing target URLs. In Section 6.3.3, we provided an extension of the Handle PID type system that allows a parallel installation of location-independent access information together with target URLs in PIDs. However, this extension of the Handle type system requires a modified PID resolution procedure that resolves PID against Magnet Link data type (cf. Section 6.3.6). Hence, we can answer RQ 2.1 with *yes, it is possible with restrictions concerning the modified PID resolution process.*

3.) Portable Trustworthy NDN Data Names are provided by our approach which extends the Magnet Link scheme into the domain of NDN. By this, we enable the transport of NDN data names together with its content-verification information in form of a cryptographic signature in a structured container format. This allows the embedding of NDN access information into media types such as websites and E-Mails. Furthermore, the concept of Magnet Link protocol handlers improves the software integration at the user side, as it enabled passing access information to suitable location-independent download applications. As a result, our contribution to the NDN Magnet Link extension makes a standardized handling of NDN data access information possible and exceeds the possibilities provided by the current URL-like NDN naming scheme. This creates the foundation for NDN-based (research) data dissemination.

4.) No major changes to the existing PID Infrastructure are necessary for the approaches we provide for the integration of location-independent data access into the Handle PID system. By this, we comply with the restrictions formulated in the introductory section (cf. Section 1.2), where we stated that PID systems have a very slow change momentum, meaning that existing principles and infrastructures can only be renewed over time, due to billions of existing PID-tagged data sets. With the evaluations in Section 6.5.2, we showed that no major impairment of existing PID infrastructure is expected for our approaches. Thus, we can answer research question RQ 2.2 by *yes, we do not expect an impairment of PID infrastructures, at least for PID systems based on Handle.*

5.) Safeguarding Data Access in Location-Independent Networks through PIDs was subject of research question RQ 2.3. The question was whether PIDs can help to support trusted data access in location-independent networks. We can answer this research question by *yes, PID can support trusted data access in location-independent networks using our approaches to integrate verification information into the data access information.* But in order to realize this, three prerequisites have to be in fulfilled. First, the datasets that are available for dissemination have to be cryptographically signed by the data issuer using a PKI. Secondly, the signatures have to be integrated in the PID access information (cf. Section 6.3.2). Thirdly, the PID needs to be signed as well, to assure the origin of the access information. With these three prerequisites in place, the data access follows a chain of cryptographic signed pointers that lead to a verifiable data set. The trusted data access in the location-independent network is then performed in the following steps. First, the PID origin and its data is verified. Next, the PID is resolved into the access information which contain the data verification information. As last step, after accessing the data, this verification information stored in the PID is used for verifying the dataset. By this, a fabrication of the data is inhibited and the trustworthiness is not bound to the data location, but only to a chain of trusted data descriptors and pointers bootstrapped through PID access.

To summarize, we can answer the second research question by *yes, we provide improvements for the research data dissemination using location-independent technology* on the side of the PIDs and on the side of the dissemination of research data with PIDs.

7.4 Limitations Of Location-Independent Data Access Using Persistent Identifiers

Although we have shown in this thesis that long-term location-independent data access information can be embedded into PIDs using an extended Magnet Link format, there are several limitations. These limitations can be overcome by future work that is not only subject of the Handle PID community but also subject of the communities advancing BitTorrent, NDN and the Internet standards (e.g. IETF, W3C). Hence, we identified the following limitations for our second approach presented in Chapter 6.

1.) Missing PID type standardization is currently a problem when integrating new data types like Magnet Links into a PID system. In the current as-is situation, an implementation of a new data type into a PID system is only possible if the PID system has an expandable type system. Additionally, with a missing standardization it is also not clear which data types are prospectively supported by the PID systems and where future development of a PID is directed to. Although this fact seems to be a technical problem first, it has two further implications. The first one is that the missing standardization is inhibiting the integration of services responsible for long-term access. As shown in Section 6.3.6, high-level service for PID resolution software like Handle HTTP-proxies use specific data types to determine their resolution behavior. If there is no agreement on alternative resolution data types, the interoperability between the high-level PID resolver and the PID software stack is defective. These integration problems hold also true for all types of high-level services using a PID system such as a data repository software or data catalogs that rely on a consistent PID type set. The second implication is that approaches to integrate new data types for PID resolution cannot be shared between different PID systems. This further implicates that advances in the long-term data dissemination cannot by offered by all PID systems. For data repository operators using multiple PID systems for data tagging, new dissemination techniques are only available if they are supported by all their PID systems.

Fortunately, the problem of a missing PID type standardization has been recognized by the research community. Currently, the international Research Data Alliance (RDA) is working on the PID type standardization in an official *PID Information Types Working Group* [171] [172]. The goal of the working group is to facilitate PID typing and enabling interoperability across PID systems. For this, the working group has created an API proposal, called *PID Info Types API* that contains a common minimal set of types which should be present in every PID system. Furthermore, it contains a software architecture for describing, querying and requesting data types using a data type registry. This data type registry manages, stores and contains type descriptions independently from the PID system families. By this, new data types can be attached to type registry-enabled PID systems and automatically requested by high-level services. Hence, the integration of new data types in different PID systems is doable in a controlled way and thus a cross-system integration of new access information schemes is possible, as well. Until the results of this RDA working group will be largely adopted by the different PID system communities, we have the limitation that the embedding of location-independent access information is possible for the Handle PID system but cannot be assured for all other existing PID systems.

2.) Missing Magnet Link standardization has been described in Section 2.5. However, we focus on the limitations when employing the non-standardized Magnet Links in location-independent research data dissemination in conjunction with persistent identifiers. On the side of the Handle PID system, the impact of an unstable Magnet Link format is not severe at the first sight. As pointed out in this thesis, Handle PIDs are self-contained and store their data in indexed isolated Handle values (cf. Section 2.4 and Figure 2.5). Thus, a change in the Magnet Link format specification has no immediate impact on existing PIDs stored in the PID system. Additionally, the PID resolution is not inhibited, as the resolution process returns the Handle value in return of a PID access. However, the data quality of the PID system is degraded by different formatted Handle value content, as the clients consuming the result of the PID resolution expect the data to be formatted in a specific way. Furthermore, a conversion of the existing Magnet Link-formatted Handle values may be necessary to provide a uniformly structured PID collection. This points to the largest problems of the missing standardization of Magnet Links. On the first hand, the adaption of the Magnet Link scheme in data dissemination systems, such as repository software may be throttled, as the uncertainty will make software developers, founders and managers anxious regarding the future support of Magnet Links and thus the integration into data dissemination infrastructure. On the other hand, new emerging research data repositories already adapt Magnet Links [96] for data dissemination if they are supporting BitTorrent technology, due to the wide adaption of Magnet Links in numerous end-user tools [167]. Hence, it may be necessary to restart the standardization of Magnet Links at the Internet standardization organizations for promoting the use of them in the PID domain and research data dissemination.

3.) Adaption of enhanced NDN access information is needed to provide a trustful data dissemination through NDN. This means that the NDN community has to provide a way to distribute NDN data names together with information for data verification. Currently, the content verification information is provided in the data packet responding a data interest. However, it is not distinguishable for the user if a data packet with a valid signature originates from the party that is authorized to issue the data set without additional access information to a cryptographic certificate proving the identity of the data issuer and the association of the signature. Hence, arbitrary parties can answer interests with incorrect data that contains their valid signature, but does not contain the data the client was looking for. To fix this problem, the data verification needs to be bootstrapped already at the stage of data access information sharing. By this, incorrect answers from unauthorized parties can be neglected using the verification information that has been provided by the enhanced NDN access information. Our approach presented in Section 6.3.1 closes this gap. On the other side, until this effort has not been adapted by the NDN community, a trusted data access to NDN resources using a PID can only be done employing our method of extended Magnet Links for providing trust-augmented access information. But this has the limitation that the standard NDN tools and libraries cannot process the Magnet Link encoded access information that is containing the content verification information. As a result, encoding and sharing NDN access information needs to be extended and promoted within the NDN research community to improve long-term research data dissemination through NDN.

Chapter 8

Conclusion

We give a short summary of this thesis and provide an outlook on potential work that is open for future research.

8.1 Summary

This thesis is divided into two parts that are located in the intersection between persistent identifiers and location-independent network technology. For the PID part, we focused on the Handle-based PID systems and for the location-independent network technology part, we concentrated on state-of-the-art overlay networks using BitTorrent and ICN-based Named Data Networking as current topics of research.

In the first part of the thesis, we proposed novel approaches for shifting the foundation of the Handle PID system from its location-dependent as-is principle to a new location-independent principle. After identifying the benefits of this transition, we provided a deep dive into the internals of this PID system and presented the impact of this shift to the location-independent network domain of NDN. We have realized this without breaking the interoperability to the existing PID infrastructure, knowing that research data dissemination has a slow change momentum. We introduced a model for the conversion of PID and NDN namespaces and transformed the communication and resource models of the Handle PID system into NDN-driven models. For this, we formulated two major communication models that we partially inspired by the original Handle system design and its infrastructure architecture. The first proposed communication model, called *NDN PID push*, extended the state-of-the-art NDN communication principles by a push-based approach that allows data transport within NDN interests. It enables a spontaneous network interaction between NDN nodes with a reduced number of network round trips and an asynchronous one-way communication with parallelized payload transport. With the proposal of our second communication model, called *NDN PID pull*, we provided an approach to PID resolution and administrative data access that is independent of the

LHS network location. Besides this, NDN PID pull takes advantages of NDN network technologies and provides a robust PID resolution in faulty network environments as well as a better scale-out behavior for frequently resolved PIDs based on NDN's distributed network cache design. We implemented both NDN Handle protocol communication modes and carried out an evaluation using a simulated Handle testbed in Mini-NDN. For having realistic input data and PID access distributions in our simulation, we employed real-world PID resolution data from EPIC that we carefully anonymized in order to protect the users. To provide an interaction between the NDN-enabled and the current PID infrastructure, we proposed a gateway architecture. Our architecture features a Handle protocol analyzer based on a state machine to select the suitable communication protocol (NDN PID push or pull) for inbound communication forwarding. For this, the header and flags of the Handle message are analyzed. As the Handle protocol is designed as a stateful protocol, we optimized protocol decisions based on the Handle message state tables. By this, a shortcut in the decision process is possible that leads to a fast message forwarding strategy. For outbound communication, all requests are rewritten into the native Handle protocol messages and the PID requests received from NDN PID pull are matched against incoming Handle response messages using a fixed set of rules.

In the second part of the thesis, we proposed an approach to persistent identifiers that include location-independent access information. First, we pointed out the distributed maintenance efforts for PIDs that are needed to adjust the target URL of the linked data set to the currently valid network location. To solve this problem, we then introduced our approach on creating maintenance-free PIDs which use Magnet Link-encoded access information that describes the content of the linked data sets instead of pointing to an adjustable network address. For this, we transferred the existing non-standardized Magnet URI scheme from the domain of the file sharing community into the domain of PID. As the Magnet URI format was only capable of storing location-independent for BitTorrent as well as other (legacy) systems, we proposed an extension for embedding NDN access information into the Magnet Links. By this, we can use Magnet Links as a container format in NDN that holds the full access information consisting of a NDN data name and cryptographic information that is needed to verify the data sets obtained from the NDN network. Furthermore, we propose to integrating data verification information into the Magnet Link-encoded access information of PID which allows solving the problem that data genuineness cannot be inferred from the data location in BitTorrent and NDN networks but only by using cryptographic signatures attributing data sets to specific data producers. Our contribution enables the NDN community to exchange trustworthy access information in a fixed format – a contribution that has not been made to the NDN community yet. We implemented our approach in order to verify it employing the Handle system. By this, we can show that the existing Handle system is able to store and resolve PIDs with Magnet Links applying new tools able to create and resolve Magnet Link-enabled PIDs. For evaluating our approach, we use structural PID data derived from real-world EPIC PID resolution data extracted with our high speed Handle PID mining tool, called Minera. Again, the resolution data has been carefully anonymized and the obtained target URLs have been deleted after extracting their structure to protect the users' interests.

8.2 Outlook

The results presented in this thesis show that the introduction of location-independent network techniques is advantageous for the concept of PIDs. This will especially important, when the demand for PID-tagging is increasing with the rise of e-science. However, there are several fields for conducting further research.

We investigated the usage for location-independent PIDs in conjunction with NDN. However, more state-of-the-art concepts for location-independent networks exist that could be employed in the domain of PID as well and generate new research questions. As foundations for this, overlay networks based on DHT can serve as a starting point for further investigation, as DHT has been understood very well and numerous DHT-based efforts exist for creating decentralized systems such as social networks or file sharing systems. The results on DHT usage in PID systems may help to harden the Handle PID system and to decentralize its critical infrastructure.

Furthermore, this thesis concentrates on the Handle PID system for verifying its concepts. Although the Handle PID system, which possesses the largest user base of all other PID systems, is a valid choice for PID system research, the approaches of this thesis might be transferable to other PID system, such as ARK or PURL. Thus, future research questions could be directed towards the integration of persistent PID resolution targets in other PID systems. We expect that this transfer is easy to realize for any PID system that allows the integration of resolution targets using extensible data types. Within the RDA, the work is already going in this direction in order to have a common standardized data type system for different PID systems. With the success of this work, the integration of Magnet Links should be possible for any PID system which is adapting the RDA-initiated type system and may generate more research questions in order to improve these PID systems.

The result of the NDN PID pull and push evaluation already indicated that the realization of well-performing NDN network stacks is necessary to provide results that are able to compete with classic network software. We hope that future research efforts will help to improve the performance of the NFD software implementation and also will provide better simulation environments. The future potential of the data push approach formulated in NDN PID push might be very interesting for NDN researchers. While it currently provides a relatively low data throughput in our test bed, it can be used for solving various problems in NDN networks where instant data transmission is needed. This might be tasks such as call signaling in Voice over IP (VOIP) software stacks using NDN or the sending of small data packets in distributed sensor clusters in IoT environments. Furthermore, we expect that this approach can also be used for resource discovery in NDN networks with a very low network foot print.

Hence, by looking at this thesis, we see that location-independent network technology and persistent identifiers form interesting research fields that have not reached their peaks yet.

Bibliography

[1] G. Bell, T. Hey, and A. Szalay, "Beyond the Data Deluge", *Science*, vol. 323, no. 5919, pp. 1297–1298, Mar. 2009.

[2] S. Lawrence, D. Pennock, G. Flake, R. Krovetz, F. Coetzee, E. Glover, F. Nielsen, A. Kruger, and C. Giles, "Persistence of Web references in scientific research", *Computer*, vol. 34, no. 3, pp. 26–31, Mar. 2001.

[3] C. Lynch, "Big data: How do your data grow?", *Nature*, vol. 455, no. 7209, pp. 28–29, Sep. 2008.

[4] P. Lord, A. Macdonald, L. Lyon, and D. Giaretta, "From Data Deluge to Data Curation", in *In Proceedings of the 3th UK e-Science All Hands Meeting*, Nottingham, UK, Aug. 2004, pp. 371–375.

[5] Deutsche Forschungsgemeinschaft, Ed., *Vorschläge zur Sicherung guter wissenschaftlicher Praxis: DENKSCHRIFT ; Empfehlungen der Kommission "Selbstkontrolle in der Wissenschaft"*, Erg. Aufl. Weinheim: Wiley-VCH, 2013.

[6] R. Kahn and R. Wilensky, "A framework for distributed digital object services", *International Journal on Digital Libraries*, vol. 6, no. 2, pp. 115–123, Apr. 2006.

[7] R. E. Duerr, R. R. Downs, C. Tilmes, B. Barkstrom, W. C. Lenhardt, J. Glassy, L. E. Bermudez, and P. Slaughter, "On the utility of identification schemes for digital earth science data: An assessment and recommendations", *Earth Science Informatics*, vol. 4, no. 3, pp. 139–160, Sep. 2011.

[8] L. Cinquini, D. Crichton, C. Mattmann, J. Harney, G. Shipman, F. Wang, R. Ananthakrishnan, N. Miller, S. Denvil, M. Morgan, Z. Pobre, G. M. Bell, C. Doutriaux, R. Drach, D. Williams, P. Kershaw, S. Pascoe, E. Gonzalez, S. Fiore, and R. Schweitzer, "The Earth System Grid Federation: An open infrastructure for access to distributed geospatial data", *Future Generation Computer Systems*, vol. 36, pp. 400–417, Jul. 2014.

[9] V. Cerf, "Avoiding Bit Rot: Long-Term Preservation of Digital Information", *Proceedings of the IEEE*, vol. 99, no. 6, pp. 915–916, Jun. 2011.

[10] A. Blazic, "Long Term Trusted Archive Services", in *Proceedings of the 1st International Conference on Digital Society (ICDS) 2007*, Guadeloupe, French Caribbean, Jan. 2007, pp. 29–29.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content", in *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, Rome, Italy, Dec. 2009, pp. 1–12.

[12] M. Pathan, R. Buyya, and A. Vakali, "Content Delivery Networks: State of the Art, Insights, and Imperatives", in *Content Delivery Networks*, ser. Lecture Notes Electrical Engineering, R. Buyya, M. Pathan, and A. Vakali, Eds., vol. 9, Berlin, Heidelberg: Springer, 2008, pp. 3–32.

[13] A. Vakali and G. Pallis, "Content delivery networks: Status and trends", *IEEE Internet Computing*, vol. 7, no. 6, pp. 68–74, Nov. 2003.

[14] C. Dannewitz, M. Herlich, and H. Karl, "OpenNetInf - prototyping an information-centric Network Architecture", in *Proceedings of the 37th IEEE Conference on Local Computer Networks Workshops 2012*, Clearwater, USA, Oct. 2012, pp. 1061–1069.

[15] W. Allcock, I. Foster, S. Tuecke, A. Chervenak, and C. Kesselman, "Protocols and services for distributed data-intensive science", in *Proceedings of Advanced Computing and Analysis Techniques in Physics Research*, Batavia, USA, Oct. 2000, pp. 161–163.

[16] B. Cohen, *BitTorrent - a new P2p app*, Jul. 2001. [Online]. Available: `https://groups.yahoo.com/neo/groups/decentralization/conversations/topics/3160` (visited on 02/29/2016).

[17] S. Shannigrahi, C. Papadopoulos, E. Yeh, H. Newman, A. J. Barczyk, R. Liu, A. Sim, A. Mughal, I. Monga, J.-R. Vlimant, and J. Wu, "Named Data Networking in Climate Research and HEP Applications", *Journal of Physics: Conference Series*, vol. 664, no. 5, Dec. 2015.

[18] C. Olschanowsky, S. Shannigrahi, and C. Papadopoulos, "Supporting climate research using named data networking", in *Proceedings of the 20th IEEE International Workshop on Local & Metropolitan Area Networks (LANMAN) 2014*, Reno, USA, May 2014, pp. 1–6.

[19] P. Mockapetris, *RFC 1034 - Domain Name concepts and facilities*, 1987. [Online]. Available: `https://tools.ietf.org/html/rfc1034` (visited on 10/10/2015).

[20] E. Tonkin, "Persistent identifiers: Considering the options", *Ariadne, Web Magazine for Information Professionals*, no. 56, 2008. [Online]. Available: `http://www.ariadne.ac.uk/issue56/tonkin` (visited on 10/13/2015).

[21] F. Berman, "Got data? A guide to data preservation in the information age", *Communications of the ACM*, vol. 51, no. 12, pp. 50–56, Dec. 2008.

[22] Boston University Libraries, *Research Data Management*, 2013. [Online]. Available: `http://www.bu.edu/datamanagement/background/whatisdata/` (visited on 03/22/2016).

[23] J. Diederich and J. Milton, "Creating domain specific metadata for scientific data and knowledge bases", *IEEE Transactions on Knowledge and Data Engineering*, vol. 3, no. 4, pp. 421–434, Dec. 1991.

[24] J. Greenberg, H. C. White, S. Carrier, and R. Scherle, "A Metadata Best Practice for a Scientific Data Repository", *Journal of Library Metadata*, vol. 9, no. 3-4, pp. 194–212, Nov. 2009.

[25] A. Shoshani and H. Wong, "Statistical and Scientific Database Issues", *IEEE Transactions on Software Engineering*, vol. SE-11, no. 10, pp. 1040–1047, Oct. 1985.

[26] J. C. French, "What is Metadata?", in *Proceedings of the SDM '92 Workshop: SCIENTIFIC Data Management Workshop*, Salt Lake City, USA, Nov. 1992, pp. 3–8.

[27] M. S. Mayernik, G. S. Choudhury, T. DiLauro, E. Metsger, B. Pralle, M. Rippin, and R. Duerr, "The Data Conservancy Instance: Infrastructure and Organizational Services for Research Data Curation", *D-Lib Magazine*, vol. 18, no. 9/10, Sep. 2012.

[28] R. Heery and S. Anderson, *Digital Repositories Review*, Feb. 2005. [Online]. Available: `http://www.ukoln.ac.uk/repositories/publications/review-200502/digital-repositories-review-2005.pdf` (visited on 10/05/2015).

[29] Re3data.org Team, *Over 1,000 research data repositories indexed in re3data.org — re3data.org*, Nov. 2014. [Online]. Available: `http://www.re3data.org/2014/11/over-1000-research-data-repositories-indexed-in-re3data-org/` (visited on 10/05/2015).

[30] Corporation for National Research Initiatives, *HDL® Identifier and Resolution Services*, Oct. 2015. [Online]. Available: `http://www.handle.net/factsheet.html` (visited on 02/07/2016).

[31] International DOI Foundation, *DOI News - September 2014*, Sep. 2014. [Online]. Available: `http://www.doi.org/news/DOI_News_Sep14.pdf` (visited on 02/07/2016).

[32] N. Paskin, "Digital Object Identifiers for scientific data", *Data Science Journal*, vol. 4, pp. 12–20, 2005.

[33] ——, "Digital Object Identifier (DOI) System", in *Encyclopedia of Library and Information Sciences*, 3rd ed, Boca Raton, FL: CRC Press, Sep. 2011, pp. 1586–1592.

[34] European Persistent Identifier Consortium, *European Persistent Identifier Consortium*, Mar. 2016. [Online]. Available: `http://www.pidconsortium.eu/` (visited on 04/18/2016).

[35] A. Karakannas and Z. Zhao, *Information Centric Networking for Delivering Big Data with Persistent Identifiers*. Amsterdam, Netherlands: University of Amsterdam, 2014. [Online]. Available: `https://www.os3.nl/_media/2013-2014/courses/rp2/p63_report.pdf` (visited on 03/04/2016).

[36] S. X. Sun, S. Reilly, and B. Boesch, *RFC 3650 - Handle System Overview*, 2003. [Online]. Available: `https://tools.ietf.org/html/rfc3650` (visited on 02/07/2016).

[37] S. X. Sun, S. Reilly, and L. Lannom, *RFC 3651 - Handle System Namespace and Service Definition*, 2003. [Online]. Available: `https://tools.ietf.org/html/rfc3651` (visited on 02/07/2016).

[38] S. X. Sun, S. Reilly, L. Lannom, and J. Petrone, *RFC 3652 - Handle System Protocol (ver 2.1) Specification*, 2003. [Online]. Available: `https://tools.ietf.org/html/rfc3652`.

[39] Corporation for National Research Initiatives, *System Fundamentals*, Jun. 2012. [Online]. Available: `http://www.handle.net/overviews/system_fundamentals.html` (visited on 02/07/2016).

[40] International Organization for Standardization (ISO), "Information and documentation – Digital object identifier system", Geneva, Switzerland, ISO 26324:2012, 2012.

[41] Digital Object Numbering Authority, *Multi-Primary Administrators*, Jan. 2016. [Online]. Available: `https://www.dona.net/mpa/` (visited on 04/05/2016).

[42] V. Boehlke, T. Compart, and T. Eckart, "Building up a CLARIN resource center–Step 1: Providing metadata", in *Proceedings of LREC Workshop on Describing LRs with Metadata: TOWARDS Flexibility and Interoperability in the Documentation of LR*, Instanbul, Turkey, May 2012, pp. 21–29.

[43] R. Fielding and J. Reschke, *RFC 7230 - Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing*, Jun. 2014. [Online]. Available: `https://tools.ietf.org/html/rfc7230` (visited on 12/07/2015).

[44] Deutsche Nationalbibliothek, *Systembeispiele*, Jan. 2008. [Online]. Available: `http://www.persistent-identifier.de/ueberblick/Beispiele.php` (visited on 10/14/2015).

[45] G. Mohr, *Magnet URI - Draft Tech Overview/Spec*, Jun. 2002. [Online]. Available: `http://magnet-uri.sourceforge.net/magnet-draft-overview.txt` (visited on 02/07/2016).

[46] J. Chapweske, *HTTP Extensions for a Content-Addressable Web*, Nov. 2001. [Online]. Available: `http://lists.w3.org/Archives/Public/www-talk/2001NovDec/0090.html` (visited on 02/07/2016).

[47] E. Van der Sar, *The Pirate Bay Tracker Shuts Down for Good*, Nov. 2009. [Online]. Available: `https://torrentfreak.com/the-pirate-bay-tracker-shuts-down-for-good-091117/` (visited on 02/07/2016).

[48] B. Cohen, *The BitTorrent Protocol Specification - BEP 3*, Oct. 2013. [Online]. Available: `http://www.bittorrent.org/beps/bep_0003.html` (visited on 02/07/2016).

[49] A. Loewenstern and A. Norberg, *DHT Protocol - BEP 5*, Jan. 2008. [Online]. Available: `http://www.bittorrent.org/beps/bep_0005.html` (visited on 02/07/2016).

[50] P. Maymounkov and D. Mazières, "Kademlia: A Peer-to-Peer Information System Based on the XOR Metric", in *Peer-to-Peer Systems*, vol. 2429, Berlin, Heidelberg: Springer, 2002, pp. 53–65.

[51] O. Wannenwetsch and T. Majchrzak, "On Constructing Persistent Identifiers with Persistent Resolution Targets", in *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS) 2016*, Gdańsk, Poland, Sep. 2016, pp. 1031–1040.

[52] J. Hoffmann, *HTTP Seeding - BEP 17*, Feb. 2008. [Online]. Available: `http://bittorrent.org/beps/bep_0017.html` (visited on 02/29/2016).

[53] M. Burford, *WebSeed - HTTP/FTP Seeding (GetRight style) - BEP 19*, Feb. 2008. [Online]. Available: `http://bittorrent.org/beps/bep_0017.html` (visited on 02/29/2016).

[54] A. Grunthal, *Peer Exchange (PEX) - BEP 11*, Oct. 2015. [Online]. Available: `http://www.bittorrent.org/beps/bep_0011.html` (visited on 02/07/2016).

[55] A. Legout, G. Urvoy-Keller, and P. Michiardi, "Rarest first and choke algorithms are enough", in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, Rio de Janeiro, Brazil, 2006, p. 203.

[56] Decentralized Systems and Network Services Research Group, *Available Bootstrap Peers*, Jun. 2014. [Online]. Available: `http://dsn.tm.kit.edu/english/2936.php#block3173` (visited on 03/01/2016).

[57] G. Hazel and A. Norberg, *Extension for Peers to Send Metadata Files - BEP 9*, Nov. 2014. [Online]. Available: `http://www.bittorrent.org/beps/bep_0009.html` (visited on 03/01/2016).

[58] Named Data Networking Project, *NDN Frequently Asked Questions (FAQ) - Named Data Networking (NDN)*, Jun. 2014. [Online]. Available: `http://named-data.net/project/faq/` (visited on 10/12/2015).

[59] Palo Alto Research Center, Inc., *Content-Centric Networking*, May 2015. [Online]. Available: `https://www.parc.com/work/focus-area/content-centric-networking/` (visited on 10/12/2015).

[60] The National Science Foundation, *NSF Award Search: Award#1345318 - FIA-NP: Collaborative Research: Named Data Networking Next Phase (NDN-NP)*, May 2014. [Online]. Available: `http://www.nsf.gov/awardsearch/showAward? AWD_ID=1345318` (visited on 10/12/2015).

[61] B. Cohen, "Incentives build robustness in BitTorrent", in *Proceedings of the First Workshop on the Economics of Peer-to-Peer Systems*, vol. 6, Berkley, USA, 2003, pp. 68–72.

[62] M. Mosko, *IETF Internet-Drafts: Labeled Content Information – draft-mosko-icnrg-ccnxlabeledcontent-00*, 2015. [Online]. Available: `http://tools.ietf.org/html/draft-mosko-icnrg-ccnxlabeledcontent-00` (visited on 10/10/2015).

[63] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "NDN Technical Memo: Naming Conventions - NDN, Technical Report NDN-0023, Revision 1", Jul. 2014. [Online]. Available: `http://named-data.net/wp-content/uploads/2014/08/ndn-tr-22-ndn-memo-naming-conventions.pdf` (visited on 02/07/2016).

[64] P. Mahadevan, E. Uzun, S. Sevilla, and J. Garcia-Luna-Aceves, "CCN-KRS: A key resolution service for CCN", in *Proceedings of the 1st ACM conference on Information-centric Networking (ICN) 2014*, Paris, France, Sep. 2014, pp. 97–106.

[65] L. Sun, F. Song, D. Yang, and Y. Qin, "DHR-CCN, Distributed hierarchical routing for content centric network", *Journal of Internet Services and Information Security (JISIS)*, vol. 3, no. 1/2, pp. 71–82, 2013.

[66] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang, "Nlsr: Named-data Link State Routing Protocol", in *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ser. ICN '13, New York, USA, 2013, pp. 15–20.

[67] V. Lehman, A. K. M. M. Hoque, Y. Yu, L. Wang, B. Zhang, and L. Zhang, "A Secure Link State Routing Protocol for NDN", Jan. 2016. [Online]. Available: `http://named-data.net/wp-content/uploads/2016/01/ndn-0037-1-nlsr.pdf` (visited on 05/30/2016).

[68] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in Named Data Networking", in *Proceedings of the 21st IEEE International Conference on Network Protocols (ICNP) 2013*, Göttingen, Germany, Oct. 2013, pp. 1–10.

[69] H. Zimmermann, "OSI Reference Model–The ISO Model of Architecture for Open Systems Interconnection", *IEEE Transactions on Communications*, vol. 28, no. 4, pp. 425–432, Apr. 1980.

[70] X. N. Nguyen, D. Saucez, and Turletti, Thierry, "Providing CCN functionalities over OpenFlow switches", Tech. Rep., Aug. 2013. [Online]. Available: `https://hal.inria.fr/hal-00920554` (visited on 03/08/2016).

[71] S. Salsano, N. Blefari-Melazzi, A. Detti, G. Morabito, and L. Veltri, "Information Centric Networking over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed", *Computer Networks*, vol. 57, no. 16, pp. 3207–3221, Nov. 2013.

[72] J. Lee and D. Kim, "Proxy-assisted content sharing using content centric networking (CCN) for resource-limited mobile consumer devices", *IEEE Transactions on Consumer Electronics*, vol. 57, no. 2, pp. 477–483, May 2011.

[73] V. Jacobson, "Congestion avoidance and control", *ACM SIGCOMM Computer Communication Review*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[74] C. A. Wood and E. Uzun, "Flexible end-to-end content security in CCN", in *Proceedings of the 11th Annual IEEE Consumer Communications & Networking Conference (CCNC) 2014*, Las Vegas, USA, Jan. 2014, pp. 858–865.

[75] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone, *Handbook of applied cryptography*, ser. CRC Press series on discrete mathematics and its applications. Boca Raton: CRC Press, 1997.

[76] D. E. Robling Denning, *Cryptography and data security*. Reading, Mass, 1982.

[77] W. Diffie and M. Hellman, "New directions in cryptography", en, *IEEE Transactions on Information Theory*, vol. 22, no. 6, pp. 644–654, Nov. 1976.

[78] B. Schneier, J. Kelsey, D. Whiting, D. Wagner, C. Hall, and N. Ferguson, *The Twofish Encryption Algorithm: A 128-bit Block Cipher*. New York, USA: John Wiley & Sons, Inc., 1999.

[79] J. Daemen and V. Rijmen, *The design of Rijndael: AES – the Advanced Encryption Standard*. Berlin, Heidelberg: Springer, 2002.

[80] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978. (visited on 05/09/2016).

[81] V. S. Miller, "Use of Elliptic Curves in Cryptography", in *Advances in Cryptology – CRYPTO '85 Proceedings*, H. C. Williams, Ed., vol. 218, Berlin, Heidelberg: Springer, 1986, pp. 417–426.

[82] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, vol. 48, no. 177, p. 203, Jan. 1987.

[83] L. Nuaymi, *WiMAX: Technology for broadband wireless access*. Chichester, England: John Wiley, 2007.

[84] S. X. Sun, "Internationalization of the Handle System - A Persistent Global Name Service", in *Proceeding of 12th International Unicode Conference*, Tokyo, Japan, 1998.

[85] M. J. Bates and M. N. Maack, Eds., *Encyclopedia of library and information sciences*, 3rd ed. Boca Raton, FL: CRC Press, 2010.

[86] K. Shafer, S. Weibel, E. Jul, and J. Fausey, "Introduction to persistent uniform resource locators", in *Proceeding of the 6th Annual Conference of the Internet Society*, ser. INET '96, Montreal, Canada, 1996.

[87] IEEE, *IEEE Xplore*, Oct. 2015. [Online]. Available: `http://ieeexplore.ieee.org` (visited on 10/05/2015).

[88] S. Rozenfeld, *RFC 3044 - Using International Serials Number as URN*, Jan. 2001. [Online]. Available: `https://tools.ietf.org/html/rfc3044` (visited on 10/14/2015).

[89] U. Ackermann, C. Berner, N. Elbert, J. Klett, K. K. Koçer, N. von der Hude, and M. Wiegand, *Policy für die Vergabe von URNs im Namensraum urn:nbn:de Version 1.0*. Leipzig, Frankfurt am Main: Deutsche Nationalbibliothek, Nov. 2012. [Online]. Available: `http://d-nb.info/1029114455/34` (visited on 10/14/2015).

[90] J. Mirkovic, S. Diederich, D. Dittrich, and P. Reiher, *Internet denial of service: Attack and defense mechanisms*, ser. The Radia Perlman series in computer networking and security. Upper Saddle River, USA: Prentice Hall Professional Technical Reference, 2005.

[91] H.-W. Hilse and J. Kothe, *Implementing persistent identifiers: Overview of concepts, guidelines and recommendations*. London: Consortium of European Research Libraries, 2006.

[92] I. Foster, C. Kesselman, and S. Tuecke, "The Anatomy of the Grid: Enabling Scalable Virtual Organizations", *International Journal of High Performance Computing Applications*, vol. 15, no. 3, pp. 200–222, Aug. 2001.

[93] G. Khanna, U. Catalyurek, T. Kurc, R. Kettimuthu, P. Sadayappan, I. Foster, and J. Saltz, "Using Overlays for Efficient Data Transfer over Shared Wide-area Networks", in *Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*, ser. SC '08, Austin, Texas, Nov. 2008, 47:1–47:12.

[94] L. Ramakrishnan, C. Guok, K. Jackson, E. Kissel, D. M. Swany, and D. Agarwal, "On-demand Overlay Networks for Large Scientific Data Transfers", in *Proceeding of the 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing (CCGrid) 2010*, Melbourne, Australia, May 2010, pp. 359–367.

[95] D. Steer, D. Boyd, B. Cregan, S. Gray, S. Price, V. Knight, and C. Gardiner, "Data.bris Research Data Repository Framework", in *Proccedings of Digital Research 2012*, Oxford, United Kingdom, Sep. 2012.

[96] J. P. Cohen and H. Z. Lo, "Academic Torrents: A Community-Maintained Distributed Repository", in *Proceedings of the 2014 Annual Conference on Extreme Science and Engineering Discovery Environment (XSEDE)*, Atlanta, USA, Jul. 2014, 2:1–2:2.

[97] ——, *Academic Torrents*, Oct. 2016. [Online]. Available: `http://academictorrents.com/` (visited on 10/08/2016).

[98] C. Fan, S. Shannigrahi, S. DiBenedetto, C. Olschanowsky, C. Papadopoulos, and H. Newman, "Managing Scientific Data with Named Data Networking", in *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, Austin, USA, Nov. 2015, 1:1–1:7.

[99] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, "XROOTD - A Highly scalable architecture for data access", *WSEAS Transactions on Computers*, vol. 1, no. 4.3, 2005.

[100] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, L. Yanbiao, S. Mastorakis, Y. Huang, J. P. Abraham, S. DiBenedetto, F. Chengyu, C. Papadopoulos, D. Pesavento, G. Grassi, G. Pau, H. Zhang, T. Song, H. Yuan, H. B. Abraham, P. Crowley, S. O. Amin, V. Lehman, and L. Wang, "NDN Technical Report: NFD Developer's Guide - NDN, Technical Report NDN-0021,Revision 5", Oct. 2015. [Online]. Available: `http://named-data.net/wp-content/uploads/2015/10/ndn-0021-5-nfd-developer-guide.pdf` (visited on 03/07/2016).

[101] K. Sollins, "Pervasive persistent identification for Information centric networking", in *Proceedings of the Second Edition of the ICN Workshop on Information-centric Networking*, Helsinki, Finland, Aug. 2012, pp. 1–6.

[102] A. Ghodsi, T. Koponen, J. Rajahalme, P. Sarolahti, and S. Shenker, "Naming in content-oriented architectures", in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking ICN '11*, Toronto, Canada, 2011, pp. 1–6.

[103] 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, Jul. 2012.

[104] O. Schmitt, T. Majchrzak, and S. Bingert, "Experimental realization of a Persistent Identifier Infrastructure stack for Named Data Networking", in *Proceedings of the 10th International IEEE Conference on Networking, Architecture, and Storage (NAS) 2015*, Boston, USA, Aug. 2015, pp. 33–38.

[105] L. Daigle, D.-W. van Gulik, R. Iannella, and P. Fältström, *RFC 3406 - URN Namespace Definition Mechanisms*, Oct. 2002. [Online]. Available: `https://tools.ietf.org/html/rfc3406` (visited on 10/14/2015).

[106] M. Mealling and R. Daniel, *RFC 2483 - URI Resolution Services Necessary for URN Resolution*, Jan. 1999. [Online]. Available: `https://tools.ietf.org/html/rfc2483` (visited on 10/14/2015).

[107] A. Karakannas and Z. Zhao, *Information Centric Networking for Delivering Big Data with Persistent Identifiers (Presentation Slides)*. Amsterdam, Netherlands: University of Amsterdam, 2014. [Online]. Available: `https://www.os3.nl/_media/2013-2014/courses/rp2/p63_presentation.pdf` (visited on 03/11/2016).

[108] C. Dannewitz, J. Golic, B. Ohlman, and B. Ahlgren, "Secure Naming for a Network of Information", in *Proceedings of IEEE Conference on Computer Communications INFOCOM*, San Diego, USA, Mar. 2010, pp. 1–6.

[109] S. Haun and A. Nürnberger, "Towards Persistent Identification of Resources in Personal Information Management", in *Proceedings of the 3rd International Workshop on Semantic Digital Archives (SDA 2013)*, vol. 1091, Valetta, Malta, Sep. 2013, pp. 73–80.

[110] H. Dai, B. Liu, Y. Chen, and Y. Wang, "On pending interest table in named data networking", in *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, Austin, USA, 2012, pp. 211–222.

[111] S. Braun, M. Monti, M. Sifalakis, and C. Tschudin, "CCN & TCP co-existence in the future Internet: Should CCN be compatible to TCP?", Ghent, Belgium, May 2013, pp. 1109–1115.

[112] C. Xia, M. Xu, and Y. Wang, "A loss-based TCP design in ICN", in *Proceedings of the 22nd Wireless and Optical Communication Conference (WOCC) 2013*, Chongqing, China, May 2013, pp. 449–454.

[113] M. Allmann, V. Paxson, and E. Blanton, *RFC 5681 - TCP Congestion Control*, Sep. 2009. [Online]. Available: `https://tools.ietf.org/html/rfc5681` (visited on 04/13/2016).

[114] Named Data Networking Project, *NDN Packet Format Specification*, Jul. 2015. [Online]. Available: `http://named-data.net/doc/ndn-tlv/` (visited on 03/13/2016).

[115] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang, "Act: Audio conference tool over named data networking", in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking ICN '11*, Toronto, Canada, 2011, p. 68.

[116] W. Shang, J. Thompson, M. Cherkaoui, J. Burkey, and L. Zhang, "NDN.JS: A javascript client library for named data networking", in *Proceedings of 2013 IEEE Conference on Computer Communications Workshops INFOCOM WKSHPS*, Turin, Italy, Apr. 2013, pp. 399–404. (visited on 07/25/2016).

[117] H. Yuan and P. Crowley, "Experimental evaluation of content distribution with NDN and HTTP", in *Proceedings of the 32nd IEEE International Conference on Computer Communications (INFOCOM) 2013*, Turin, Italy, Mar. 2013, pp. 240–244.

[118] J. Kneschke, *Lighttpd*, 2016. [Online]. Available: `https://www.lighttpd.net/` (visited on 07/27/2016).

[119] D. Wessels, H. Nordström, A. Jeffries, A. Russkov, F. Chemolli, R. Collins, and G. Serassio, *Squid : Optimising Web Delivery*, 2016. [Online]. Available: `http://www.squid-cache.org/` (visited on 07/27/2016).

[120] Y. Wang and X. Qiao, "Design and implementation of web browser for named data networking in Windows", in *Proceedings of the 2nd International Conference on Systems and Informatics (ICSAI) 2014*, Shanghai, China, Nov. 2014, pp. 607–612.

[121] G. Nan, X. Qiao, Y. Tu, W. Tan, L. Guo, and J. Chen, "Design and Implementation: The Native Web Browser and Server for Content-Centric Networking", *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 5, pp. 609–610, Aug. 2015.

[122] X. Qiao, G. Nan, W. Tan, L. Guo, J. Chen, W. Quan, and Y. Tu, "Ccnxtomcat: An extended web server for Content-Centric Networking", *Computer Networks*, vol. 75, pp. 276–296, Dec. 2014.

[123] Y. Tu, X. Qiao, G. Nan, J. Chen, and S. Li, "A Priority-Based Dynamic Web Requests Scheduling for Web Servers over Content-Centric Networking", in *Proceedings of the Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb) 2015*, Washington DC, USA, Nov. 2015, pp. 43–48.

[124] N. L. van Adrichem and F. A. Kuipers, "Globally accessible names in named data networking", in *Proceedings of the 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, Turin, Italy, Apr. 2013, pp. 345–350.

[125] European Persistent Identifier Consortium, *Documentation - PID generation*, Dec. 2014. [Online]. Available: `http://doc.pidconsortium.eu/guides/api-generation/` (visited on 04/20/2016).

[126] J. Postel, *RFC 791 - Internet Protocol*, 1981. [Online]. Available: `https://tools.ietf.org/html/rfc791` (visited on 09/11/2016).

[127] S. Deering and R. Hinden, *RFC 2640 - Internet Protocol, Version 6 (IPv6) Specification*, 1998. [Online]. Available: `https://tools.ietf.org/html/rfc2640` (visited on 09/11/2016).

[128] J. Chen, M. Arumaithurai, L. Jiao, X. Fu, and K. Ramakrishnan, "Copss: An Efficient Content Oriented Publish/Subscribe System", in *Proceedings of the 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 11*, New York, USA, Oct. 2011, pp. 99–110.

[129] Z. Liu, L. Liu, R. Hill, and Y. Zhan, "Base62x: An alternative approach to Base64 for non-alphanumeric characters", in *Proceedings of the 8th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD) 2011*, Shanghai, China, Jul. 2011, pp. 2667–2670.

[130] C. Ghali, A. Narayanan, D. Oran, G. Tsudik, and C. A. Wood, "Secure Fragmentation for Content-Centric Networks", in *Proceedings of the 14th IEEE International Symposium on Network Computing and Applications*, Cambridge, USA, Sep. 2015, pp. 47–56.

[131] A. Afanasyev, J. Shi, L. Wang, B. Zhang, and L. Zhang, "NDN Technical Memo: Naming Conventions - NDN, Technical Report NDN-0023, Revision 1", May 2015. [Online]. Available: `https://named-data.net/wp-content/uploads/2015/05/ndn-0032-1-ndn-memo-fragmentation.pdf` (visited on 02/07/2016).

[132] F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus, "Lightweight, Payload-Based Traffic Classification: An Experimental Evaluation", in *Proceedings of the 2008 IEEE Conference on Communication ICC '08*, May 2008, pp. 5869–5875.

[133] Object Management Group, *OMG Unified Modeling Language (OMG UML)*. Mar. 2015. [Online]. Available: `http://www.omg.org/spec/UML/2.5/PDF` (visited on 07/11/2016).

[134] Corporation for National Research Initiatives, "Handle.Net Version 8.1.1 Software Release Notes", in, Apr. 2016. [Online]. Available: `http://www.handle.net/HN_v8.1.1_ReleaseNotes.pdf` (visited on 05/30/2015).

[135] J. Thompson and A. Brown, *Jndn: A Named Data Networking client library for Java*, May 2016. [Online]. Available: `https://github.com/named-data/jndn/releases/tag/v0.12` (visited on 05/30/2016).

[136] A. Afanasyev, J. Shi, D. Pesavento, V. Lehman, F. Chengyu, S. O. Amin, E. Newberry, W. Shang, Y. Tu, H. Yuan, and Y. Yingdi, *NFD - Named Data Networking Forwarding Daemon*, Mar. 2016. [Online]. Available: `https://github.com/named-data/NFD/releases/tag/NFD-0.4.1` (visited on 05/30/2016).

[137] A. K. M. M. Hoque, V. Lehman, S. O. Amin, A. Afanasyev, Y. Yingdi, A. Alyyan, and J. Shi, *NLSR - Named Data Link State Routing Protocol*, Jan. 2016. [Online]. Available: `https://github.com/named-data/NLSR/releases/tag/NLSR-0.2.2` (visited on 05/30/2016).

[138] Corporation for National Research Initiatives, *HDL.NET® Software Client Libraries*, Apr. 2016. [Online]. Available: `https://www.handle.net/client_download.html` (visited on 06/20/2016).

[139] E. R. Harold, *Java network programming*, 4th edition. Beijing: O'Reilly, 2014.

[140] Oracle Corporation, *DatagramSocket (Java Platform SE 7)*, Jan. 2016. [Online]. Available: `http://docs.oracle.com/javase/7/docs/api/java/net/DatagramSocket.html` (visited on 06/27/2016).

[141] M. Grand, *Patterns in Java: A Catalog of Reusable Design Patterns Illustrated with UML*, 2nd. New York, USA: John Wiley & Sons, Inc., 2002.

[142] S. Mastorakis, A. Afanasyev, I. Moiseenko, and L. Zhang, "ndnSIM 2.0: A new version of the NDN simulator for NS-3 - NDN, Technical Report NDN-0028,Revision 1", Jan. 2015. [Online]. Available: `http://www.named-data.net/techreport/ndn-0028-1-ndnsim-v2.pdf` (visited on 03/07/2016).

[143] G. F. Riley and T. R. Henderson, "The ns-3 Network Simulator", in *Modeling and Tools for Network Simulation*, K. Wehrle, M. Güneş, and J. Gross, Eds., Berlin, Heidelberg: Springer, 2010, pp. 15–34.

[144] A. Afanasyev, Y. Yingdi, J. Shi, D. Pesavento, E. Newberry, J. Pereira, S. Mastorakis, W. Shang, M. Sweatt, V. Lehman, J. Thompson, T. Jiewam, J. Quevedo, I. Moiseenko, S. Chen, X. Jiang, and M. Juszkiewicz, *Ndn-cxx: NDN C++ library with eXperimental eXtensions*, Jul. 2016. [Online]. Available: `https://github.com/named-data/ndn-cxx` (visited on 07/04/2016).

[145] D. Kulinski, A. Afanasyev, W. Shang, and Y. Yingdi, *PyNDN - NDN bindings for Python*, May 2016. [Online]. Available: `https://github.com/cawka/PyNDN` (visited on 07/04/2016).

[146] A. Gawande, V. Lehman, L. Wang, J. Shi, B. Zhang, and A. Afanasyev, *Mini-NDN*, Jun. 2016. [Online]. Available: `https://github.com/named-data/mini-ndn` (visited on 07/04/2016).

[147] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks", in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks (Hotnets-IX)*, Monterey, USA, 2010, pp. 1–6.

[148] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown, "Reproducible network experiments using container-based emulation", in *Proceedings of the 8th international conference on Emerging networking experiments and technologies (CoNEXT '12)*, Nice, France, 2012, pp. 253–264.

[149] W. Felter, A. Ferreira, R. Rajamony, and J. Rubio, "An updated performance comparison of virtual machines and Linux containers", in *Proceedings of the 2015 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Philadelphia, USA, Mar. 2015, pp. 171–172.

[150] D. Comer, *Internetworking with TCP/IP*, 4th. Upper Saddle River, USA: John Wiley & Sons, Inc., 2000.

[151] C. Yi, J. Abraham, A. Afanasyev, L. Wang, B. Zhang, and L. Zhang, "On the role of routing in named data networking", in *Proceedings of the 1st ACM conference on Information-centric Networking (ICN) 2014*, Paris, France, Sep. 2015, pp. 27–36.

[152] T. Cruse, *General Assembly 2016, moving DataCite forward*, website, Mar. 2016. [Online]. Available: `https://blog.datacite.org/general-assembly-2016/` (visited on 04/11/2016).

[153] DataCite, *Metadata Stats*, Apr. 2016. [Online]. Available: `http://stats.datacite.org/` (visited on 04/11/2016).

[154] M. Fenner, *Digging into Metadata using R*, website, Aug. 2015. [Online]. Available: `https://blog.datacite.org/digging-into-data-using-r/` (visited on 04/11/2016).

[155] OpenSSL Software Foundation, *OpenSSL - Cryptography and SSL/TLS Toolkit*, 2016. [Online]. Available: `https://www.openssl.org/docs/manmaster/apps/openssl.html` (visited on 08/31/2016).

[156] OCLC Online Computer Library Center, Inc., *PURL Help*, 2015. [Online]. Available: `https://purl.org/docs/help.html` (visited on 10/19/2015).

[157] Corporation for National Research Initiatives, "4.9 Handle Value Line Format", in *HANDLE.NET (version 8.1) Technical Manual*, Nov. 2015, pp. 28–29. [Online]. Available: `https://hdl.handle.net/20.1000/105`.

[158] T. Weigel, S. Kindermann, and M. Lautenschlager, "Actionable Persistent Identifier Collections", *Data Science Journal*, vol. 12, pp. 191–206, 2014.

[159] D. Broeder and L. Lannom, "Data Type Registries: A Research Data Alliance Working Group", *D-Lib Magazine*, vol. 20, no. 1/2, Jan. 2014.

[160] Corporation for National Research Initiatives, *HDL.NET® Proxy Server System*, 2015. [Online]. Available: `https://www.handle.net/proxy_servlet.html` (visited on 02/07/2016).

[161] M. Dürst, L. Masinter, and J. Zawinski, *RFC 6068 - The mailto URI Scheme*, Oct. 2010. [Online]. Available: `https://tools.ietf.org/html/rfc6068` (visited on 08/16/2016).

[162] H. Schulzrinne, *RFC 3966 - The tel URI for Telephone Numbers*, Dec. 2004. [Online]. Available: `https://tools.ietf.org/html/rfc3966` (visited on 08/16/2016).

[163] R. Fielding and J. Reschke, *RFC 7231 - Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content*, Jun. 2014. [Online]. Available: `http://tools.ietf.org/html/rfc7231#section-6.4.4` (visited on 02/07/2016).

[164] A. Norberg, *Libtorrent python binding*, 2015. [Online]. Available: `http://www.rasterbar.com/products/libtorrent/python_binding.html` (visited on 08/30/2016).

[165] M. Hellkamp, *Bottle: Python Web Framework*, Feb. 2016. [Online]. Available: `http://bottlepy.org/docs/0.12/` (visited on 08/30/2016).

[166] European Persistent Identifier Consortium, *Pidconsortium/EPIC-API-v2*, Mar. 2016. [Online]. Available: `https://github.com/pidconsortium/EPIC-API-v2` (visited on 04/13/2016).

[167] Transmission Project, *Transmission*, Mar. 2016. [Online]. Available: `https://www.transmissionbt.com/` (visited on 07/20/2016).

[168] A. Afanasyev, S. Chen, W. Shang, and J. Shi, *Repo-ng: Next generation of NDN repository*, Nov. 2015. [Online]. Available: `https://github.com/named-data/repo-ng` (visited on 04/14/2016).

[169] V. Almeida, A. Bestavros, M. Crovella, and A. de Oliveira, "Characterizing reference locality in the WWW", in *Proceedings of the 4th International Conference on Parallel and Distributed Information Systems*, Miami Beach, USA, Dec. 1996, pp. 92–103.

[170] E. Van der Sar, *Download a Copy of The Pirate Bay, It's Only 90 MB*, Feb. 2012. [Online]. Available: `https://torrentfreak.com/download-a-copy-of-the-pirate-bay-its-only-90-mb-120209/` (visited on 02/07/2016).

[171] T. Weigel and T. DiLauro, *PID Information Types RDA Working Group*, Aug. 2013. [Online]. Available: `https://www.rd-alliance.org/groups/pid-information-types-wg.html` (visited on 07/02/2016).

[172] T. Weigel, T. DiLauro, and T. Zastrow, *RDA PID Information Types WG: Final Report*, Aug. 2015. [Online]. Available: `https://b2share.eudat.eu/record/245/files/PID%20Information%20Types%20Final%20Report.pdf` (visited on 07/10/2016).

[173] Corporation for National Research Initiatives, "Tools", in *HANDLE.NET (version 8.1.1) Software Release Notes*, Feb. 2016, p. 4. [Online]. Available: `https://www.handle.net/HN_v8.1.1_ReleaseNotes.pdf` (visited on 05/15/2016).

[174] S. Nakov, *Internet programming with Java*. Faber, 2004.

[175] ——, *TCPForwardServer.java - Internet programming with Java*, 2004. [Online]. Available: `http://www.nakov.com/books/inetjava/source-code-html/Chapter-1-Sockets/1.4-TCP-Sockets/TCPForwardServer.java.html` (visited on 07/20/2016).

List of Acronyms

ACK   Acknowledgement

AES   Advanced Encryption Standard

API   Application Interface

ARK   Archival Resource Key

ASCII   American Standard Code for Information Interchange

CCN   Content Centric Network

CDN   Content Distribution Network

CHC   Chinese Handle Coalition

CIDR   Classless Inter-Domain Routing

CNRI   Corporation for National Research Initiatives

COPSS   Content Oriented Publish/Subscribe System

CPU   Central Processing Unit

CS   Content Store

DFG   Deutsche Forschungsgemeinschaft

DHT   Distributed Hash Table

DI   Directory Indicator

DNS   Domain Name System

DOI   Document Object Identifier

DONA   Data Oriented Network Architecture

ECC   Elliptic Curve Cryptography

EPIC	European Persistent Identifier Consortium
EQ	Evaluation Question
ESGF	Earth System Grid Federation
ESNet	Energy Science Network
FIB	Forward Information Base
FQDN	Fully-Qualified Domain Name
FTP	File Transfer Protocol
FUSE	Filesystem in Userspace
GHR	Global Handle Registry
GWDG	Gesellschaft für wissenschaftliche Datenverarbeitung
HEP	High Energy Particle Physics
HMAC	Hash-based Message Authentication Code
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ICANN	Internet Corporation for Assigned Names and Numbers
ICN	Information Centric Network
IDF	International DOI Foundation
IETF	Internet Engineering Task Force
IoT	Internet of Things
IP	Internet Protocol
IPC	Inter-Process Communication
ISBN	International Standard Book Number
ISSN	International Standard Serial Number
ITU	International Telecommunication Union
JSON	JavaScript Object Notation

LHC	Large Hadron Collider
LHS	Local Handle System
LRU	Last Recently Used
LSA	Link State Advertisement
MAC	Message Authentication Code
MPA	Multi Primary Administrator
NA	Naming Authority
NASA	National Aeronautics and Space Administration
NDN	Named Data Networking
NETINF	Network of Information
NFD	Network Forwarding Daemon
NLSR	Named Data Link State Routing Protocol
OSI	Open Systems Interconnection Model
P2P	Peer-to-Peer
PARC	Xerox Palo Alto Research Center
PDF	Portable Document Format
PEX	Peer Exchange
PID	Persistent Identifier
PIM	Personal Information Manager
PIT	Pending Interest Table
PKI	Public Key Infrastructure
PPINS	Pervasive Persistent Identification System
PPOID	Pervasive Persistent Object Id
PSM	Protocol State Machine
PURL	Persistent URL

PURSUIT	Publish-Subscribe Architecture
QoS	Quality of Service
RDA	Research Data Alliance
REST	Representational State Transfer
RFC	Request for Comments
RPC	Remote Procedure Call
RSA	Rivest, Shamir and Adleman
SDN	Software Defined Networking
SSD	Solid State Disk
SWORD2	Simple Web-service Offering Repository Deposit 2.0
TCP	Transmission Control Protocol
TLD	Top Level Domain
TTL	Time To Live
UDP	User Datagram Protocol
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
URN	Uniform Resource Name
VCS	Voice Conference System
VIP	Virtual Internet Protocol Address
VM	Virtual Machine
VOIP	Voice over IP
VPN	Virtual Private Network
W3C	World Wide Web Consortium
WAN	Wide Area Network

WORM   Wrote Once Read Multiple

XHR   XMLHttpRequest

List of Symbols

a	message authentication code
C	cipher text – output of an encryption operation
K_D	decryption key – the private key used for decryption in asymmetric encryption
K_E	encryption key – the public key used for encryption in asymmetric encryption
K	key – the secret key used in encryption operations
m	plaintext message
P	plain text – the source message of an encryption operation
i	Handle prefix with the a rank
x_i	number of hops between the measurement server and the official Handle HTTP-proxy
s	digital signature
w_i	weight in percent of all collected measurements
$\bar{x}_w$	weighted average

List of Definitions

List of Figures

List of Listings

List of Tables

Appendix

A.1 Handle Source Code Remarks

In this section, we highlight properties and features of the Handle Server and the Handle library (HDLLib) source code base from Handle Server 8.1.1 [173].

A.1.1 Removal of URN Data Type Support

The support for the URN data type `0.TYPE/URN` has been commented out of the Handle Server 8.1.1 code base [134] for unknown reasons. Thus, URN-support not part of the default Handle Server and client 8.1.1 functionality, as the changes are part of the common HDLLib that is share between the client and server code base. As a consequence, storing URNs in Handle values as a native data type in only possible, if a URN data type has been registered at each LHS individually.

Listing A.1: Removal of the Native URN Data Type Support in HDLLib

```
1  // File: /net/handle/hdllib/Common.java

3  // -------- Line Start: 120 - End: 128 ---------
   public static final byte STD_TYPE_URL[] = Util.encodeString("URL");
5  public static final byte STD_TYPE_EMAIL[] = Util.encodeString("EMAIL"
      );
   public static final byte STD_TYPE_HSALIAS[] = Util.encodeString("
      HS_ALIAS");
7  public static final byte STD_TYPE_HSSITE[] = Util.encodeString("
      HS_SITE");
   public static final byte STD_TYPE_HSSITE6[] = Util.encodeString("
      HS_SITE.6");
9  public static final byte STD_TYPE_HSADMIN[] = Util.encodeString("
      HS_ADMIN");
   public static final byte STD_TYPE_HSSERV[] = Util.encodeString("
      HS_SERV");
11 //  public static final byte STD_TYPE_HOSTNAME[] = Util.encodeString
      ("INET_HOST");
   //  public static final byte STD_TYPE_URN[] = Util.encodeString("URN
      ");
```

```
13
   // -------- Line Start: 134 - End: 146 ---------
15 public static final byte STD_TYPES[][] = { STD_TYPE_URL,
                                              STD_TYPE_EMAIL,
17                                            STD_TYPE_HSADMIN,
                                              STD_TYPE_HSALIAS,
19                                            STD_TYPE_HSSITE,
                                              STD_TYPE_HSSITE6,
21                                            STD_TYPE_HSSERV,
                                              STD_TYPE_HSSECKEY,
23                                            STD_TYPE_HSPUBKEY,
                                              STD_TYPE_HSVALLIST,
25 //                                            STD_TYPE_HOSTNAME,
   //                                            STD_TYPE_URN,
27 };
```

A.2 Handle Source Code Patches and Additions

In this section, all modifications on the Handle Server and the Handle library (HDLLib) source code base are documented in the ANSI patch format. Additions to the code base that extend the native functionality are added as full source code files.

A.2.1 Patches for NDN-enabled Native Handle Communication Using NDN PID Push

Following changes and have been applied to the Handle Server 8.1.1 code base [134], in order to realize native Handle protocol communication over a NDN network using NDN PID push. These changes affect eleven files of the code base.

Listing A.2: Patch 1 for Handle Java API integration of NDN-enabled native communication using NDN PID push

```
1  --- /hsj-8.1.1/src/net/handle/api/GenericHSAdapter.java
   +++ /ndnhandle/src/net/handle/api/GenericHSAdapter.java
3  @@ -347,6 +347,23 @@
      public void setUseUDP(boolean useUDP) {
5        if(useUDP) {
           resolver.getResolver().setPreferredProtocols(new int[] {
7  +         Interface.SP_HDL_UDP,
   +         Interface.SP_HDL_TCP,
9  +         Interface.SP_HDL_HTTP });
   +      } else {
11 +        resolver.getResolver().setPreferredProtocols(new int[] {
   +         Interface.SP_HDL_TCP,
13 +         Interface.SP_HDL_HTTP });
   +      }
15 +    }
   +
17 +  public void setUseNDN(boolean useNDN) {
   +      if(useNDN) {
19 +        resolver.getResolver().setPreferredProtocols(new int[] {
   +         Interface.SP_HDL_NDN,
21          Interface.SP_HDL_UDP,
            Interface.SP_HDL_TCP,
23          Interface.SP_HDL_HTTP });
```

Listing A.3: Patch 2 for Handle Java API integration of NDN-enabled native communication
using NDN PID push

```
1   --- /hsj -8.1.1/ src/net/handle/api/HSAdapter.java
    +++ /ndnhandle/src/net/handle/api/HSAdapter.java
3   @@ -164,6 +164,12 @@
      public void setUseUDP(boolean useUDP);
5
      /**
7   +  * Adds and prioritizes NDN for communication with the Handle
       server
    +  * @param useNDN
9   +  */
    + public void setUseNDN(boolean useNDN);
11  +
    + /**
13     * Updates the specified data handle values. </br> <b> Note: </b>
       * <li>Make sure that the index value is specified in the array of
15     * handle values or else this method will not work well.</li>
```

Listing A.4: Patch 3 for Handle HDLLib integration of NDN-enabled native communication
using NDN PID push (Configuration Subsystem)

```
1   --- /hsj -8.1.1/ src/net/handle/hdllib/GsonUtility.java
    +++ /ndnhandle/src/net/handle/hdllib/GsonUtility.java
3   @@ -597,6 +597,7 @@
      String protocol = obj.get("protocol").getAsString();
5     if("UDP".equals(protocol)) intf.protocol = Interface.SP_HDL_UDP;
      else if("TCP".equals(protocol)) intf.protocol = Interface.
        SP_HDL_TCP;
7   + else if("NDN".equals(protocol)) intf.protocol = Interface.
       SP_HDL_NDN;
      else if("HTTP".equals(protocol)) intf.protocol =
9         Interface.SP_HDL_HTTP;
      else if("HTTPS".equals(protocol)) intf.protocol =
11        Interface.SP_HDL_HTTPS;
      else intf.protocol = obj.get("protocol").getAsByte();
```

Listing A.5: Patch 4 for Handle HDLLib integration of NDN-enabled native communication using NDN PID push (Native Handle Communication Subsystem)

```
--- /hsj-8.1.1/src/net/handle/hdllib/HandleResolver.java
+++ /ndnhandle/src/net/handle/hdllib/HandleResolver.java
@@ -11,6 +11,7 @@

 import java.net.*;
 import java.nio.channels.SocketChannel;
+import java.nio.charset.StandardCharsets;
 import java.io.*;
 import java.util.*;
 import java.util.concurrent.CancellationException;
@@ -27,9 +28,11 @@
 import javax.net.ssl.SSLContext;
 import javax.net.ssl.SSLSocket;

+import de.gwdg.ndn.hdldatagram.transport.HdlDatagramSocket;
 import net.cnri.util.StringUtils;
 import net.handle.security.*;
 import net.handle.util.LRUCacheTable;
+import net.named_data.jndn.Name;

@@ -52,7 +55,7 @@

    boolean useIPv6FastFallback = true;

-   int preferredProtocols[] = { Interface.SP_HDL_UDP,
+   int preferredProtocols[] = { Interface.SP_HDL_NDN,
                                   Interface.SP_HDL_TCP,
                                   Interface.SP_HDL_HTTP,
                                   Interface.SP_HDL_HTTPS };
@@ -2286,11 +2289,13 @@
      case Interface.SP_HDL_TCP:
          response = sendHdlTcpRequest(req, addr, port, callback);
          break;
+     case Interface.SP_HDL_NDN:
+      response = sendHdlNdnRequest(req, addr, port, callback);
      case Interface.SP_HDL_HTTP:
          if (req.hasEqualOrGreaterVersion(2, 8) &&
              expectStreamingResponse(req) && !isDsaPublicKey(req.
              serverPubKeyBytes)) {
             response = sendHttpsRequest(req, addr, port, callback);
          } else {
-            response = sendHttpRequest(req, addr, port, callback);
+            //response = sendHttpRequest(req, addr, port, callback);
          }
          break;
      case Interface.SP_HDL_HTTPS:
@@ -2312,144 +2317,9 @@
      return response;
    }
```

```
48
-   private boolean expectStreamingResponse(AbstractRequest req) {
50 -       return req.opCode == AbstractMessage.OC_RETRIEVE_TXN_LOG ||
    req.opCode == AbstractMessage.OC_DUMP_HANDLES
-               || (req instanceof ChallengeAnswerRequest &&
    expectStreamingResponse(((ChallengeAnswerRequest)req).
    originalRequest));
52 - }
-
54 - /**
-    * Verify response message with the pre-established session key.
56 -    */
-   private final boolean verifyResponseWithSessionKey(AbstractRequest
     req, AbstractResponse response)
58 -     throws HandleException
-   {
60 -     boolean veriPass = false;
-     if (req == null || response == null) return false;
62 -
-     try {
64 -         veriPass = response.verifyMessage(req.sessionInfo.
    getSessionKey());
-     } catch (HandleException e) {
66 -         throw e;
-     } catch (Exception e){
68 -         throw new HandleException(
-                 HandleException.MISSING_OR_INVALID_SIGNATURE,
70 -                 "Error verifying MAC code",e);
-     }
72 -     if(veriPass) {
-         req.sessionInfo.addSessionCounter(response.sessionCounter,
    true);
74 -     }
-     return veriPass;
76 - }
-
78 -
-   /** This function verifies the integrity of a response given the
    request
80 -       that it is for.  The public key of the server is attached to
    the
-       request so that this can verify the signature of the response.
     This
82 -       function also checks the digest of the request that was
    included
-       (if requested) in the response. */
84 - private static final void verifyResponseWithServerPublicKey(
    AbstractRequest req, AbstractResponse response)
-     throws HandleException
86 - {
-
```

```
88  -      if(req.serverPubKeyBytes==null) {
    -        throw new HandleException(HandleException.SECURITY_ALERT,
90  -                                  "Unable to verify certified message:
     no pubkey associated with request");
    -      }
92  -
    -      // the request was certified so we should verify the signature
    here
94  -      PublicKey pubKey;
    -      try {
96  -          pubKey = Util.getPublicKeyFromBytes(req.serverPubKeyBytes,
    0);
    -      } catch (Exception e) {
98  -        throw new HandleException(HandleException.INVALID_VALUE,
    -                                  "Unable to extract public key",e);
100 -      }
    -
102 -      try {
    -        if(response.signature==null || response.signature.length<=0) {
104 -            throw new HandleException(HandleException.
    MISSING_OR_INVALID_SIGNATURE,
    -                   "Verification failed, missing signature.");
106 -        }
    -        if(!response.verifyMessage(pubKey)) {
108 -          throw new HandleException(HandleException.
    MISSING_OR_INVALID_SIGNATURE,
    -                                   "Verification failed.");
110 -        }
    -      } catch (Exception e) {
112 -        // e.printStackTrace();
    -        throw new HandleException(HandleException.
    MISSING_OR_INVALID_SIGNATURE,
114 -                                   "Unable to verify signature for
    message: "+response,e);
    -      }
116 -
    -      if (req.sessionInfo != null) req.sessionInfo.addSessionCounter(
    response.sessionCounter, true);
118 -    }
    -
120 -  private static void verifyRequestDigestIfNeeded(AbstractRequest
    req, AbstractResponse response) throws HandleException {
    -      // Make sure that the server is responding to the request as
    we sent it.
122 -      // This is because our request could have been modified on its
     way to
    -      // the server since requests aren't signed.  We get around
    that by having
124 -      // the server include a digest of the original request with
    its response.
    -      if(req.returnRequestDigest) {
```

```
126   -            byte requestDigest[] =
      -                    Util.doDigest(response.rdHashType, req.
      getEncodedMessageBody());
128   -            if(!Util.equals(requestDigest, response.requestDigest)) {
      -                throw new HandleException(HandleException.
      SECURITY_ALERT,
130   -                    "Message came back with invalid request digest
      .");
      -            }
132   -        }
      -    }
134   -
      -    /******************************************************************
136   -     * Shortcut to sendHdlUdpRequest(req, addr, port, null);
      -     ******************************************************************/
138   -    public AbstractResponse sendHdlUdpRequest(AbstractRequest req,
      -                                              InetAddress addr,
140   -                                              int port)
      -        throws HandleException
142   -    {
      -        return sendHdlUdpRequest(req, addr, port, null);
144   -    }
      -
146   -    private static void
      waitIfSiblingConnectedAndThrowHandleExceptionIfFinished(
      AbstractRequest req) throws HandleException {
      -        if(req.multithread) {
148   -            try {
      -
      waitIfSiblingConnectedAndThrowInterruptedExceptionIfFinished(req);
150   -            }
      -            catch(InterruptedException e) {
152   -                throw new HandleException(HandleException.
      OTHER_CONNECTION_ESTABLISHED,
      -                        HandleException.
      OTHER_CONNECTION_ESTABLISHED_STRING);
154   -            }
      -        }
156   -    }
      -
158   -    private static void
      waitIfSiblingConnectedAndThrowInterruptedExceptionIfFinished(
      AbstractRequest req) throws InterruptedException {
      -        if(req.multithread) {
160   -            if(req.completed.get()) throw new InterruptedException();
      -            if(!req.connectionLock.tryLock()) {
162   -                req.connectionLock.lockInterruptibly();
      -            }
164   -            req.connectionLock.unlock();
      -            if(req.completed.get()) throw new InterruptedException();
166   -        }
```

```
168  -    }
     -    private static void
     -    lockConnectionAndThrowHandleExceptionIfFinished(AbstractRequest
     -    req) throws HandleException {
170  -        if(req.multithread) {
     -            if(req.completed.get()) throw new HandleException(
     -    HandleException.OTHER_CONNECTION_ESTABLISHED, HandleException.
     -    OTHER_CONNECTION_ESTABLISHED_STRING);
172  -            try {
     -                req.connectionLock.lockInterruptibly();
174  -            }
     -            catch(InterruptedException e) {
176  -                throw new HandleException(HandleException.
     -    OTHER_CONNECTION_ESTABLISHED,
     -                        HandleException.
     -    OTHER_CONNECTION_ESTABLISHED_STRING);
178  -            }
     -            if(req.completed.get()) {
180  -                req.connectionLock.unlock();
     -                throw new HandleException(HandleException.
     -    OTHER_CONNECTION_ESTABLISHED, HandleException.
     -    OTHER_CONNECTION_ESTABLISHED_STRING);
182  -            }
     -        }
184  -    }
     -
186  -    public AbstractResponse sendHdlUdpRequest(AbstractRequest req,
     +
188  +public AbstractResponse sendHdlNdnRequest(AbstractRequest req,
     +                                          InetAddress addr,
190  +                                          int port,
     +                                          ResponseMessageCallback
     +                                               callback)
192  @@ -2457,14 +2327,15 @@
         {
194        config.startAutoUpdate(this);
          addr = config.mapLocalAddress(addr);
196  -     DatagramSocket socket = null;
     +     HdlDatagramSocket socket = null;
198       DatagramPacket[] packets = null;
          Exception lastException = null;
200
          // create the socket, set the timeout value
202       try {
            try {
204  -           socket = new DatagramSocket();
     +           socket = HdlDatagramSocket.getInstance(NDNClientName.
          getInstance().getClientName());
206  +           socket.addremotePeer(new Name("1234"));
            } catch (Exception e) {
```

```
208                 try { socket.close(); } catch (Exception e2){}
                    throw new HandleException(HandleException.INTERNAL_ERROR,
                       e);
210  @@ -2493,6 +2364,304 @@

212              // send out each request packet
                 try {
214  +             //socket.setSoTimeout(udpRetryScheme[attempt]);
     +             for(int packet = 0; packet < packets.length; packet++) {
216  +               socket.send(packets[packet]);
     +             }
218  +             whenToTimeout = System.currentTimeMillis() +
         udpRetryScheme[attempt];
     +           } catch (Exception e) {
220  +             throw new HandleException(HandleException.INTERNAL_ERROR,
     +                                   String.valueOf(e)+" sending NDN
         request to "+
222  +                                      Util.rfcIpRepr(addr));
     +           }
224  +
     +         // loop, waiting for packets until the timeout is reached or
226  +         // we have all of the packets, whichever comes first.
     +         byte returnMessage[] = null;
228  +         boolean packetsReceived[] = null;
     +         boolean haveAllPackets = false;
230  +         while(!haveAllPackets && System.currentTimeMillis() <=
         whenToTimeout) {
     +             DatagramPacket rspnsPkt = null;
232  +
     +             waitIfSiblingConnectedAndThrowHandleExceptionIfFinished(
         req);
234  +
     +             try {
236  +           rspnsPkt = socket.receive();
     +               if(rspnsPkt.getLength()<=0) continue;
238  +
     +               // need to decode the envelop data...
240  +               byte rspnsPktData[] = rspnsPkt.getData();
     +               int rspnsPktDataLen = rspnsPkt.getLength();
242  +
     +               Encoder.decodeEnvelope(rspnsPktData, rcvEnvelope);
244  +
     +               // if we got someone else's packet, ignore it.
246  +               if(rcvEnvelope.requestId != req.requestId) continue;
     +
248  +               if(packetsReceived==null) {
     +                 int numPkts =
250  +                   rcvEnvelope.messageLength / maxUDPDataSize;
     +                 if((rcvEnvelope.messageLength % maxUDPDataSize) != 0)
252  +                   numPkts++;
     +                 packetsReceived = new boolean[numPkts];
```

```
254 +                      for(int pr=0; pr<packetsReceived.length; pr++)
    +                        packetsReceived[pr] = false;
256 +                      returnMessage = new byte[rcvEnvelope.messageLength];
    +                    }
258 +
    +                    packetsReceived[rcvEnvelope.messageId] = true;
260 +                    System.arraycopy(rspnsPktData, Common.
    MESSAGE_ENVELOPE_SIZE,
    +                                     returnMessage,
262 +                                     rcvEnvelope.messageId*maxUDPDataSize,
    +                                     rspnsPktDataLen-Common.
    MESSAGE_ENVELOPE_SIZE);
264 +                    haveAllPackets = true;
    +                    for(int pr=0; pr<packetsReceived.length; pr++) {
266 +                      if(!packetsReceived[pr]) {
    +                        haveAllPackets = false;
268 +                        pr = packetsReceived.length;
    +                      }
270 +                    }
    +
272 +                    if(haveAllPackets) {
    +
274 +                      if(req.multithread) req.connectionLock.
    lockInterruptibly();
    +
276 +                        // decrypt the message using pre-established session
    information
    +                        if (rcvEnvelope.encrypted) {
278 +                          // try session key decryption first ...
    +                          ClientSideSessionInfo sessionInfo = req.sessionInfo;
280 +                          if(sessionInfo==null)
    +                            throw new HandleException(HandleException.
    INCOMPLETE_SESSIONSETUP,
282 +                                                      "Cannot decrypt message
    without a session");
    +
284 +                          if(traceMessages)
    +                            System.err.println("Decrypting UDP message: "+
    rcvEnvelope);
286 +
    +                          returnMessage = sessionInfo.decryptBuffer(
    returnMessage, 0, returnMessage.length);
288 +                          rcvEnvelope.encrypted = false;
    +                          rcvEnvelope.messageLength = returnMessage.length;
290 +                        }
    +
292 +                        // parse the message
    +                        AbstractResponse response =
294 +                        (AbstractResponse)Encoder.decodeMessage(
    returnMessage, 0, rcvEnvelope);
    +
```

```
296 +                    if(traceMessages)
    +                        System.err.println("     received HDL-UDP response: "
    +response);
298 +
    +                    checkSignatureIfNeeded(req, response);
300 +
    +                    if(callback!=null) {
302 +                        callback.handleResponse(response);
    +                    }
304 +                    return response;
    +                }
306 +            } catch(InterruptedException e) {
    +                throw new HandleException(HandleException.
    OTHER_CONNECTION_ESTABLISHED,
308 +                        HandleException.
    OTHER_CONNECTION_ESTABLISHED_STRING);
    +            } catch (Exception e) {
310 +                lastException = e;
    +            }
312 +        }
    +    }
314 +    } finally {
    +        if(socket!=null) {
316 +            try { socket.close(); } catch (Exception e){}
    +        }
318 +    }
    +
320 +    if(lastException!=null) {
    +        if(lastException instanceof HandleException)
322 +            throw (HandleException)lastException;
    +        else
324 +            throw new HandleException(HandleException.
    CANNOT_CONNECT_TO_SERVER,
    +                                    addr+": "+lastException.toString()
    );
326 +    }
    +    throw new HandleException(HandleException.
    CANNOT_CONNECT_TO_SERVER,
328 +                                "Unable to connect to server: "+addr);
    +  }
330 +
+private boolean expectStreamingResponse(AbstractRequest req) {
332 +        return req.opCode == AbstractMessage.OC_RETRIEVE_TXN_LOG ||
    req.opCode == AbstractMessage.OC_DUMP_HANDLES
    +            || (req instanceof ChallengeAnswerRequest &&
    expectStreamingResponse(((ChallengeAnswerRequest)req).
    originalRequest));
334 +  }
    +
336 +  /**
    +   * Verify response message with the pre-established session key.
```

```
338 +   */
    + private final boolean verifyResponseWithSessionKey(AbstractRequest
    +  req, AbstractResponse response)
340 +    throws HandleException
    + {
342 +    boolean veriPass = false;
    +    if (req == null || response == null) return false;
344 +
    +    try {
346 +        veriPass = response.verifyMessage(req.sessionInfo.
    + getSessionKey());
    +    } catch (HandleException e) {
348 +        throw e;
    +    } catch (Exception e){
350 +        throw new HandleException(
    +                HandleException.MISSING_OR_INVALID_SIGNATURE,
352 +                "Error verifying MAC code",e);
    +    }
354 +    if(veriPass) {
    +        req.sessionInfo.addSessionCounter(response.sessionCounter,
    + true);
356 +    }
    +    return veriPass;
358 + }
    +
360 +
    + /** This function verifies the integrity of a response given the
    + request
362 +     that it is for.  The public key of the server is attached to
    + the
    +     request so that this can verify the signature of the response.
    +  This
364 +     function also checks the digest of the request that was
    + included
    +     (if requested) in the response. */
366 + private static final void verifyResponseWithServerPublicKey(
    + AbstractRequest req, AbstractResponse response)
    +    throws HandleException
368 + {
    +
370 +    if(req.serverPubKeyBytes==null) {
    +        throw new HandleException(HandleException.SECURITY_ALERT,
372 +                                  "Unable to verify certified message:
    +  no pubkey associated with request");
    +    }
374 +
    +    // the request was certified so we should verify the signature
    + here
376 +    PublicKey pubKey;
    +    try {
378 +        pubKey = Util.getPublicKeyFromBytes(req.serverPubKeyBytes,
```

```
         0);
     + } catch (Exception e) {
380  +     throw new HandleException(HandleException.INVALID_VALUE,
     +                               "Unable to extract public key",e);
382  + }
     +
384  + try {
     +   if(response.signature==null || response.signature.length<=0) {
386  +       throw new HandleException(HandleException.
     MISSING_OR_INVALID_SIGNATURE,
     +                 "Verification failed, missing signature.");
388  +   }
     +   if(!response.verifyMessage(pubKey)) {
390  +       throw new HandleException(HandleException.
     MISSING_OR_INVALID_SIGNATURE,
     +                               "Verification failed.");
392  +   }
     + } catch (Exception e) {
394  +   // e.printStackTrace();
     +   throw new HandleException(HandleException.
     MISSING_OR_INVALID_SIGNATURE,
396  +                            "Unable to verify signature for
     message: "+response,e);
     + }
398  +
     + if (req.sessionInfo != null) req.sessionInfo.addSessionCounter(
     response.sessionCounter, true);
400  + }
     +
402  + private static void verifyRequestDigestIfNeeded(AbstractRequest
     req, AbstractResponse response) throws HandleException {
     +   // Make sure that the server is responding to the request as
     we sent it.
404  +   // This is because our request could have been modified on its
      way to
     +   // the server since requests aren't signed.  We get around
     that by having
406  +   // the server include a digest of the original request with
     its response.
     +   if(req.returnRequestDigest) {
408  +       byte requestDigest[] =
     +               Util.doDigest(response.rdHashType, req.
     getEncodedMessageBody());
410  +       if(!Util.equals(requestDigest, response.requestDigest)) {
     +           throw new HandleException(HandleException.
     SECURITY_ALERT,
412  +                       "Message came back with invalid request digest
     .");
     +       }
414  +   }
     + }
```

```
416 +
    +    /*****************************************************************
418 +     * Shortcut to sendHdlUdpRequest(req, addr, port, null);
    +     *****************************************************************/
420 +    public AbstractResponse sendHdlUdpRequest(AbstractRequest req,
    +                                              InetAddress addr,
422 +                                              int port)
    +      throws HandleException
424 +    {
    +      return sendHdlUdpRequest(req, addr, port, null);
426 +    }
    +
428 +    private static void
       waitIfSiblingConnectedAndThrowHandleExceptionIfFinished(
       AbstractRequest req) throws HandleException {
    +        if(req.multithread) {
430 +            try {
    +
       waitIfSiblingConnectedAndThrowInterruptedExceptionIfFinished(req);
432 +            }
    +            catch(InterruptedException e) {
434 +                throw new HandleException(HandleException.
       OTHER_CONNECTION_ESTABLISHED,
    +                         HandleException.
       OTHER_CONNECTION_ESTABLISHED_STRING);
436 +            }
    +        }
438 +    }
    +
440 +    private static void
       waitIfSiblingConnectedAndThrowInterruptedExceptionIfFinished(
       AbstractRequest req) throws InterruptedException {
    +        if(req.multithread) {
442 +            if(req.completed.get()) throw new InterruptedException();
    +            if(!req.connectionLock.tryLock()) {
444 +                req.connectionLock.lockInterruptibly();
    +            }
446 +            req.connectionLock.unlock();
    +            if(req.completed.get()) throw new InterruptedException();
448 +        }
    +    }
450 +
    +    private static void
       lockConnectionAndThrowHandleExceptionIfFinished(AbstractRequest
       req) throws HandleException {
452 +        if(req.multithread) {
    +            if(req.completed.get()) throw new HandleException(
       HandleException.OTHER_CONNECTION_ESTABLISHED, HandleException.
       OTHER_CONNECTION_ESTABLISHED_STRING);
454 +            try {
    +                req.connectionLock.lockInterruptibly();
```

```
456  +            }
     +          catch ( InterruptedException e) {
458  +             throw new HandleException ( HandleException .
         OTHER_CONNECTION_ESTABLISHED ,
     +                   HandleException .
         OTHER_CONNECTION_ESTABLISHED_STRING );
460  +          }
     +          if(req.completed.get()) {
462  +             req.connectionLock.unlock ();
     +             throw new HandleException ( HandleException .
         OTHER_CONNECTION_ESTABLISHED , HandleException .
         OTHER_CONNECTION_ESTABLISHED_STRING );
464  +          }
     +       }
466  +    }
     +
468  +    public AbstractResponse sendHdlUdpRequest ( AbstractRequest req,
     +                                               InetAddress addr,
470  +                                               int port,
     +                                               ResponseMessageCallback
         callback)
472  +      throws HandleException
     +    {
474  +      config.startAutoUpdate (this);
     +      addr = config.mapLocalAddress (addr);
476  +      DatagramSocket socket = null;
     +      DatagramPacket [] packets = null;
478  +      Exception lastException = null;
     +
480  +      // create the socket , set the timeout value
     +      try {
482  +        try {
     +            socket = new DatagramSocket ();
484  +        } catch (Exception e) {
     +            try { socket.close (); } catch (Exception e2){}
486  +            throw new HandleException ( HandleException .INTERNAL_ERROR ,
         e);
     +        }
488  +
     +        MessageEnvelope rcvEnvelope = new MessageEnvelope ();
490  +        long whenToTimeout ;
     +
492  +        for(int attempt =0; attempt < udpRetryScheme .length; attempt ++)
         {
     +
494  +          if(packets == null) {
     +              packets = getUdpPacketsForRequest (req,addr ,port);
496  +          } else {
     +              // if sending UDP requests in a session , resign each
         attempt to avoid duplicate session counters
498  +              if (req.sessionInfo !=null && req.authInfo !=null && req.
```

```
      hasEqualOrGreaterVersion (2 ,5)) {
+                  req.signMessageForSession ();
+                  packets = getUdpPacketsForRequest (req ,addr ,port );
+              }
+          }
+          waitIfSiblingConnectedAndThrowHandleExceptionIfFinished (req)
    ;
+
+          if (traceMessages ) {
+            System.err.println ("   sending HDL -UDP request ("+req+") to
    "+
+                              Util.rfcIpPortRepr (addr ,port ));
+          }
+
+          // send out each request packet
+          try {
            socket.setSoTimeout (udpRetryScheme [attempt ]);
            for (int packet = 0; packet < packets.length; packet ++) {
              socket.send (packets [packet ]);
```

Listing A.6: Patch 5 for Handle HDLLib integration of NDN-enabled native communication using NDN PID push (Native Handle Communication Subsystem)

```
   --- /hsj-8.1.1/src/net/handle/hdllib/Interface.java
 2 +++ /ndnhandle/src/net/handle/hdllib/Interface.java
   @@ -21,6 +21,7 @@
 4    public static final byte SP_HDL_TCP = 1;
      public static final byte SP_HDL_HTTP = 2;
 6    public static final byte SP_HDL_HTTPS = 3;
   +  public static final byte SP_HDL_NDN = 4;
 8
      public byte type;                 // OUT_OF_SERVICE, ADMIN, QUERY,
         ADMIN_AND_QUERY
10    public int port;                  // usually 2641
   @@ -88,6 +89,8 @@
12       return "TCP";
       case SP_HDL_UDP:
14       return "UDP";
   +   case SP_HDL_NDN:
16 +     return "NDN";
       default:
18       return "UNKNOWN";
       }
```

Listing A.7: Patch 5 for Handle Server integration of NDN-enabled native communication
using NDN PID push (Native Handle Communication Subsystem)

```
1  --- /hsj-8.1.1/src/net/handle/server/HdlUdpPendingRequest.java
   +++ /ndnhandle/src/net/handle/server/HdlUdpPendingRequest.java
3  @@ -17,8 +17,8 @@
   -class HdlUdpPendingRequest {
5  -  String idString;
   +public class HdlUdpPendingRequest {
7  +  public String idString;
      boolean gotPacket[] = null;
9     byte message[];

11 @@ -26,11 +26,11 @@
   -  static final String getRequestId(InetAddress addr, int requestId)
      {
13 +  public static final String getRequestId(InetAddress addr, int
      requestId) {
      return String.valueOf(requestId)+'@'+Util.rfcIpRepr(addr);
15    }

17 -  HdlUdpPendingRequest(String idString,
   +  public HdlUdpPendingRequest(String idString,
19                         MessageEnvelope firstEnv,
                           DatagramPacket firstPkt)
21    {
   @@ -50,7 +50,7 @@
23 -  boolean isComplete() {
   +  public boolean isComplete() {
25    for(int i=0; i<gotPacket.length; i++)
        if(!gotPacket[i]) return false;
27    return true;
   @@ -61,7 +61,7 @@
29 -  void addPacket(MessageEnvelope env, DatagramPacket pkt) {
   +  public void addPacket(MessageEnvelope env, DatagramPacket pkt) {
31    gotPacket[env.messageId] = true;

33    if(message==null) message = new byte[env.messageLength];
```

231

Listing A.8: Patch 6 for Handle Server integration of NDN-enabled native communication using NDN PID push (Native Handle Communication Subsystem)

```
--- /hsj-8.1.1/src/net/handle/server/NetworkInterface.java
+++ /ndnhandle/src/net/handle/server/NetworkInterface.java
@@ -36,6 +36,7 @@
  public static final String INTFC_HDLTCP = "hdl_tcp";
  public static final String INTFC_HDLUDP = "hdl_udp";
  public static final String INTFC_HDLHTTP = "hdl_http";
+ public static final String INTFC_NDN = "hdl_ndn";
  public static final String INTFC_DNSUDP = "dns_udp";
  public static final String INTFC_DNSTCP = "dns_tcp";

@@ -49,6 +50,8 @@
      return new HdlUdpInterface(main, (StreamTable)configTable.get(
          frontEndLabel+"_config"));
    } else if(frontEndLabel.startsWith(INTFC_HDLTCP)) {
      return new HdlTcpInterface(main, (StreamTable)configTable.get(
          frontEndLabel+"_config"));
+    } else if(frontEndLabel.startsWith(INTFC_NDN)) {
+      return new HdlNDNInterface(main, (StreamTable)configTable.get(
    frontEndLabel+"_config"));
    } else if(frontEndLabel.startsWith(INTFC_HDLHTTP)) {
        // dealt with elsewhere
        return null;
        if(message==null) message = new byte[env.messageLength];
```

A.2.2 Additions for NDN-enabled Native Handle Communication Using NDN PID Push

Following additions have been added to the Handle Server 8.1.1 code base [134], in order to realize native Handle protocol communication over a NDN network using NDN PID push. These additions extend the Handle code base by three files.

Listing A.9: Addition 1 for Handle HDLLib integration of NDN-enabled native communication using NDN PID push

```java
// File /ndnhandle/src/net/handle/hdllib/NNDNClientName.java
package net.handle.hdllib;

import java.math.BigInteger;
import java.security.SecureRandom;

import net.named_data.jndn.Name;

public final class NDNClientName {
  private static NDNClientName instance;

  private Name clientname;
  private final SecureRandom random = new SecureRandom();

  public static synchronized NDNClientName getInstance() {
    if (NDNClientName.instance == null) {
      NDNClientName.instance = new NDNClientName();
    }
    return NDNClientName.instance;
  }

  private NDNClientName() {
    this.clientname = new Name("cl-" + new BigInteger(130, random).
      toString(32));
  }

  public Name getClientName() {
    return this.clientname;
  }
}
```

Listing A.10: Addition 2 for Handle Server integration of NDN-enabled native communication using NDN PID push

```java
1   // File /ndnhandle/src/net/handle/server/HdlNDNInterface.java

3   // This implementation uses methods and code from the
    // the file net.handle.server.HdlUdpInterface
5
    package net.handle.server;
7
    import net.cnri.util.GrowBeforeQueueThreadPoolExecutor;
9   import net.cnri.util.StreamTable;
    import net.handle.hdllib.*;
11  import net.named_data.jndn.Name;
    import java.net.*;
13  import java.util.concurrent.ConcurrentHashMap;
    import java.util.concurrent.ConcurrentMap;
15  import java.util.concurrent.LinkedBlockingQueue;
    import java.util.concurrent.TimeUnit;
17  import de.gwdg.ndn.hdldatagram.transport.HdlDatagramSocket;

19  public class HdlNDNInterface extends NetworkInterface {
      private InetAddress bindAddress;
21    private int threadLife = 500;
      private int bindPort = 2641;
23    private int numThreads = 10;
      private int maxHandlers = 200;
25    private boolean logAccesses=false;
      private boolean trackThreads=false;
27    private HdlDatagramSocket dsocket = null;
      private boolean keepServing = true;
29
      private ConcurrentMap<String,HdlUdpPendingRequest> pendingRequests;
31
      public HdlNDNInterface(Main main, StreamTable config) throws
          Exception {
33      super(main);
        pendingRequests = new ConcurrentHashMap<String,
          HdlUdpPendingRequest>();
35      init(config);
      }
37
      public byte getProtocol() { return Interface.SP_HDL_NDN; }
39    public int getPort() { return bindPort; }

41    private void init(StreamTable config) throws Exception {

43      try { // get the number of thread (default is 10);
          numThreads = Integer.parseInt((String)config.get("num_threads")
            );
45      } catch (Exception e) {
          main.logError(ServerLog.ERRLOG_LEVEL_NORMAL, "unspecified
```

```
               thread count, using default: " + numThreads);
47     }

49     try { // get the max backlog size...
           if (config.containsKey("max_handlers")) {
51             maxHandlers = Integer.parseInt((String)config.get("
                   max_handlers"));
           }
53     } catch (Exception e) {
         main.logError(ServerLog.ERRLOG_LEVEL_NORMAL, "unspecified
             max_handlers count, using default: " + maxHandlers);
55     }

57     try { // get the maximum thread life (default is 500)
         if(config.containsKey("thread_life")) {
59         threadLife = Integer.parseInt((String)config.get("thread_life
             "));
         }
61     } catch (Exception e) {
         main.logError(ServerLog.ERRLOG_LEVEL_NORMAL, "Invalid thread
             life, using default: " + threadLife);
63     }
       trackThreads = config.getBoolean("track_threads");
65     // check if we should log accesses or not...
       logAccesses = config.getBoolean("log_accesses");
67     super.initialize();
     }

69


71   /******************************************************************
      * Tells the interface to finish up the current operation and
73    * stop listening for new connections.
      ******************************************************************/
75   protected void stopService() {
       keepServing = false;
77     try {
         dsocket.close();
79     } catch (Exception e) {}
     }

81


83   /******************************************************************
      * Tells the interface to listen for incoming requests until
85    * stopService() is called.
      ******************************************************************/
87   public void serveRequests() {
       keepServing = true;
89     try {
         dsocket = HdlDatagramSocket.getInstance(new Name("1234"),
           InetAddress.getByName("0.0.0.0"));
91     } catch (Exception e) {
```

```
          main.logError(ServerLog.ERRLOG_LEVEL_FATAL, String.valueOf(this
              .getClass()) + ": Error setting up server socket: " + e);
 93       return;
        }

 95


 97     handlerPool = new GrowBeforeQueueThreadPoolExecutor(numThreads,
            maxHandlers, 1, TimeUnit.MINUTES, new LinkedBlockingQueue<
            Runnable>());
        //    handlerPool.setHandlerLife(threadLife);
 99     System.out.println("Starting NDN request handlers...");
        try { System.out.flush(); } catch (Exception e) {}
101     long reqCount = 0;
        long recvTime = 0;
103     while(keepServing) {
          try {
105         DatagramPacket dPacket =  dsocket.receive();
            recvTime = System.currentTimeMillis();
107         handlerPool.execute(new HdlNDNRequestHandler(main, dsocket,
                this, logAccesses, dPacket, recvTime));
          } catch (Exception e) {
109         if(keepServing) {
              main.logError(ServerLog.ERRLOG_LEVEL_REALBAD, "" + this.
                  getClass() + ": Error handling request: " + e);
111           e.printStackTrace(System.err);
            }
113       }
        }
115     try {
          dsocket.close();
117     } catch (Exception e) { }
      }
119
       // Multipacket Listener is shared with the original
121     // UDP Interface Implementation of the Handle server
      HdlUdpPendingRequest addMultiPacketListener(MessageEnvelope env,
          DatagramPacket pkt, InetAddress addr) {
123     String id = HdlUdpPendingRequest.getRequestId(addr, env.requestId
            );
        HdlUdpPendingRequest req = null;
125
        // check the list of pending requests to see if someone else is
127     // already listening for this request
        HdlUdpPendingRequest existingReq = pendingRequests.get(id);
129     if(existingReq==null) {
          req = new HdlUdpPendingRequest(id, env, pkt);
131       existingReq = pendingRequests.putIfAbsent(id, req);
        }
133     if(existingReq!=null) { // the request is already pending... and
            already has a handler.
          // so we will add this packet to the request and go away
```

```
135      existingReq.addPacket(env, pkt);
         if(existingReq.isComplete()) {
137        // notify the handler that the request is complete
           synchronized(existingReq) {
139          existingReq.notifyAll();   // could just be a notify() call
               .. shouldn't matter
           }
141      }
         return null;
143    }

145    // this is the first packet received for a new request
       // go to sleep until the rest of the request comes in...
147    // at which time, someone will wake us up.  Or just
       // timeout after a certain period.
149    synchronized(req) {
         // wait for a maximum of 5 seconds
151      try{ req.wait(5000); } catch (Exception e) {}
         // remove the request since we are handling (or ignoring) it
153      pendingRequests.remove(req.idString);
       }
155    // if the request is complete, return it.  Otherwise, throw it
         out.
       if(!req.isComplete()) return null;
157    return req;
     }
159 }
```

Listing A.11: Addition 3 for Handle Server integration of NDN-enabled native communication using NDN PID push

```java
// File /ndnhandle/src/net/handle/server/HdlNDNRequestHandler.java

// This implementation uses methods and code from the
// the file net.handle.server.HdlUdpRequestHandler

package net.handle.server;

import net.handle.hdllib.*;
import net.named_data.jndn.Name;
import java.net.*;
import de.gwdg.ndn.hdldatagram.transport.HdlDatagramSocket;

/***********************************************************************
 * An HdlNDNRequestHandler object will handle requests submitted
     using
 * the NDN handle protocol.  The request will be processed using the
 * server object and a response will be returned using the NDN handle
 * protocol.
 ***********************************************************************/

public class HdlNDNRequestHandler implements Runnable,
    ResponseMessageCallback {
  private DatagramPacket packet;
  private HdlDatagramSocket dsocket;
  private AbstractServer server;
  private Main main;
  private HdlNDNInterface listener;

  private boolean logAccesses = false;
  private MessageEnvelope envelope = new MessageEnvelope();

  private AbstractRequest currentRequest;
  private long recvTime;

  public static final String ACCESS_TYPE = "NDN:HDL";
    public static final byte MSG_INVALID_MSG_SIZE[] = Util.
        encodeString("Invalid message length");
  private static final byte[] MSG_CANNOT_STREAM_NDN = Util.
      encodeString("Cannot stream NDD messages");

  public HdlNDNRequestHandler(Main main, HdlDatagramSocket dsock,
      HdlNDNInterface listener, boolean logAccesses, DatagramPacket
      packet, long recvTime) {
    this.main = main;
    this.server = main.getServer();
    this.dsocket = dsock;
    this.logAccesses = logAccesses;
    this.listener = listener;
    this.packet = packet;
```

```
       this.recvTime = recvTime;
45   }

47   public void run() {
       boolean multiPartRequest = false;
49     byte pkt[] = null;
       int pktLen = 0;
51     int offset = Common.MESSAGE_ENVELOPE_SIZE;
       try {
53       pktLen = packet.getLength();
         pkt = packet.getData();
55       Encoder.decodeEnvelope(pkt, envelope);
         if(envelope.messageLength > Common.MAX_MESSAGE_LENGTH ||
           envelope.messageLength < 0) {
57         handleResponse(new ErrorResponse(AbstractMessage.OC_RESERVED,
               AbstractMessage.RC_PROTOCOL_ERROR,
59             MSG_INVALID_MSG_SIZE));
           return;
61       }

63       if(envelope.truncated) {
           HdlUdpPendingRequest req = listener.addMultiPacketListener(
               envelope, packet, packet.getAddress());
65         if(req==null) return;
           pkt = req.getMessage();
67         offset = 0;
         }

69
         //decrypt incoming request if it says so
71       if (envelope.encrypted) {
           if (envelope.sessionId > 0) {
73           ServerSideSessionInfo sssinfo = null;
             if (server instanceof HandleServer) {
75             sssinfo = ((HandleServer)server).getSession(envelope.
                 sessionId);
               if (sssinfo != null) {
77               try {
                   pkt = sssinfo.decryptBuffer(pkt,offset,envelope.
                     messageLength);
79                 envelope.encrypted = false;
                   envelope.messageLength = pkt.length;
81                 offset = 0;
                 }
83               catch (Exception e) {
                   main.logError(ServerLog.ERRLOG_LEVEL_REALBAD, "
                     Exception decrypting request: " + e);
85                 e.printStackTrace();
                   System.err.println("Exception decrypting request with
                     session key: " + e.getMessage());
87                 handleResponse(new ErrorResponse(AbstractMessage.
                     OC_RESERVED,
```

```
                        AbstractMessage.RC_SESSION_FAILED ,
89                      Util.encodeString("Exception decrypting request
                            with session key " + e)));
                      return;
91                  }
                } else {
93              // sssinfo == null, maybe time out!
                main.logError(ServerLog.ERRLOG_LEVEL_REALBAD , "Session
                    information not available or time out. Unable to
                    decrypt request message");
95              System.err.println("Session information not available
                    or time out. Unable to decrypt request message.");
                handleResponse(new ErrorResponse(AbstractMessage.
                    OC_RESERVED ,
97                   AbstractMessage.RC_SESSION_TIMEOUT ,
                     Util.encodeString("Session information not
                         available or time out. Unable to decrypt request
                         message.")));
99              return;
                }
101         } else {
                // serverSessionMan == null
103         main.logError(ServerLog.ERRLOG_LEVEL_REALBAD , "Session
                manager not available. Unable to decrypt request
                message.");
            System.err.println("Session manager not available.
                Request message not decrypted.");
105         handleResponse(new ErrorResponse(AbstractMessage.
                OC_RESERVED ,
                 AbstractMessage.RC_SESSION_FAILED ,
107              Util.encodeString("Session manager not available.
                     Unable to decrypt request message.")));
            return;
109         }
        } else {
111     main.logError(ServerLog.ERRLOG_LEVEL_REALBAD , "Invalid
            session id. Request message not decrypted.");
        System.err.println("Invalid session id. Request message not
            decrypted.");
113     handleResponse(new ErrorResponse(AbstractMessage.
            OC_RESERVED ,
            AbstractMessage.RC_SESSION_FAILED ,
115          Util.encodeString("Invalid session id. Unable to
                 decrypt request message.")));
        return;
117     }
    }

119
    if(envelope.messageLength < 24) {
121     handleResponse(new ErrorResponse(AbstractMessage.
            OC_RESERVED , AbstractMessage.RC_PROTOCOL_ERROR ,
```

```
                          MSG_INVALID_MSG_SIZE ));
                  return ;
123           }

125           int opCode = Encoder . readOpCode ( pkt , offset );
              if ( opCode == 0) {
127               handleResponse ( new ErrorResponse ( AbstractMessage .
                      OC_RESERVED ,
                          AbstractMessage . RC_PROTOCOL_ERROR ,
129                       Util . encodeString (" Unknown opCode in message : " +
                              opCode )));
                  return ;
131           }

133           currentRequest = ( AbstractRequest ) Encoder . decodeMessage ( pkt ,
                  offset , envelope );
              String errMsg = listener . canProcessMsg ( currentRequest );
135           if ( errMsg != null ) {
                main . logError ( ServerLog . ERRLOG_LEVEL_REALBAD , errMsg );
137             handleResponse ( new ErrorResponse ( currentRequest . opCode ,
                    AbstractMessage . RC_PROTOCOL_ERROR ,
139               Util . encodeString ( errMsg )));
                return ;
141           }
              server . processRequest ( currentRequest , this );
143       } catch ( Throwable e) {
            handleResponse ( new ErrorResponse ( AbstractMessage . OC_RESERVED ,
                AbstractMessage . RC_ERROR , Util . encodeString (" Server error
                processing request , see server logs ")));
145         main . logError ( ServerLog . ERRLOG_LEVEL_REALBAD , String . valueOf (
                this . getClass ()) + ": Exception processing request : " + e);
            e. printStackTrace ( System . err );
147       }
        }

149
/* **********************************************************************
151  * Handle (log) any messages that are reported by the upstream
       message
     * provider .
153  **********************************************************************/
     public void handleResponseError ( String error ) {
155     main . logError ( ServerLog . ERRLOG_LEVEL_NORMAL , String . valueOf ( this .
           getClass ()) + ": Server error : " + error );
     }

157
/* **********************************************************************
159  * Encode and send the response
     **********************************************************************/
161   public void handleResponse ( AbstractResponse response ) {
        try {
163         byte msg [] = response . getEncodedMessage ();
```

```
        //when to encrypt? right before sending it out! after the
            credential portion is formed!
165     //encrypt response here if the request asks for encryption
        //and set the flag in envelop if successfull
167     boolean encrypted = false;
        if (response.sessionId > 0 && (response.encrypt || response.
            shouldEncrypt())){
169       ServerSideSessionInfo sssinfo = null;
          if (server instanceof HandleServer) {
171         sssinfo = ((HandleServer)server).getSession(response.
                sessionId);
            if (sssinfo != null) {
173           try {
                msg = sssinfo.encryptBuffer(msg, 0, msg.length);
175             encrypted = true;
              } catch (Exception e) {
177             main.logError(ServerLog.ERRLOG_LEVEL_NORMAL, "Exception
                    encrypting response: " + e);
                System.err.println("Exception encrypting message with
                    session key: " + e.getMessage());
179             encrypted = false;
              }
181         } // sssinfo != null
          } else {
183         // serverSessionMan == null
            main.logError(ServerLog.ERRLOG_LEVEL_NORMAL, "Session
                manager not available. Message not encrypted.");
185         System.err.println("Session manager not available. Message
                not encrypted.");
            encrypted = false;
187       }
        }
189
        //set the envelop flag for encryption
191     envelope.encrypted = encrypted;
        envelope.messageLength = msg.length;  //get the length after
            encryption
193     envelope.messageId = 0;
        envelope.requestId = response.requestId;
195     envelope.sessionId = response.sessionId;
        envelope.protocolMajorVersion = response.majorProtocolVersion;
197     envelope.protocolMinorVersion = response.minorProtocolVersion;
            envelope.suggestMajorProtocolVersion = response.
                suggestMajorProtocolVersion;
199         envelope.suggestMinorProtocolVersion = response.
                suggestMinorProtocolVersion;
        if(msg.length > Common.MAX_UDP_DATA_SIZE) {
201       // split the response into multiple pieces and send it
          int bytesRemaining = msg.length;
203       while(bytesRemaining>0) {
            byte buf[];
```

```
205          if(bytesRemaining <= Common.MAX_UDP_DATA_SIZE) {
               buf = new byte[bytesRemaining+Common.
                  MESSAGE_ENVELOPE_SIZE];
207            System.arraycopy(msg, msg.length - bytesRemaining,
                   buf, Common.MESSAGE_ENVELOPE_SIZE,
209               bytesRemaining);
             } else {
211            buf = new byte[Common.MAX_UDP_DATA_SIZE+Common.
                  MESSAGE_ENVELOPE_SIZE];
               System.arraycopy(msg, msg.length - bytesRemaining,
213               buf, Common.MESSAGE_ENVELOPE_SIZE,
                  Common.MAX_UDP_DATA_SIZE);
215          }
             Encoder.encodeEnvelope(envelope,buf);
217          dsocket.send(new DatagramPacket(buf, buf.length,
                 packet.getAddress(),
219              packet.getPort()));
             bytesRemaining -= Common.MAX_UDP_DATA_SIZE;
221          envelope.messageId++;
           }
223      } else {
           // all of the response fits in one packet, so let's send it..
225        byte buf[] = new byte[msg.length + Common.
              MESSAGE_ENVELOPE_SIZE];
           Encoder.encodeEnvelope(envelope, buf);
227        System.arraycopy(msg,0,buf,Common.MESSAGE_ENVELOPE_SIZE,msg.
              length);
           dsocket.send(new DatagramPacket(buf, buf.length,
229          packet.getAddress(),
             packet.getPort()));
231      }
     } catch (Exception e) {
233          String clientString = "";
             try {
235              clientString = " to " + Util.rfcIpRepr(dsocket.getVip
                    (new Name("1234")));
             } catch (Exception ex) {
237              // ignore
             }
239          main.logError(ServerLog.ERRLOG_LEVEL_REALBAD, String.
                valueOf(this.getClass()) + ": Exception sending
                response" + clientString + ": " + e);
       e.printStackTrace(System.err);
241      }
     if(logAccesses){
243      if (currentRequest != null) {
             long time = System.currentTimeMillis() - recvTime;
245          main.logAccess(ACCESS_TYPE + "(" + currentRequest.
                suggestMajorProtocolVersion + "." + currentRequest.
                suggestMinorProtocolVersion + ")",
                   packet.getAddress(),
```

```
247                             currentRequest.opCode,
                                (response != null ? response.responseCode :
                                    AbstractMessage.RC_ERROR),
249                             Util.getAccessLogString(currentRequest), time);
                }
251         }
        }
253 }
```

A.2.3 Patches for PID Publishing Using NDN PID Pull

Following patches have been applied to the Handle Server 8.1.1 code base [134], in order to realize the PID publishing subsystem that leverages NDN PID pull for communication. These patches extend the Handle code base in one file.

Listing A.12: Patch 7 for PID Publishing subsystem integration using NDN PID pull

```
1  --- /hsj-8.1.1/src/net/handle/hdllib/HandleValue.java
   +++ /ndnhandle/src/net/handle/hdllib/HandleValue.java
3    @@ -8,13 +8,16 @@
      \***********************************************************/
5
     package net.handle.hdllib;
7   +import java.io.Serializable;
     import java.util.*;
9
     import net.cnri.util.FastDateFormat;
11
      /** Represents a single handle value */
13  -public class HandleValue {
    -
15  +public class HandleValue implements Serializable {
    +
17  +
    +  private static final long serialVersionUID = 1570689295099260332
       L;
19     public static final byte SUBTYPE_SEPARATOR = (byte)'.';
       public static final byte TTL_TYPE_RELATIVE = 0;
21     public static final byte TTL_TYPE_ABSOLUTE = 1;
```

A.2.4 Additions for PID Publishing Using NDN PID Pull (Server)

Following additions have been added to the Handle Server 8.1.1 code base [134], in order to realize the PID publishing subsystem on the server side that leverages NDN PID pull for communication.

Listing A.13: Addition 4 for Handle Server integration of NDN PID pull (Starter)

```
1  package de.gwdg.ndn.hdlpull.server;

3  import net.cnri.util.StreamTable;
   import net.handle.hdllib.*;
5  import net.handle.server.*;

7  public class Main {

9    public static void main(String argv[]) throws Exception {
       if (argv == null || argv.length < 1) {
11       System.err.println("usage: java de.gwdg.ndn.hdlpull.server <
            server-directory>");
         return;
13     }

15     java.io.File serverDir = new java.io.File(argv[0]);
       StreamTable serverInfo = new StreamTable();
17     serverInfo.readFromFile(new java.io.File(serverDir, "config.dct")
         );
       serverInfo = (StreamTable) serverInfo.get("server_config");
19     HandleStorage storage = HandleStorageFactory.getStorage(serverDir
         , serverInfo, true, true);
       Boolean caseSensitive = serverInfo.getBoolean("case_sensitive");
21     DBHelper dbhelper = new DBHelper(storage, caseSensitive);
       new MainThread(dbhelper).start();
23     System.out.println("NDN PID Pull :: Server is running");
     }

25 }
```

Listing A.14: Addition 5 for Handle Server integration of NDN PID pull (NFD Event
Handling)

```
package de.gwdg.ndn.hdlpull.server;

import java.io.IOException;

import de.gwdg.ndn.hdldatagram.transport.ProcessorThread;
import net.named_data.jndn.Face;
import net.named_data.jndn.Name;
import net.named_data.jndn.security.SecurityException;

public class MainThread extends Thread {

  private Face transportface;
  private final Name NODENAME = new Name("/1234");
  private ProcessorThread processorthread;
  private Thread procthread;
  private DBHelper dbhelper;

  public MainThread(DBHelper dbhelper) throws Exception{
    this.dbhelper = dbhelper;

      // Start up the NDN transport interfaces
      this.transportface = TransportFace.getInstance(NODENAME).
          getFace();
      this.processorthread = new ProcessorThread(this.transportface);
      procthread = new Thread(processorthread);
      procthread.setName("NDNPIDPull - Processing Thread");
      procthread.start();

      // Register the prefix Handler
      this.registerPrefix(NODENAME, this.transportface);
  }
  public void run() {
    System.out.println("NDN PID Pull :: Thread running");
    while(true){
      try {
        Thread.sleep(30000);
      } catch (InterruptedException e) {
        e.printStackTrace();
      }
    }
  }

  private void registerPrefix(Name myNodeName, Face transportface) {
    InterestHandler interesthandler = new InterestHandler(this.
        dbhelper);
    try {
      transportface.registerPrefix(new Name(myNodeName.toUri()),
          interesthandler, interesthandler);
    } catch (IOException | SecurityException e) {
```

```
        System.out.println("NDNTransp :: Cannot register prefix "+
            myNodeName.toUri());
48        e.printStackTrace();
      }
50    }

52 }
```

Listing A.15: Addition 6 for Handle Server integration of NDN PID pull (Interest Handler)

```
package de.gwdg.ndn.hdlpull.server;

import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectOutput;
import java.io.ObjectOutputStream;

import org.apache.commons.codec.binary.Base64;

import net.handle.hdllib.HandleException;
import net.handle.hdllib.HandleValue;
import net.named_data.jndn.Data;
import net.named_data.jndn.Interest;
import net.named_data.jndn.Name;
import net.named_data.jndn.OnInterest;
import net.named_data.jndn.OnRegisterFailed;
import net.named_data.jndn.transport.Transport;
import net.named_data.jndn.util.Blob;

public class InterestHandler implements OnInterest, OnRegisterFailed
    {

  private static final Blob HANDLE_NOT_FOUND_PLD = new Blob("
      NOT_FOUND");
  private DBHelper dbhelper;

  public InterestHandler(DBHelper dbhelper) {
    this.dbhelper = dbhelper;
  }

  @Override
  public void onRegisterFailed(Name prefix) {
    System.out.println("NDNPIDPullTransp :: cannot register prefix "
        + prefix.toUri() + ". Exiting.");
    System.exit(-1);
  }

  @Override
  public void onInterest(Name prefix, Interest interest, Transport
      transport, long registeredPrefixId) {
    // System.out.println("INT :: " + interest.getName().toUri());
    Data data = new Data(interest.getName());
    HandleValue[] handleValue;
    // Resolve the Handle
    try {
      try {
        handleValue = dbhelper.resolve(prefix.toUri().substring(1) +
            "/" + getHandleSuffix(interest));
        // Handle found -> encode target Handlevalues as NDN data
            payload
```

```
            data.setContent(new Blob(serializeHandleValueToNDNBytePayload
                (handleValue)));
46        } catch (HandleException e) {
            // Handle not found -> encode "Not found" message
48          data.setContent(HANDLE_NOT_FOUND_PLD);
          }
50        // set back via transport
          Blob encodedData = data.wireEncode();
52        transport.send(encodedData.buf());
      } catch (IOException e) {
54        System.out.println("NDNPIDPullTransp :: IOException! Data
            packet data could not be encoded");
          e.printStackTrace();
56      }
    }
58
    private static final String getHandleSuffix(Interest interest) {
60      int numberOfNameParts = interest.getName().size();
        Name handleSuffix = interest.getName().getSubName(
            numberOfNameParts - 1);
62      String encodedSuffix = handleSuffix.toUri().substring(1);
        byte[] valueDecoded = Base64.decodeBase64(encodedSuffix.getBytes
            ());
64      return new String(valueDecoded);
    }
66
    private static final byte[] serializeHandleValueToNDNBytePayload(
        HandleValue[] handleValue) {
68      ByteArrayOutputStream bos = new ByteArrayOutputStream();
        ObjectOutput out = null;
70      byte[] ndnPayload = null;
        try {
72        out = new ObjectOutputStream(bos);
          out.writeObject(handleValue);
74        ndnPayload = bos.toByteArray();
      } catch (IOException e) {
76        System.out.println("NDNPIDPullTransp :: IOException! Could not
            Serialize Handle Data to NDN ByteArray.");
          e.printStackTrace();
78      } finally {
        try {
80          if (out != null)
              out.close();
82        } catch (IOException e) {
            System.out.println("NDNPIDPullTransp :: IOException! Could
                not close Bytestream for NDN ByteArray serialization");
84          e.printStackTrace();
          }
86        try {
            bos.close();
88        } catch (IOException e) {
```

```
          System.out.println("NDNPIDPullTransp :: IOException! Could
              not close ObjectOutput for NDN ByteArray serialization");
90        e.printStackTrace();
        }
92    }
      return ndnPayload;
94  }

96 }
```

Listing A.16: Addition 7 for Handle Server integration of NDN PID pull (Database Layer)

```
package de.gwdg.ndn.hdlpull.server;
2
import net.handle.hdllib.Encoder;
4 import net.handle.hdllib.HandleException;
import net.handle.hdllib.HandleStorage;
6 import net.handle.hdllib.HandleValue;
import net.handle.hdllib.Util;
8
public class DBHelper {
10  private HandleStorage storage;
    private boolean caseSensitive;
12  public DBHelper(HandleStorage storage, boolean caseSensitive) {
      this.storage = storage;
14    this.caseSensitive = caseSensitive;
    }
16
    public HandleValue[] resolve(String handle) throws HandleException
       {
18    byte[][] rawValues = this.storage.getRawHandleValues(
          caseSensitive ? Util.encodeString(handle) : Util.upperCase(
             Util.encodeString(handle)), null, null);
20    if (rawValues == null)
        return null;
22    HandleValue values[] = new HandleValue[rawValues.length];
      for (int i = 0; i < values.length; i++) {
24      values[i] = new HandleValue();
        Encoder.decodeHandleValue(rawValues[i], 0, values[i]);
26    }
      return values;
28  }
}
```

Listing A.17: Addition 8 for Handle Server integration of NDN PID pull (Transport Layer)

```
1  package de.gwdg.ndn.hdlpull.server;

3  import de.gwdg.ndn.hdldatagram.netmanager.KeyStore;
   import de.gwdg.ndn.hdldatagram.transport.ProcessorThread;
5  import net.named_data.jndn.Face;
   import net.named_data.jndn.Name;
7  import net.named_data.jndn.security.KeyChain;
   import net.named_data.jndn.security.KeyType;
9  import net.named_data.jndn.security.SecurityException;
   import net.named_data.jndn.security.identity.IdentityManager;
11 import net.named_data.jndn.security.identity.MemoryIdentityStorage;
   import net.named_data.jndn.security.identity.MemoryPrivateKeyStorage;
13 import net.named_data.jndn.security.policy.SelfVerifyPolicyManager;
   import net.named_data.jndn.util.Blob;
15
   public class TransportFace {
17
     private static TransportFace instance;
19   private Name mynodename;
     private Face face = null;
21   private Thread processfacethread;

23   private TransportFace(Name myNodeName) {
       this.mynodename = myNodeName;
25   }

27   public static synchronized TransportFace getInstance(Name
         myNodeName) {
       if (TransportFace.instance == null) {
29       TransportFace.instance = new TransportFace(myNodeName);
       }
31     return TransportFace.instance;
     }
33
     public Face getFace() {
35     // Return existing faces
       if (this.face != null)
37       return this.face;
       // No face has been created yet. Create a new one
39     this.face = new Face();
       Name keyname = new Name("/" + this.mynodename + "/pubkey");
41     Name certificateName = keyname.getSubName(0, keyname.size() - 1).
           append("KEY").append(keyname.get(-1))
             .append("ID-CERT").append("0");
43     MemoryIdentityStorage identityStorage = new MemoryIdentityStorage
           ();
       MemoryPrivateKeyStorage privateKeyStorage = new
           MemoryPrivateKeyStorage();
45     KeyChain keyChain = new KeyChain(new IdentityManager(
           identityStorage, privateKeyStorage),
```

```
                new SelfVerifyPolicyManager ( identityStorage ) );
47    keyChain . setFace ( face );
      face . setCommandSigningInfo ( keyChain , certificateName );
49    this . processfacethread = new Thread ( new ProcessorThread ( this . face
         ) );
      this . processfacethread . setName ( "NDNTransp - FaceEventProcessor" );
51    this . processfacethread . start ();
      try {
53        identityStorage . addKey ( keyname , KeyType . RSA , new Blob ( KeyStore .
            DEFAULT_RSA_PUBLIC_KEY_DER , false ));
          privateKeyStorage . setKeyPairForKeyName ( keyname , KeyType . RSA ,
            KeyStore . DEFAULT_RSA_PUBLIC_KEY_DER ,
55            KeyStore . DEFAULT_RSA_PRIVATE_KEY_DER );
      } catch ( SecurityException e ) {
57        // TODO Auto - generated catch block
          e . printStackTrace ();
59    }
      return face ;
61  }

63 }
```

A.2.5 Additions for PID Publishing Using NDN PID Pull (Client)

Following additions have been added to the Handle Client Library 8.1.1 code base, in order to let clients resolve Handle PIDs using the NDN PID push mechanism.

Listing A.18: Addition 10 for HDLLib integration of NDN PID pull (API Layer)

```
1  package de.gwdg.ndn.hdlpull.client;

3  import java.io.ByteArrayInputStream;
   import java.io.IOException;
5  import java.io.ObjectInput;
   import java.io.ObjectInputStream;
7  import java.nio.ByteBuffer;
   import java.util.Random;

   import org.apache.commons.codec.binary.Base64;

11
   import net.handle.hdllib.HandleValue;
13 import net.named_data.jndn.Data;
   import net.named_data.jndn.Face;
15 import net.named_data.jndn.Interest;
   import net.named_data.jndn.Name;
17 import net.named_data.jndn.OnData;
   import net.named_data.jndn.OnTimeout;

19
   public class PidPullClient {

21
     private Face face;
23   public HandleValue[] retval;

25   public PidPullClient() {
       Name nodename = new Name("cl" + getSaltString());
27     face = TransportFace.getInstance(nodename).getFace();
     }

29
     class DataHandler implements OnData, OnTimeout {
31     private PidPullClient pc;
       public int cbCounter = 0;

33
       public DataHandler(PidPullClient pc) {
35       this.pc = pc;
       }

37
       public void onData(Interest interest, Data data) {
39       pc.retval = null;
         ++cbCounter;
41       ByteBuffer payload = data.getContent().buf();
         pc.retval = this.unserializeNDNPayload(payload);
43     }
```

```java
45    public void onTimeout(Interest interest) {
        ++cbCounter;
47      System.out.println("Time out for interest " + interest.getName
            ().toUri());
      }
49
      private HandleValue[] unserializeNDNPayload(ByteBuffer payload) {
51      byte[] data = new byte[payload.remaining()];
        payload.get(data);
53      ByteArrayInputStream bis = new ByteArrayInputStream(data);
        ObjectInput in = null;
55      HandleValue[] handleValue = null;
        try {
57        in = new ObjectInputStream(bis);
          handleValue = (HandleValue[]) in.readObject();
59      } catch (IOException e) {
          System.out.println("NDN PID Pull Client :: Could not read NDN
              packet payload");
61        e.printStackTrace();
        } catch (ClassNotFoundException e) {
63        System.out.println("NDN PID Pull Client :: Could not
              unserialize NDN packet payload");
          e.printStackTrace();
65      } finally {
          try {
67          bis.close();
          } catch (IOException e) {
69          System.out.println("NDN PID Pull Client :: Could not close
                byte input stream for NDN packet unserialization");
            e.printStackTrace();
71        }
          try {
73          if (in != null) {
              in.close();
75          }
          } catch (IOException e) {
77          System.out.println("NDN PID Pull Client :: Could not close
                inout stream for NDN packet unserialization");
            e.printStackTrace();
79        }
        }
81      return handleValue;
      }
83  }

85  public HandleValue[] resolveHandle(String handle) {
      try {
87      DataHandler datahdl = new DataHandler(this);
        Name dataname = getDataNameForHandle(handle);
89      face.expressInterest(dataname, datahdl, datahdl);
```

```java
 91       while (datahdl.cbCounter < 3) {
           face.processEvents();
 93        Thread.sleep(5);
           if (this.retval != null && this.retval.length > 0)
 95          break;
         }
 97     } catch (Exception e) {
         // System.out.println("exception: " + e.getMessage());
 99      System.out.println("NDN PID Pull Client :: Resolving problem at
             : " + handle);
         e.printStackTrace();
101     }
       return this.retval;
103   }

105   private static final Name getDataNameForHandle(String handle) {
       String[] handleSplit = handle.split("/"); // prefix is on 0,
           suffix is
107                               // on 1
       // Encode the suffix part on pos 1
109      byte[] bytesEncoded = Base64.encodeBase64(handleSplit[1].getBytes
           ());
       // create the name: suffix / enc(prefix)
111     return new Name(handleSplit[0] + "/" + new String(bytesEncoded));
     }

113
     private static final String getSaltString() {
115     String SALTCHARS = "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890";
       StringBuilder salt = new StringBuilder();
117     Random rnd = new Random();
       while (salt.length() < 18) {
119      int index = (int) (rnd.nextFloat() * SALTCHARS.length());
         salt.append(SALTCHARS.charAt(index));
121     }
       String saltStr = salt.toString();
123     return saltStr;

125   }
}
```

Listing A.19: Addition 11 for HDLLib integration of NDN PID pull (Transport Layer)

```
package de.gwdg.ndn.hdlpull.client;

import de.gwdg.ndn.hdldatagram.netmanager.KeyStore;
import de.gwdg.ndn.hdldatagram.transport.ProcessorThread;
import net.named_data.jndn.Face;
import net.named_data.jndn.Name;
import net.named_data.jndn.security.KeyChain;
import net.named_data.jndn.security.KeyType;
import net.named_data.jndn.security.SecurityException;
import net.named_data.jndn.security.identity.IdentityManager;
import net.named_data.jndn.security.identity.MemoryIdentityStorage;
import net.named_data.jndn.security.identity.MemoryPrivateKeyStorage;
import net.named_data.jndn.security.policy.SelfVerifyPolicyManager;
import net.named_data.jndn.util.Blob;

public class TransportFace {

  private static TransportFace instance;
  private Name mynodename;
  private Face face = null;
  private Thread processfacethread;

  private TransportFace(Name myNodeName) {
    this.mynodename = myNodeName;
  }

  public static synchronized TransportFace getInstance(Name
    myNodeName) {
    if (TransportFace.instance == null) {
      TransportFace.instance = new TransportFace(myNodeName);
    }
    return TransportFace.instance;
  }

  public Face getFace() {
    // Return existing faces
    if(this.face != null)
      return this.face;
    // No face has been created yet. Create a new one
    this.face = new Face();
    Name keyname = new Name("/" + this.mynodename + "/pubkey");
    Name certificateName = keyname.getSubName(0, keyname.size() - 1).
      append("KEY").append(keyname.get(-1))
        .append("ID-CERT").append("0");
    MemoryIdentityStorage identityStorage = new MemoryIdentityStorage
      ();
    MemoryPrivateKeyStorage privateKeyStorage = new
      MemoryPrivateKeyStorage();
    KeyChain keyChain = new KeyChain(new IdentityManager(
      identityStorage, privateKeyStorage),
```

```
46          new SelfVerifyPolicyManager(identityStorage));
      keyChain.setFace(face);
48    face.setCommandSigningInfo(keyChain, certificateName);
      this.processfacethread = new Thread(new ProcessorThread(this.face
          ));
50    this.processfacethread.setName("NDNTransp - FaceEventProcessor");
      this.processfacethread.start();
52    try {
        identityStorage.addKey(keyname, KeyType.RSA, new Blob(KeyStore.
            DEFAULT_RSA_PUBLIC_KEY_DER, false));
54      privateKeyStorage.setKeyPairForKeyName(keyname, KeyType.RSA,
            KeyStore.DEFAULT_RSA_PUBLIC_KEY_DER,
             KeyStore.DEFAULT_RSA_PRIVATE_KEY_DER);
56    } catch (SecurityException e) {
        // TODO Auto-generated catch block
58      e.printStackTrace();
      }
60    return face;
    }
62  }
```

A.3 Simulation Environment

For the evaluation scenarios used in this thesis, we provide the scripts for data collection in the Handle ecosystem, the Mini-NDN simulator control scripts and the source codes for generating the evaluation scenarios. Furthermore, we include the source code of reference implementations we use for comparing unmodified Handle protocols against the NDN-enabled implementations using NDN PID push and pull communication modes.

A.3.1 PID Resolution Request Classification By Handle Prefixes

Listing A.20: Classification of PID Resolution Requets according to their Handle Prefixes from Anonymized Log Data

```python
__author__ = 'owannen'
import glob
import os
import collections

class PrefixClassification:

    def __init__(self):
        pass

    def getNumberOfInputFiles(self):
        return len(self.__getFilePathNamesInputdir())

    def __getFilePathNamesInputdir(self):
        return glob.glob(self.__getRootPath() + "../workdata/*.txt")

    def __getRootPath(self):
        return str(os.path.dirname(os.path.realpath(__file__)) + "/")

    def getTopXnumbers(self, number_of_ranks):
        prefixes = []
        return_val = []
        hitcounter = 0
        # Iterate over all files
        for logfile in self.__getFilePathNamesInputdir():
            input_filepointer = open(logfile, "r")
            # get all prefixes in the file
            print "Processing :: " + str(logfile)
            for currentline in input_filepointer:
                prefix = currentline.split(",")[1].rstrip().lstrip().
                    lower()
                prefixes.append(prefix)
                hitcounter += 1
            input_filepointer.close()
        print ""
```

```
35          print "Total number of hits %s" % hitcounter
            print ""
37          # Calculate the sums
            for prefix_str , count in collections.Counter(prefixes).
                most_common(number_of_ranks):
39              percentage = float(count) / float(float(hitcounter) /
                    100.0)
                return_val.append((prefix_str , count , percentage))
41          return return_val

43      def run_analysis(self , number_of_ranks):
            return self.getTopXnumbers(number_of_ranks)
```

A.3.2 Collecting Primary Handle Site Data

Listing A.21: Collection of IP-addresses from Primary Handle Sites

```
import json
import urllib2

# Collection of top-200 Handle prefixes
prefixes = [...]

# Get the IP of the Primary Handle LHS via the public REST API
def resolve(prefix):
    return [site['data']['value']['servers'][0]['address'] for site in
        [x for x in json.load(urllib2.urlopen('http://hdl.handle.net/
        api/handles/' + prefix))['values'] if x['type'] == 'HS_SITE']
        if site['data']['value']['primarySite'] == True]

ips = [resolve(prefix) for prefix in prefixes]

# Print the result:
print '\n\n=== RESULTS =='
index_prefixes = 0
for ip in ips:
    print prefixes[index_prefixes] + "," + str(ip)
    index_prefixes = index_prefixes + 1
```

Listing A.22: Calculation of Hop Counts between a fixed server and Primary Site of a Handle LHS

```
from scapy.all import *

results = {}

# IP-Addresses pf all primary Handle LHS Sites matched with Handle
    prefix

sites = {...}

# Perform a traceroute on all IP-address using UDP
def count_hops(ip_address):
  target = [ip_address]
  result, unans = traceroute(target,l4=UDP(sport=RandShort())/DNS(qd=
      DNSQR(qname="www.google.com")))
  return len(result)


# Process all IP-addresses of available Primary Handle Sites
for key, value in sites.iteritems():
  results[key] = count_hops(value)

# Print the result:
print '\n\n=== RESULTS =='
for key, value in results.iteritems():
  print str(key) + ',' + str(value)
```

A.3.3 Network Hop Calculations For Classified Handle Prefixes

Rank	Prefix	Hits	Percent	Hops
1	Prefix-A	3,848,567	16.911201%	10
2	Prefix-B	1,882,874	8.273641%	12
3	Prefix-C	1,571,502	6.905424%	0 *corr. 14*
4	Prefix-D	985,922	4.332294%	10
5	Prefix-E	862,209	3.788680%	13
6	Prefix-F	855,999	3.761392%	9
7	Prefix-H	344,398	1.513338%	15
8	Prefix-I	301,685	1.325651%	21
9	Prefix-J	301,667	1.325572%	12
10	Prefix-K	264,584	1.162623%	21
11	Prefix-L	256,810	1.128463%	16
12	Prefix-M	232,642	1.022265%	27

Rank	Prefix	Hits	Percent	Hops
13	Prefix-N	230,000	1.010656%	12
14	Prefix-O	186,273	0.818512%	12
15	Prefix-P	185,579	0.815463%	15
16	Prefix-Q	153,435	0.674217%	8
17	Prefix-R	137,262	0.603151%	13
18	Prefix-S	134,799	0.592328%	13
19	Prefix-T	129,922	0.570897%	12
20	Prefix-U	124,152	0.545543%	12
21	Prefix-V	119,870	0.526727%	12
22	Prefix-W	115,798	0.508834%	15
23	Prefix-X	114,083	0.501298%	18
24	Prefix-Y	106,764	0.469138%	11
25	Prefix-Z	106,684	0.468786%	11
26	Prefix-AA	92,701	0.407343%	13
27	Prefix-AB	92,415	0.406086%	8
28	Prefix-AC	92,181	0.405058%	16
29	Prefix-AD	87,837	0.385969%	16
30	Prefix-AE	86,572	0.380411%	12
31	Prefix-AF	85,184	0.374312%	13
32	Prefix-AG	82,061	0.360589%	10
33	Prefix-AH	80,751	0.354832%	11
34	Prefix-AI	79,933	0.351238%	16
35	Prefix-AJ	79,808	0.350689%	21
36	Prefix-AK	79,171	0.347890%	12
37	Prefix-AL	74,666	0.328094%	19
38	Prefix-AM	74,230	0.326178%	12
39	Prefix-AN	73,519	0.323054%	17
40	Prefix-AO	72,679	0.319363%	14
41	Prefix-AP	70,940	0.311721%	15
42	Prefix-AQ	66,407	0.291803%	10
43	Prefix-AR	63,308	0.278185%	12
44	Prefix-AS	60,358	0.265222%	22
45	Prefix-AT	59,552	0.261681%	14
46	Prefix-AU	59,197	0.260121%	22
47	Prefix-AV	58,105	0.255322%	11
48	Prefix-AW	57,800	0.253982%	9
49	Prefix-AX	56,250	0.247171%	16
50	Prefix-AY	54,126	0.237838%	13
51	Prefix-AZ	53,675	0.235856%	24
52	Prefix-BA	53,386	0.234586%	11

Rank	Prefix	Hits	Percent	Hops
53	Prefix-BB	53,078	0.233233%	10
54	Prefix-BC	51,273	0.225302%	12
55	Prefix-BD	50,357	0.221276%	11
56	Prefix-BE	50,258	0.220841%	13
57	Prefix-BF	50,251	0.220811%	11
58	Prefix-BG	49,554	0.217748%	20
59	Prefix-BH	49,231	0.216329%	14
60	Prefix-BI	48,983	0.215239%	10
61	Prefix-BJ	48,420	0.212765%	10
62	Prefix-BK	48,132	0.211499%	18
63	Prefix-BL	47,797	0.210027%	13
64	Prefix-BM	47,517	0.208797%	13
65	Prefix-BN	46,960	0.206350%	13
66	Prefix-BO	46,146	0.202773%	13
67	Prefix-BP	44,923	0.197399%	14
68	Prefix-BQ	44,815	0.196924%	23
69	Prefix-BR	44,542	0.195724%	11
70	Prefix-BS	44,038	0.193510%	11
71	Prefix-BT	43,901	0.192908%	12
72	Prefix-BU	42,958	0.188764%	22
73	Prefix-BV	42,924	0.188615%	15
74	Prefix-BW	41,781	0.183592%	13
75	Prefix-BX	41,431	0.182054%	13
76	Prefix-BY	41,153	0.180833%	11
77	Prefix-BZ	40,151	0.176430%	22
78	Prefix-CA	39,650	0.174228%	14
79	Prefix-CB	39,221	0.172343%	17
80	Prefix-CC	38,068	0.167277%	19
81	Prefix-CD	37,986	0.166916%	13
82	Prefix-CE	37,892	0.166503%	10
83	Prefix-CF	37,590	0.165176%	10
84	Prefix-CG	36,831	0.161841%	16
85	Prefix-CH	36,384	0.159877%	11
86	Prefix-CI	35,816	0.157381%	13
87	Prefix-CJ	35,459	0.155812%	11
88	Prefix-CK	35,119	0.154318%	19
89	Prefix-CL	34,159	0.150100%	14
90	Prefix-CM	34,112	0.149893%	14
91	Prefix-CN	33,318	0.146404%	12
92	Prefix-CO	33,266	0.146176%	21

Rank	Prefix	Hits	Percent	Hops
93	Prefix-CP	32,956	0.144814%	18
94	Prefix-CQ	32,341	0.142111%	13
95	Prefix-CR	32,304	0.141949%	11
96	Prefix-CS	32,014	0.140674%	13
97	Prefix-CT	31,707	0.139325%	28
98	Prefix-CU	31,533	0.138561%	8
99	Prefix-CV	31,371	0.137849%	13
100	Prefix-CW	30,978	0.136122%	18
101	Prefix-CX	30,825	0.135450%	15
102	Prefix-CY	30,273	0.133024%	23
103	Prefix-CZ	29,190	0.128265%	13
104	Prefix-DA	29,005	0.127452%	11
105	Prefix-DB	28,429	0.124921%	11
106	Prefix-DC	28,385	0.124728%	10
107	Prefix-DD	27,700	0.121718%	17
108	Prefix-DE	27,588	0.121226%	14
109	Prefix-DF	27,335	0.120114%	15
110	Prefix-DG	27,194	0.119495%	13
111	Prefix-DH	27,075	0.118972%	13
112	Prefix-DI	27,031	0.118778%	16
113	Prefix-DJ	27,027	0.118761%	20
114	Prefix-DK	26,811	0.117812%	24
115	Prefix-DL	26,795	0.117741%	10
116	Prefix-DM	26,636	0.117043%	14
117	Prefix-DN	26,437	0.116168%	16
118	Prefix-DO	26,205	0.115149%	9
119	Prefix-DP	26,092	0.114652%	13
120	Prefix-DQ	26,023	0.114349%	15
121	Prefix-DR	25,819	0.113453%	9
122	Prefix-DS	25,819	0.113453%	9
123	Prefix-DT	25,633	0.112635%	13
124	Prefix-DU	25,492	0.112016%	14
125	Prefix-DV	25,390	0.111568%	12
126	Prefix-DW	25,344	0.111365%	15
127	Prefix-DX	25,280	0.111084%	17
128	Prefix-DY	25,128	0.110416%	20
129	Prefix-DZ	24,537	0.107819%	14
130	Prefix-EA	24,478	0.107560%	14
131	Prefix-EB	23,890	0.104976%	14
132	Prefix-EC	23,823	0.104682%	13

Rank	Prefix	Hits	Percent	Hops
133	Prefix-ED	23,692	0.104106%	14
134	Prefix-EE	23,427	0.102942%	10
135	Prefix-EF	23,359	0.102643%	21
136	Prefix-EG	23,233	0.102089%	22
137	Prefix-EH	23,130	0.101637%	7
138	Prefix-EI	23,026	0.101180%	14
139	Prefix-EJ	22,820	0.100275%	7
140	Prefix-EK	22,643	0.099497%	16
141	Prefix-EL	22,611	0.099356%	10
142	Prefix-EM	22,458	0.098684%	11
143	Prefix-EN	21,484	0.094404%	12
144	Prefix-EO	21,256	0.093402%	13
145	Prefix-EP	21,140	0.092892%	16
146	Prefix-EQ	21,055	0.092519%	22
147	Prefix-ER	21,053	0.092510%	11
148	Prefix-ES	21,030	0.092409%	21
149	Prefix-ET	20,910	0.091882%	14
150	Prefix-EU	20,592	0.090484%	15
151	Prefix-EV	20,306	0.089228%	11
152	Prefix-EW	20,305	0.089223%	16
153	Prefix-EX	20,137	0.088485%	11
154	Prefix-EY	20,076	0.088217%	8
155	Prefix-EZ	20,061	0.088151%	11
156	Prefix-FA	19,967	0.087738%	10
157	Prefix-FB	19,831	0.087140%	15
158	Prefix-FC	19,625	0.086235%	12
159	Prefix-FD	19,550	0.085906%	21
160	Prefix-FE	19,493	0.085655%	14
161	Prefix-FF	19,431	0.085383%	16
162	Prefix-FG	19,425	0.085356%	13
163	Prefix-FH	19,421	0.085339%	18
164	Prefix-FI	19,367	0.085102%	12
165	Prefix-FJ	19,367	0.085102%	14
166	Prefix-FK	19,124	0.084034%	14
167	Prefix-FL	19,120	0.084016%	12
168	Prefix-FM	19,038	0.083656%	15
169	Prefix-FN	18,754	0.082408%	23
170	Prefix-FO	18,692	0.082136%	18
171	Prefix-FP	18,537	0.081454%	24
172	Prefix-FQ	18,471	0.081164%	13

Rank	Prefix	Hits	Percent	Hops
173	Prefix-FR	18,317	0.080488%	9
174	Prefix-FS	18,019	0.079178%	16
175	Prefix-FT	17,987	0.079038%	20
176	Prefix-FU	17,566	0.077188%	10
177	Prefix-FV	17,462	0.076731%	13
178	Prefix-FW	17,455	0.076700%	17
179	Prefix-FX	17,329	0.076146%	13
180	Prefix-FY	16,972	0.074578%	16
181	Prefix-FZ	16,895	0.074239%	14
182	Prefix-GA	16,845	0.074020%	15
183	Prefix-GB	16,677	0.073281%	14
184	Prefix-GC	16,668	0.073242%	13
185	Prefix-GD	16,603	0.072956%	14
186	Prefix-GE	16,527	0.072622%	15
187	Prefix-GF	16,321	0.071717%	14
188	Prefix-GG	16,009	0.070346%	16
189	Prefix-GH	15,995	0.070285%	10
190	Prefix-GI	15,974	0.070192%	13
191	Prefix-GJ	15,857	0.069678%	16
192	Prefix-GK	15,744	0.069182%	25
193	Prefix-GL	15,682	0.068909%	13
194	Prefix-GM	15,566	0.068399%	14

Table A.1: Network Hop Calculations For Classified Handle Prefixes

Average	14.28
Variance (population)	16.06
Standard deviation (population)	4.01
Variance (sample)	16.14
Standard deviation (sample)	4.02

Table A.2: Summary of Network Hops

A.3.4 NDN PID Push Evaluation Testbed with Mini-NDN

Listing A.23: Mini-NDN Experiment NDN Environment Setup Program

```python
#!/usr/bin/python

from ndn.experiments.experiment import Experiment
import time
import sys
import os

class CurrentExperiment(Experiment):
  def __init__(self, args):
    self.hosts = {}
    Experiment.__init__(self, args)

  def setup(self):
    self.checkNetworkSanity()

  def run(self):
    self.setup_host_name_map()
    self.__runExperiment()

  def checkNetworkSanity(self):
    waittime = 10; # 25 Sekunden is minimal waiting time. If you
    print "Waiting " + str(waittime) + " seconds for sanity..."
    time.sleep(waittime)
    for host in self.net.hosts:
      host.cmd('ndnpingserver /ndn/de/gwdg/' + host.name + ' &')
      statusPing = host.cmd("nfd-status -b | grep /ndn/de/gwdg/" +
          host.name)
      host.cmd("nfd-status > nfd-status.txt")
      if "/ndn/de/gwdg/" + host.name not in statusPing:
        print " failed. Host %s is not sane!" % (host.name)
        sys.exit(1)
    print " done!"

  def setup_host_name_map(self):
    for host in self.net.hosts:
      self.hosts[str(host.name)] = host

  # Register name for the given host id
  # node_name = Name of the node you want to contact for announcement
  # name_to_announce = What do you have to offer
  # ip_of_host_running_service = what ip address does the offering
      node have
  def __setup_ndn_route(self, node_name, name_to_announce,
      ip_of_host_running_service):
    result = str(self.hosts[node_name].cmd('nfdc register ' +
        name_to_announce + ' udp4://' + ip_of_host_running_service))
    print '==== Routed added ==='
```

```
       print 'Node: %s Service: %s  Service-IP: %s' % (node_name,
          name_to_announce, ip_of_host_running_service)
45     print result
       if len(result) < 5:
47       print 'Route adding failed.'
         #sys.exit(1)
49     else:
         print result
51     print '===================='
       print ' '
53
   def __forward_announcement(self, name_to_announce):
55     ipcounter = 1
       i = 0
57     while i < len(self.net.hosts):
         if i + 1 == len(self.net.hosts):
59         return
         self.__setup_ndn_route(self.net.hosts[i + 1].name,
            name_to_announce, '1.0.0.' + str(ipcounter))
61       ipcounter += 4
         i += 1
63

65   def __backward_announcement(self, name_to_announce):
       highest_octet = int(str(self.net.hosts[-1].IP()).split('.')[-1])
67     i = len(self.net.hosts) - 1
       while i > 0:
69       self.__setup_ndn_route(self.net.hosts[ i - 1 ].name,
            name_to_announce, '1.0.0.' + str(highest_octet))
         highest_octet -= 4
71       i -= 1

73
   def __runExperiment(self):
75     print "Running the experiment ..."
       # Copy in the binaries
77     os.system('cp /home/oliver/workspace/ndn-handle/
          benchmark_environment/binaries/*.* /tmp/client')
       os.system('cp /home/oliver/workspace/ndn-handle/
          benchmark_environment/binaries/*.* /tmp/server')
79
       # Copy the Handle Server Config
81     os.system('cp /home/oliver/workspace/ndn-handle/mini_net/
          simulation_data/ndn_hdl-server-config_empty.tar.gz /tmp/server
          ')
       os.system('cd /tmp/server && tar xzf ndn_hdl-server-config_empty.
          tar.gz')
83
       # Install a prefilled database if required by the benchmark
85     if os.path.isfile('/tmp/prefill.bin'):
```

```
         os.system('cp /home/oliver/workspace/ndn-handle/mini_net/
            simulation_data/100
            k_PID_preloaded_berkeydb_with_target_urls_from_11858.tar.gz
             /tmp/server')
87       os.system('cd /tmp/server && tar xzf 100
            k_PID_preloaded_berkeydb_with_target_urls_from_11858.tar.gz
            ')
         os.system('rm -rf /tmp/server/hdl-server-config/bdbje && mv /
            tmp/server/bdbje /tmp/server/hdl-server-config/bdbje')
89       print "== Prefilled Handle Database with 100k PIDs installed ==
            "

91     # Start the Handle Server
       self.hosts["server"].cmd('cd /tmp/server && java -jar /tmp/server
          /pid_push_benchmarks_handle_server_git_cc42515.jar /tmp/server
          /hdl-server-config > /tmp/server/log/hdl-server.log 2>&1 &')
93
       # Setup routes according to node selections
95     # forward announcement
       self.__forward_announcement('/ndn/de/gwdg/client/ping')
97     self.__forward_announcement('/hdlclient')
       self.__forward_announcement('/hdlclient/netmgr')
99
       # Backward announcement
101    self.__backward_announcement('/ndn/de/gwdg/server/ping')
       self.__backward_announcement('/1234')
103    self.__backward_announcement('/1234/netmgr')

105    time.sleep(2)
       print "Start the interactive shell to execute experiments"
107
Experiment.register("current-experiment", CurrentExperiment)
```

Listing A.24: Mini-NDN NDN Sample Topology Configuration File Connecting Client to
Server with one Intermediate Node

```
[nodes]
client: _ hyperbolic-state=off radius=0.0 angle=0.0 network=/ndn
    router=/%C1.Router/cs/host site=/edu/site nlsr-log-level=INFO max-
    faces-per-prefix=0 nfd-log-level=INFO
inter1: _ hyperbolic-state=off radius=0.0 angle=0.0 network=/ndn
    router=/%C1.Router/cs/host site=/edu/site nlsr-log-level=INFO max-
    faces-per-prefix=0 nfd-log-level=INFO
server: _ hyperbolic-state=off radius=0.0 angle=0.0 network=/ndn
    router=/%C1.Router/cs/host site=/edu/site nlsr-log-level=INFO max-
    faces-per-prefix=0 nfd-log-level=INFO
[switches]
[links]
client:inter1 bw=1000
inter1:server bw=1000
```

Listing A.25: Mini-NDN NDN Experiment Launcher

```
#!/bin/bash
if [ "$EUID" -ne 0 ]
  then echo "Please run as root"
  exit
fi

echo "How many intermediate nodes should be deployed?"
PS3='Please enter your choice: '
options=("One Intermediate Node" "Two Intermediate Nodes" "Three
    Intermediate Nodes" "Four Intermediate Nodes" "Five Intermediate
    Nodes" "Six Intermediate Nodes" "Seven Intermediate Nodes" "Eight
    Intermediate Nodes" "Nine Intermediate Nodes" "Ten Intermediate
    Nodes" "Eleven Intermediate Nodes" "Twelve Intermediate Nodes" "
    Thirteen Intermediate Nodes" "Direct Connect" "Quit")
topology_file=''
select opt in "${options[@]}"
do
    case $opt in
        "One Intermediate Node")
            echo "Running one intermediate node"
            topology_file='one_node.conf'
            break
            ;;
        "Two Intermediate Nodes")
            echo "Running two intermediate nodes"
            topology_file='two_nodes.conf'
            break
            ;;
        "Three Intermediate Nodes")
            echo "Running three intermediate nodes"
            topology_file='three_nodes.conf'
            break
```

```
28              ;;
        "Four Intermediate Nodes")
30              echo "Running four intermediate nodes"
                topology_file='four_nodes.conf'
32              break
                ;;
34          "Five Intermediate Nodes")
                echo "Running five intermediate nodes"
36              topology_file='five_nodes.conf'
                break
38              ;;
            "Six Intermediate Nodes")
40              echo "Running six intermediate nodes"
                topology_file='six_nodes.conf'
42              break
                ;;
44          "Seven Intermediate Nodes")
                echo "Running seven intermediate nodes"
46              topology_file='seven_nodes.conf'
                break
48              ;;
            "Eight Intermediate Nodes")
50              echo "Running eight intermediate nodes"
                topology_file='eight_nodes.conf'
52              break
                ;;
54          "Nine Intermediate Nodes")
                echo "Running nine intermediate nodes"
56              topology_file='nine_nodes.conf'
                break
58              ;;
            "Ten Intermediate Nodes")
60              echo "Running ten intermediate nodes"
                topology_file='ten_nodes.conf'
62              break
                ;;
64          "Eleven Intermediate Nodes")
                echo "Running eleven intermediate nodes"
66              topology_file='eleven_nodes.conf'
                break
68              ;;
            "Twelve Intermediate Nodes")
70              echo "Running twelve intermediate nodes"
                topology_file='twelve_nodes.conf'
72              break
                ;;
74          "Thirteen Intermediate Nodes")
                echo "Running thirteen intermediate nodes"
76              topology_file='thirteen_nodes.conf'
                break
78              ;;
```

```
                "Direct Connect")
80                    echo "Running direct connect"
                      topology_file='direct_connect.conf'
82                    break
                      ;;
84            "Quit")
                      exit -1;
86                    ;;
              *) echo invalid option;;
88       esac
   done
90
   export LC_ALL="en_US.UTF-8"
92 SCRIPT=`realpath $0`
   SCRIPTPATH=`dirname $SCRIPT`
94 rm /usr/local/lib/python2.7/dist-packages/Mini_NDN-0.1.1-py2.7.egg/
       ndn/experiments/current_experiment.py > /dev/null 2>&1
   rm /usr/local/lib/python2.7/dist-packages/Mini_NDN-0.1.1-py2.7.egg/
       ndn/experiments/current_experiment.pyc > /dev/null 2>&1
96 ln -s $SCRIPTPATH/current_experiment.py /usr/local/lib/python2.7/dist
       -packages/Mini_NDN-0.1.1-py2.7.egg/ndn/experiments/
       current_experiment.py > /dev/null 2>&1
   minindn --experiment current-experiment $SCRIPTPATH/$topology_file
98 rm /usr/local/lib/python2.7/dist-packages/Mini_NDN-0.1.1-py2.7.egg/
       ndn/experiments/current_experiment.py > /dev/null 2>&1
   rm /usr/local/lib/python2.7/dist-packages/Mini_NDN-0.1.1-py2.7.egg/
       ndn/experiments/current_experiment.pyc > /dev/null 2>&1
100
   # Clean up
102 mn -c
   rm -rf /tmp/client
104 rm -rf /tmp/server
   rm -rf /tmp/inter*
```

A.3.5 TCP Evaluation Reference Testbed with Mini-NDN

Listing A.26: Mini-NDN Experiment TCP Environment Setup Program

```python
#!/usr/bin/python

from ndn.experiments.experiment import Experiment
import time
import sys
import os

class CurrentExperiment(Experiment):
  def __init__(self, args):
    self.hosts = {}
    Experiment.__init__(self, args)

  def setup(self):
    self.checkNetworkSanity()

  def run(self):
    self.setup_host_name_map()
    self.__runExperiment()

  def checkNetworkSanity(self):
    waittime = 10; # 25 Sekunden is minimal waiting time. If you
    print "Waiting " + str(waittime) + " seconds for stabilising"
    time.sleep(waittime)
    print " done!"

  def setup_host_name_map(self):
    for host in self.net.hosts:
      self.hosts[str(host.name)] = host

  def __setup_forwarder(self, host, target_ip):
    hostname = host.name
    host.cmd('cd /tmp/' + hostname +' && java -Djava.net.
        preferIPv4Stack=true -jar /tmp/' + hostname + '/
        tcp_userland_forwarder_81a36205.jar 2641 ' + target_ip + '
        2641 2>&1 > /dev/null &')
    print "Added TCP forwarder for " + hostname + " to IP: " +
        target_ip

  def __forward_announcement(self):
    highest_octet = int(str(self.net.hosts[-1].IP()).split('.')[-1])
    i = len(self.net.hosts) - 1
    while i > 0:
      self.__setup_forwarder(self.net.hosts[ i - 1 ], '1.0.0.' + str(
          highest_octet))
      highest_octet -= 4
      i -= 1

  def __runExperiment(self):
```

```
        print "Running the experiment ..."
45      # Copy in the binaries
        for host in self.net.hosts:
47        os.system('cp /home/oliver/workspace/ndn-handle/
              benchmark_environment/binaries/*.* /tmp/' + host.name)

49      # Copy the Handle Server Config
        os.system('cp /home/oliver/workspace/ndn-handle/mini_net/
            simulation_data/ndn_hdl-server-config_empty.tar.gz /tmp/server
            ')
51      os.system('cd /tmp/server && tar xzf ndn_hdl-server-config_empty.
            tar.gz')

53      # Patch the Handle Server Config to use TCP instead of NDN
        os.system('cat /home/oliver/workspace/ndn-handle/mini_net/
            simulation_data/tcp_config.dct > /tmp/server/hdl-server-config
            /config.dct')
55      print "== Handle Server configured for TCP only support =="

57      # Install a prefilled database if required by the benchmark
        if os.path.isfile('/tmp/prefill.bin'):
59        os.system('cp /home/oliver/workspace/ndn-handle/mini_net/
              simulation_data/100
              k_PID_preloaded_berkeydb_with_target_urls_from_11858.tar.gz
               /tmp/server')
          os.system('cd /tmp/server && tar xzf 100
              k_PID_preloaded_berkeydb_with_target_urls_from_11858.tar.gz
              ')
61        os.system('rm -rf /tmp/server/hdl-server-config/bdbje && mv /
              tmp/server/bdbje /tmp/server/hdl-server-config/bdbje')
          print "== Prefilled Handle Database with 100k PIDs installed ==
              "
63
        # Start the Handle Server
65      self.hosts["server"].cmd('cd /tmp/server && java -Djava.net.
            preferIPv4Stack=true -jar /tmp/server/
            pid_push_benchmarks_handle_server_git_cc42515.jar /tmp/server/
            hdl-server-config > /tmp/server/log/hdl-server.log 2>&1 &')

67      # Setup routes according to node selection
        self.__forward_announcement()
69
        time.sleep(2)
71      print "Start the interactive shell to execute experiments"

73 Experiment.register("current-experiment", CurrentExperiment)
```

A.3.6 TCP User Space Forwarder

The source codes in this section A.3.6 is realizing the TCP user space forwarder and has been implemented in Java by S. Nakov [174] [175] and was extended by the author of the thesis to support the Mini-NDN simulation environment. It is used as a reference benchmark implementation for a location-based network.

Listing A.27: TCP User Space Forwarder Server Part

```java
package de.gwdg.usforwarder;

import java.io.*;
import java.net.*;

/**
 *
 * TCPForwardServer is a simple TCP bridging software that allows
 * a TCP port on some host to be transparently forwarded to some
 * other TCP port on some other host. TCPForwardServer continuously
 * accepts client connections on the listening TCP port
 * (source port) and starts a thread (ClientThread) that
 * connects to the destination host and starts forwarding
 * the data between the client socket and destination socket.
 */

public class TCPForwardServer {
  public static int SOURCE_PORT = 0;
  public static String DESTINATION_HOST = "127.0.0.1";
  public static int DESTINATION_PORT = 0;
  private static ServerSocket serverSocket;

  public static void main(String[] args) throws IOException {
    if (args == null || args.length < 3) {
      System.err.println("usage: SOURCE_PORT DESTINATION_HOST
          DESTINATION_PORT");
      return;
    }
    SOURCE_PORT = Integer.parseInt(args[0]);
    DESTINATION_HOST = args[1];
    DESTINATION_PORT = Integer.parseInt(args[2]);

    serverSocket = new ServerSocket(SOURCE_PORT);
    while (true) {
      Socket clientSocket = serverSocket.accept();
      ClientThread clientThread = new ClientThread(clientSocket);
      clientThread.start();
    }
  }
}
```

Listing A.28: TCP User Space Forwarder Forward Thread

```java
package de.gwdg.usforwarder;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;

class ForwardThread extends Thread {

  private static final int BUFFER_SIZE = 8192;
  InputStream mInputStream;
  OutputStream mOutputStream;
  ClientThread mParent;

  /**
   * Creates a new traffic redirection thread specifying
   * its parent, input stream and output stream.
   */
  public ForwardThread(ClientThread aParent, InputStream
  aInputStream, OutputStream aOutputStream) {
    mParent = aParent;
    mInputStream = aInputStream;
    mOutputStream = aOutputStream;
  }

  /**
   * Runs the thread. Continuously reads the input stream and
   * writes the read data to the output stream. If reading or
   * writing fail, exits the thread and notifies the parent
   * about the failure.
   */
  public void run() {
    byte[] buffer = new byte[BUFFER_SIZE];
    try {
      while (true) {
        int bytesRead = mInputStream.read(buffer);
        if (bytesRead == -1)
          break; // End of stream is reached --> exit
        mOutputStream.write(buffer, 0, bytesRead);
        mOutputStream.flush();
      }
    } catch (IOException e) {
      // Read/write failed --> connection is broken
    }
    // Notify parent thread that the connection is broken
    mParent.connectionBroken();
  }
}
```

Listing A.29: TCP User Space Forwarder Client Thread

```
1  package de.gwdg.usforwarder;

3  import java.io.IOException;
   import java.io.InputStream;
5  import java.io.OutputStream;
   import java.net.Socket;

7
   /**
9   *
    * ClientThread is responsible for starting forwarding between
11   * the client and the server. It keeps track of the client
    * and servers sockets that are both closed on input/output
13   * error during the forwarding. The forwarding is bidirectional
    * and is performed by two ForwardThread instances.
15   *
    */
17  class ClientThread extends Thread {
     private Socket mClientSocket;
19    private Socket mServerSocket;
     private boolean mForwardingActive = false;
21    public ClientThread(Socket aClientSocket) {
       mClientSocket = aClientSocket;
23    }

25    /**
      * Establishes connection to the destination server and starts
27      * bidirectional forwarding ot data between the client
      * and the server.
29      */
     public void run() {
31      InputStream clientIn;
       OutputStream clientOut;
33      InputStream serverIn;
       OutputStream serverOut;
35      try {
         // Connect to the destination server
37        mServerSocket = new Socket(TCPForwardServer.DESTINATION_HOST,
           TCPForwardServer.DESTINATION_PORT);
         // Turn on keep-alive for both the sockets
39        mServerSocket.setKeepAlive(true);
         mClientSocket.setKeepAlive(true);
41        // Obtain client & server input & output streams
         clientIn = mClientSocket.getInputStream();
43        clientOut = mClientSocket.getOutputStream();
         serverIn = mServerSocket.getInputStream();
45        serverOut = mServerSocket.getOutputStream();
       } catch (IOException ioe) {
47        System.err.println("Can not connect to " + TCPForwardServer.
           DESTINATION_HOST + ":"
             + TCPForwardServer.DESTINATION_PORT);
```

```
49        connectionBroken ();
          return ;
51      }
        // Start forwarding data between server and client
53      mForwardingActive = true;
        ForwardThread clientForward = new ForwardThread(this, clientIn,
            serverOut);
55      clientForward.start ();
        ForwardThread serverForward = new ForwardThread(this, serverIn,
            clientOut);
57      serverForward.start ();
        System.out.println("TCP Forwarding " + mClientSocket.
            getInetAddress().getHostAddress() + ":"
59       + mClientSocket.getPort() + " <--> " + mServerSocket.
             getInetAddress().getHostAddress() + ":"
         + mServerSocket.getPort() + " started.");
61    }

63    /**
       * Called by some of the forwarding threads to indicate
65      * that its socket connection is brokean and both client
       * and server sockets should be closed. Closing the client
67      * and server sockets causes all threads blocked on reading
       * or writing to these sockets to get an exception and to finish
69      * their execution.
       */
71
      public synchronized void connectionBroken () {
73      try {
          mServerSocket.close ();
75      } catch (Exception e) {
        }
77      try {
          mClientSocket.close ();
79      }
        catch (Exception e) {
81      }
        if (mForwardingActive) {
83        System.out.println("TCP Forwarding " + mClientSocket.
              getInetAddress().getHostAddress() + ":"
           + mClientSocket.getPort() + " <--> " + mServerSocket.
               getInetAddress().getHostAddress() + ":"
85         + mServerSocket.getPort() + " stopped.");
          mForwardingActive = false;
87      }
      }
89 }
```

A.4 Comparison of Magnet Link Collections to PID Target URLs

For evaluating the approach of PIDs with persistent resolution targets, we provide a comparison of Magnet Link collections against the target URLs of existing PID collections.

A.4.1 Minera Handle Miner

In this section, we provide excerpts of the Minera Handle Miner source code, which focus on the PID resolution process.

Listing A.30: Interactive Mining Control Console

```
 1  ___    __
    |  \/  (_)
 3  | \  / |_ _ __   ___ _ __ __ _
    | |\/| | | '_ \ / _ \ '__/ _` |
 5  | |  | | | | | | |  __/ | | (_| |
    |_|  |_|_|_| |_|\___|_|  \__,_|
 7   Massively Parallel Handle Mining

 9  Detected CPUs:      4
    JVM version:        1.8.0_102
11  JVM vendor:         Oracle Corporation
    Operating System:   Linux 4.7.7-100.fc23.x86_64
13  System architecture: amd64
    Available Memory:   3557 MB
15
    Please initialize the system with init <jobfile> or leave with exit
17  >> init Prefix-A/job
    init :: **********************************
19  init :: loading jobfile: Prefix-A/job
    init :: Inputfile invalid.
21  init :: **********************************
    Please initialize the system with init <jobfile> or leave with exit
23  >>
```

Listing A.31: Mining Process Control

```java
package de.gwdg.minera.cmd;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.HashMap;
import java.util.Vector;

import de.gwdg.minera.cmd.cmdlets.*;
import de.gwdg.minera.threads.ProgramControl;
import de.gwdg.minera.threads.ProgramControlAction;
import de.gwdg.minera.threads.ThreadMonitor;

public class RunEvalPrintLoop {

  private Vector<ICliCommand> commands;
  private HashMap<String, String> environmentVariables;
  private ProgramControl programcontrol;
  private ThreadMonitor threadmonitor;
  private Thread threadmonitorHandle;

  public RunEvalPrintLoop() throws IOException {

    // Initialize command set
    this.commands = new Vector<ICliCommand>();

    this.commands.add(new Banner());
    this.commands.add(new Boost());
    this.commands.add(new Down());
    this.commands.add(new Exit());
    this.commands.add(new Help());
    this.commands.add(new Init());
    this.commands.add(new Pause());
    this.commands.add(new Run());
    this.commands.add(new State());
    this.commands.add(new Up());
    this.setBackReferences();

    // Initialize environment variables
    this.initEnvironmentVariables();

    // Print the Banner
    new Banner().execute(null);

    // Start interactive initialization
    this.forceSystemInitThroughUser();
  }

  private void initEnvironmentVariables() {
    this.environmentVariables = new HashMap<String, String>();
```

```
51      this.environmentVariables.put("prompt", ">>");
      }

      private void setBackReferences() {
55        for (ICliCommand command : this.commands) {
          command.setREPLBackReference(this);
57        }
      }

59
      public void setEnvironmentVariable(String key, String value) {
61        this.environmentVariables.put(key, value);
      }

63
      public String getEnvironmentVariable(String key) {
65        return this.environmentVariables.get(key);
      }

67
      public Vector<ICliCommand> getCommandSet(){
69        return this.commands;
      }

71
      public ProgramControl getProgramControl() {
73        return this.programcontrol;
      }

75
      public void enterRunEvalPrintLoop() {
77        BufferedReader br = new BufferedReader(new InputStreamReader(
            System.in));
        String currentUserInput = null;
79        String[] tokenizedInput = null;
        boolean commandNotExisting = true;

81
        while (true) {
83          commandNotExisting = true;
          try {
85            System.out.print(this.environmentVariables.get("prompt") + "
                ");
            currentUserInput = br.readLine();
87            if (currentUserInput.length() < 1)
              continue;
89            tokenizedInput = currentUserInput.split(" ");
          } catch (IOException ioe) {
91            System.out.println("IO error trying to read user input");
            System.exit(1);
93          }
          for (ICliCommand command : this.commands) {
95            if (command.getCommandKeyword().equals(tokenizedInput[0].
              toLowerCase())) {
              command.execute(tokenizedInput);
97              commandNotExisting = false;
            }
```

```
 99       }
          if (commandNotExisting)
101         System.out.println(tokenizedInput[0] + " :: command not found
              .");
        }
103   }

105   private void forceSystemInitThroughUser() {
        System.out.println("Please initialize the system with init <
          jobfile> or leave with exit");
107     BufferedReader br = new BufferedReader(new InputStreamReader(
          System.in));
        String currentUserInput = null;
109     String[] tokenizedInput = null;
        while (true) {
111       try {
            System.out.print(this.environmentVariables.get("prompt") + "
              ");
113         currentUserInput = br.readLine();
            if (currentUserInput.length() < 1)
115           continue;
            tokenizedInput = currentUserInput.split(" ");
        } catch (IOException ioe) {
            System.out.println("IO error trying to read user input");
119         System.exit(1);
        }
121       // Allow leaving the program
          if(tokenizedInput[0].equals("exit"))
123         System.exit(0);
          // Continue initialization
125       Init initcommand = new Init();
          initcommand.setREPLBackReference(this);
127       initcommand.execute(tokenizedInput);
          if (this.getEnvironmentVariable("inputfile") != null && this.
            getEnvironmentVariable("tafiledirectory") != null
129         && this.getEnvironmentVariable("outputdirectory") != null)
              {
          this.programcontrol = ProgramControl.getInstance();
131       this.threadmonitor = new ThreadMonitor(this.
              getEnvironmentVariable("inputfile"),
               this.getEnvironmentVariable("tafiledirectory"), this.
                getEnvironmentVariable("outputdirectory"), this.
                programcontrol);
133       this.programcontrol.changeExecution(ProgramControlAction.IDLE
              );
          this.threadmonitorHandle = new Thread(this.threadmonitor);
135       this.threadmonitorHandle.start();
          return;
137       } else
            System.out.println("Please initialize the system with init <
                jobfile> or leave with exit");
```

```
139        }

141      }
      }
```

Listing A.32: Input File Processor

```java
package de.gwdg.minera.io;

import de.gwdg.minera.pojo.Cent;

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.Vector;

import au.com.bytecode.opencsv.CSVReader;

public class FileInput {

  private static FileInput instance = null;
  private Vector<Cent> content;
  private int pointer = 0;
  private int batchsize = 20;

  public static FileInput getInstance(String filepath, String
      taDirectory) {
    if (instance == null) {
      instance = new FileInput(filepath, taDirectory);
    }
    return instance;
  }

  public FileInput(String filepath, String taDirectory) {
    this.content = new Vector<Cent>();
    Cent currentcent = null;
    CSVReader reader = null;
    Iterator<String[]> csviterator = null;
    String[] record = null;
    ArrayList<String> transactionlog = this.readTransactionlog(
        filepath, taDirectory);

    try {
      reader = new CSVReader(new FileReader(filepath), ',');
      csviterator = reader.readAll().iterator();
    } catch (FileNotFoundException e) {
      System.out.println("SEVERE :: " + filepath + " not found.
          Exiting.");
      System.exit(-1);
    } catch (IOException e) {
      System.out.println("SEVERE :: " + filepath + " not parsable as
          CSV-file. Exiting.");
      System.exit(-1);
    }
```

```
      // Skip first line
48    record = csviterator.next();

50    while (csviterator.hasNext()) {
        record = csviterator.next();
52      // If the uuid has been processed skip it
        if (transactionlog.contains(record[0]))
54        continue;
        currentcent = new Cent();
56      currentcent.setSerialnumber(record[0]);
        currentcent.setPrefix(record[3]);
58      currentcent.setSuffix(record[4]);
        this.content.addElement(currentcent);
60    }
    }

62
    private ArrayList<String> readTransactionlog(String filepath,
      String taDirectory) {
64    FileReader filereader = null;
      String line = null;
66    String transactionLogFilePath = CommonIO.getTransactionFilePath(
          filepath, taDirectory);
      ArrayList<String> output = new ArrayList<String>();
68    try {
        filereader = new FileReader(transactionLogFilePath);
70      BufferedReader bufferedreader = new BufferedReader(filereader);
        while ((line = bufferedreader.readLine()) != null) {
72        output.add(line);
        }
74      bufferedreader.close();
      } catch (FileNotFoundException e) {
76      System.out.println("WARNING :: Transaction log " +
            transactionLogFilePath + " not found");
      } catch (IOException e) {
78      System.out.println("WARNING :: Transaction log " +
            transactionLogFilePath + " not readable. Exiting");
        System.exit(-1);
80    }
      return output;
82  }

84  public synchronized void setBatchsize(int batchsize) {
      this.batchsize = batchsize;
86  }

88  public synchronized Vector<Cent> getBatch() {
      Vector<Cent> output = new Vector<Cent>();
90    int batchcounter = 0;
      while ((batchcounter < this.batchsize) && ((this.content.size() -
          1) >= this.pointer)) {
92      output.add(this.content.get(this.pointer));
```

```
        batchcounter++;
94      this.pointer++;
      }
96    return output;
    }
98 }
```

Listing A.33: Mining Worker Thread

```
package de.gwdg.minera.threads;

import de.gwdg.minara.exceptions.ContentNotProebableException;
import de.gwdg.minara.exceptions.HandleNotResolvableException;
import de.gwdg.minera.contentprobe.ContentProber;
import de.gwdg.minera.handle.HandleResolver;
import de.gwdg.minera.io.CoinPersistence;
import de.gwdg.minera.io.FileInput;
import de.gwdg.minera.pojo.Cent;
import de.gwdg.minera.pojo.Probe;
import de.gwdg.minera.pojo.ResolutionResult;

import java.util.Vector;

public class Worker implements Runnable {

  private FileInput fileinput;
  private CoinPersistence coinpersistence;
  private HandleResolver handleresolver;
  private ContentProber contentprober;
  private ProgramControl main;
  String inputfilepath;
  private String taDirectory;
  private String coinOutputDirectory;

  public Worker(String inputfilepath, String taDirectory, String
      coinOutputDirectory, ProgramControl main) {
    this.handleresolver = new HandleResolver();
    this.contentprober = new ContentProber();
    this.fileinput = FileInput.getInstance(inputfilepath, taDirectory
        );
    this.coinpersistence = CoinPersistence.getInstance();
    this.main = main;
    this.inputfilepath = inputfilepath;
    this.taDirectory = taDirectory;
    this.coinOutputDirectory = coinOutputDirectory;
  }

  @Override
  public void run() {
    Vector<Cent> centbatch = new Vector<Cent>();
    this.main.reportNumberOfRunningThreads(ThreadCounterAction.
        INCREMENT);
    // System.out.println("INFO :: Thread " + this.hashCode() +
    // " has started up.");
    while (true) {
      // If idle was selected then exit loop
      if (this.main.changeExecution(ProgramControlAction.INFO) ==
          ProgramControlAction.IDLE)
        break;
```

```
        // Work on new batch
48      centbatch = fileinput.getBatch();
        if (centbatch.size() == 0)
50        break;
        else {
52        try {
            this.processCentbatch(centbatch, taDirectory,
              coinOutputDirectory);
54        } catch (Exception e) {
            System.out.println("SEVERE :: Thread " + this.hashCode() +
              " died. Cause:");
56          e.printStackTrace();
            break;
58        }
        }
60    }
      if (centbatch.size() == 0)
62      this.main.changeExecution(ProgramControlAction.SHUTDOWN);
      this.main.reportNumberOfRunningThreads(ThreadCounterAction.
        DECREMENT);
64  }

66  private void processCentbatch(Vector<Cent> centbatch, String
      taDirectory, String coinOutputDirectory) {
      Vector<Cent> processedcentbatch = new Vector<Cent>();
68    Cent processedcent;
      for (Cent currentcent : centbatch) {
70      if (isCentSain(currentcent) == false) {
          System.out.println("INFO :: Malforemed input data caused
              skipping cent: " + currentcent.getSerialnumber() + " in
              thread: "
72            + this.hashCode());
          continue;
74      }
        try {
76        processedcent = this.processCent(currentcent);
          processedcentbatch.add(processedcent);
78      } catch (Exception e) {
          System.out.println("INFO :: Exeception caused skipping cent:
              " + currentcent.getSerialnumber() + " in thread: "
80            + this.hashCode());
          continue;
82      }
      }
84    this.coinpersistence.persistBatchToCoins(processedcentbatch, this
        .inputfilepath, taDirectory, coinOutputDirectory);
    }
86
    private Cent processCent(Cent currentcent) throws
      HandleNotResolvableException, ContentNotProebableException {
88    Cent workingcent = currentcent;
```

```
90      // Handle resolution
        ResolutionResult result = this.handleresolver.
            resolveHandleUrltoIP(workingcent.getPrefix(), workingcent.
            getSuffix());
92      workingcent.setTargetHostname(result.getHostname());
        workingcent.setTargetIp(result.getIp());
94      workingcent.setTargetUrl(result.getUrl());

96      // Content probing
        Probe probe = this.contentprober.probeContent(workingcent.
            getTargetUrl());
98      workingcent.setTargetMime(probe.getMime());
        workingcent.setTargetProbeMd5Sum(probe.getMd5Checksum());
100     workingcent.setTargetSize("" + probe.getSize());

102     return workingcent;
    }

104
    private boolean isCentSain(Cent currentcent) {
106     if (currentcent.getSerialnumber().length() < 30)
          return false;
108     if (currentcent.getSuffix().length() < 1)
          return false;
110     if (currentcent.getPrefix().length() < 1)
          return false;
112     return true;
    }
114 }
```

Listing A.34: Handle Resolver

```
package de.gwdg.minera.handle;

import java.io.IOException;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;
import java.net.UnknownHostException;

import de.gwdg.minara.exceptions.HandleNotResolvableException;
import de.gwdg.minera.pojo.ResolutionResult;

public class HandleResolver {

  public HandleResolver() {
    System.setProperty("http.agent", "Minera");
  }

  private String getRandomHandleServer(){
    //Alternative code - random handle server assignment
    String[] server = {"38.100.138.165"};
    return server[new java.util.Random().nextInt(server.length)];
  }

  private String resolveHandleUrltoUrl(String prefix, String suffix)
      throws HandleNotResolvableException {
    String resolvedUrl = "";
    HttpURLConnection httpconnection = null;
    try {
      URL urlToResolve = new URL("http://" + this.
          getRandomHandleServer() + "/" + prefix + "/" + suffix);
      httpconnection = (HttpURLConnection) urlToResolve.
          openConnection();
      httpconnection.setInstanceFollowRedirects(false);
      httpconnection.setRequestProperty("User-Agent", "Minera");
      httpconnection.connect();
      if(httpconnection.getResponseCode() != HttpURLConnection.
          HTTP_SEE_OTHER)
        throw new HandleNotResolvableException(prefix, suffix);
      resolvedUrl = httpconnection.getHeaderField("Location");
      httpconnection.disconnect();
    } catch (MalformedURLException e) {
      httpconnection.disconnect();
      throw new HandleNotResolvableException(prefix, suffix);
    } catch (IOException e) {
      httpconnection.disconnect();
      throw new HandleNotResolvableException(prefix, suffix);
    }
    if (resolvedUrl == null){
      httpconnection.disconnect();
      throw new HandleNotResolvableException(prefix, suffix);
```

```
     }
48   return resolvedUrl;
   }

50
   public ResolutionResult resolveHandleUrltoIP(String prefix, String
       suffix) throws HandleNotResolvableException {
52   ResolutionResult resolutionresult;
     String handleTargetUrl = this.resolveHandleUrltoUrl(prefix,
         suffix);
54   try {
       resolutionresult = new ResolutionResult(new URL(handleTargetUrl
           ));
56   } catch (MalformedURLException e) {
       throw new HandleNotResolvableException(prefix, suffix);
58   } catch (UnknownHostException e) {
       e.printStackTrace();
60     throw new HandleNotResolvableException(prefix, suffix);
     }
62   return resolutionresult;
   }
64 }
```

A.4.2 PID Target URL Collection

For aggregating the PID target URL collections from the Minera Handle Miner (cf. Figure 6.13 and Appendix A.4.1), following data aggregation scrips have been used. They deliver the data for Table 6.4 and Figure 6.15.

Listing A.35: Script for Assembling Target URL and Magnet Link Collection Data

```python
#!/usr/bin/python

import os
import operator
import pickle
import gzip

def get_csv_files():
  files = []
  for file in os.listdir('/mnt/data/output/'):
    if file.endswith('.csv.gz'):
      files.append(file)
  return files

def get_csv_file_names_ordered_size_desc():
  files = get_csv_files()
  file_size_dict = {}
  for file in files:
    num_lines = sum(1 for line in gzip.open('/mnt/data/output/' + str
        (file)))
    file_size_dict[str(file)] = num_lines
  return sorted(file_size_dict.items(), key=operator.itemgetter(1),
     reverse=True)

def get_url_lengths_from_csv(file):
  output = []
  with gzip.open('/mnt/data/output/' + str(file)) as f:
    for line in f:
      url = line.split(',')[15]
      output.append(len(url))
  return output

def get_url_length_with_files():
  output ={}
  files = get_csv_file_names_ordered_size_desc()
  for item_pair in files:
    output[item_pair[0]] = get_url_lengths_from_csv(item_pair[0])
  return output

def analyze_pirate_bay_file():
  sizes = []
  with gzip.open('/mnt/data/output/piratebay/complete.csv.gz') as f:
    for line in f:
```

```
42          chunks = line.split('|')
            filename = chunks[1]
44          btih = chunks[5]
            magnet_link = 'magnet:?xt=' + str(btih) + '&dn=' + str(filename
                )
46          sizes.append(len(magnet_link))
    return ('piratebay', sizes)
48
def analyze_academic_torrents_file():
50      sizes = []
    with gzip.open('/mnt/data/output/acacemic_torrents/20160902
        _academic_torrents_magnet_links_dump.txt.gz') as f:
52      for line in f:
            magnet_link = line.replace('\n','')
54          sizes.append(len(magnet_link))
    return ('acadmic_torrents', sizes)
56
raw_data = get_url_length_with_files()
58 captions = []
data = []
60 magnetlink_data = analyze_pirate_bay_file()
at_data = analyze_academic_torrents_file()
62 ml_minimal_example = [len('magnet:?xt=urn:btih:411576
    c7e80787e4b40452360f5f24acba9b5159'), len(' magnet:?xt=urn:btih:
    fc0b958e0d8b958145d52e2eb641ee83a4d32ade')]

64 # This was Top-Prefix 10 - Prefix-J
captions.append('#10 - Prefix-J')
66 data.append(raw_data['Prefix-J_minera_fusioned_output.csv.gz'])

68 # This was Top-Prefix 9 - Prefix-I
captions.append('#9 - Prefix-I')
70 data.append(raw_data['Prefix-I_minera_fusioned_output.csv.gz'])

72 # This was Top-Prefix 8 - Prefix-H
captions.append('#8 - Prefix-H')
74 data.append(raw_data['Prefix-H_minera_fusioned_output.csv.gz'])

76 # This was Top-Prefix 7 - Prefix-G
captions.append('#7 - Prefix-G')
78 data.append(raw_data['Prefix-G_minera_fusioned_output.csv.gz'])

80 # This was Top-Prefix 6 - Prefix-F
captions.append('#6 - Prefix-F')
82 data.append(raw_data['Prefix-F_minera_fusioned_output.csv.gz'])

84 # This was Top-Prefix 5 - Prefix-E
captions.append('#5 - Prefix-E')
86 data.append(raw_data['Prefix-E_minera_fusioned_output.csv.gz'])

88 # This was Top-Prefix 4 - Prefix-D
```

```
     captions.append('#4 - Prefix-D')
 90  data.append(raw_data['Prefix-D_minera_fusioned_output.csv.gz'])

 92  # This was Top-Prefix 3 - Prefix-C - data present
     captions.append('#3 - Prefix-C')
 94  data.append(raw_data['Prefix-C_minera_fusioned_output.csv.gz'])

 96  # This was Top-Prefix 2 - Prefix-B - data present
     captions.append('#2 - Prefix-B')
 98  data.append(raw_data['Prefix-B_minera_fusioned_output.csv.gz'])

100  # This was Top-Prefix 1 - Prefix-A - data present
     captions.append('#1 - Prefix-A')
102  data.append(raw_data['Prefix-A_minera_fusioned_output.csv.gz'])

104  # Pirate Bay Data
     captions.append('ML Pirate Bay')
106  data.append(magnetlink_data[1])

108  # Academic Torrents
     captions.append('ML Academic Torrents')
110  data.append(at_data[1])

112  # Minimal Magnet Link Example
     captions.append('ML Minimal Example')
114  data.append(ml_minimal_example)

116  export_path_data = '/tmp/data.bin'
     export_path_captions = '/tmp/captions.bin'
118
     with open(export_path_data, 'wb') as datafile:
120    pickle.dump(data, datafile)
       print "Data dumped with", os.stat(export_path_data).st_size, "bytes
         "
122
     with open(export_path_captions, 'wb') as captionfile:
124    pickle.dump(captions, captionfile)
       print "Captions dumped with", os.stat(export_path_captions).st_size
         , "bytes"
```

Listing A.36: Script for Visualization of Data (Figure 6.15)

```
1
  import matplotlib.pyplot as plt
3 import numpy as np
  import pandas as pd
5 from matplotlib.backends.backend_pdf import PdfPages
  import pickle
7
  export_path_data = '/tmp/data.bin'
9 export_path_captions = '/tmp/captions.bin'
  export_pdf_path = '/tmp/chart.pdf'
11 data = ''
  captions = ''
13

15 with open(export_path_data, "rb") as datafile:
    print "Reading", export_path_data
17   data = pickle.load(datafile)

19 with open(export_path_captions, "rb") as captionfile:
    print "Reading", export_path_captions
21   captions = pickle.load(captionfile)

23 print "rendering graphics ..."
  with PdfPages(export_pdf_path) as pdf:
25   fig = plt.figure()
    df = pd.DataFrame(data, index=captions)
27   plt.xlabel('number of characters')
    plt.xlim(0, 500)
29   bp = df.T.boxplot(vert=False, sym='+', return_type='dict')
    plt.setp(bp['boxes'], color='black')
31   plt.setp(bp['whiskers'], color='black')
    plt.subplots_adjust(left=0.43, bottom=0.43)
33   fig.savefig(export_pdf_path + '.png', format='png', dpi=500)
    pdf.savefig(fig)
35   print "pdf saved as", export_pdf_path
    print "finished."
```

A.4.3 Academic Torrent Magnet Link Collection

For gathering the complete Magnet Link collection of Academic Torrents [97] a crawler
script and a conversion script has been used. In the following, we provide both scripts that
deliver the data for Table 6.4 and Figure 6.15.

Listing A.37: Crawler Script For Downloading all Torrent Files From Academic Torrents
using the Official API

```python
#!/usr/bin/env python
import requests, urlib, json

torrent_url = "http://academictorrents.com/download/"
api_url="http://academictorrents.com/apiv2/"

# Dowload all the research data
r = requests.get(api_url+"entries?cat=6&limit=100000&uid=
    USERID_REMOVED&pass=PASSWORD_REMOVED")
print("I found "+str(len(r.json()))+" datasets");
for dataset in r.json():
  dataset['name'] = urllib.quote(dataset['name'].encode('utf8'))
  with open("datasets/"+dataset['infohash']+".json","w") as fd:
    fd.write(json.dumps(dataset))
  t_dl = requests.get(torrent_url+dataset['infohash']+".torrent",
      stream=True)
  with open("datasets/"+dataset['infohash']+".torrent",'w') as fd:
    for chunk in t_dl.iter_content(1024):
      fd.write(chunk)

# Download all the papers
r = requests.get(api_url+"entries?cat=5&limit=100000&uid=
    USERID_REMOVED&pass=PASSWORD_REMOVED")
print("I found "+str(len(r.json()))+" papers");
for dataset in r.json():
  dataset['name'] = urllib.quote(dataset['name'].encode('utf8'))
  with open("papers/"+dataset['infohash']+".json","w") as fd:
    fd.write(json.dumps(dataset))
  t_dl = requests.get(torrent_url+dataset['infohash']+".torrent",
      stream=True)
  with open("papers/"+dataset['infohash']+".torrent",'w') as fd:
    for chunk in t_dl.iter_content(1024):
      fd.write(chunk)
```

Listing A.38: Script For Access Data Into Standardized Magnet Links

```python
#!/usr/bin/env python

import os;
import json;
import urllib;

for f in os.listdir("datasets/"):
  if f.endswith(".json"):
    with open("datasets/"+f,"r") as fd:
      j = json.loads(fd.read())
      m = 'magnet:?xt=urn:btih:'+j['infohash']+'&dn='+j['name']
      with open('datasets/'+j['infohash']+'_magnet-link.txt','w') as
          wf:
        wf.write(m)

for f in os.listdir("papers/"):
  if f.endswith(".json"):
    with open("papers/"+f,"r") as fd:
      j = json.loads(fd.read())
      m = 'magnet:?xt=urn:btih:'+j['infohash']+'&dn='+j['name']
      with open('papers/'+j['infohash']+'_magnet-link.txt','w') as wf:
        wf.write(m)
```

A.5 PIDs with Persistent Resolution Targets

For evaluation the approach of PIDs with persistent resolution targets a full software stack has been implemented. In this section, we provide the screen shots of the user interface and source codes of the proof-of-concept software named *PID-Burner*.

A.5.1 User Interface

In this section screenshots of the proof-of-concept implementation of the web-service PID-Burner are included. All screenshots are showing the web application and were made on OSX Version 10.11.6 using Google Chrome 52.0.2743.116.
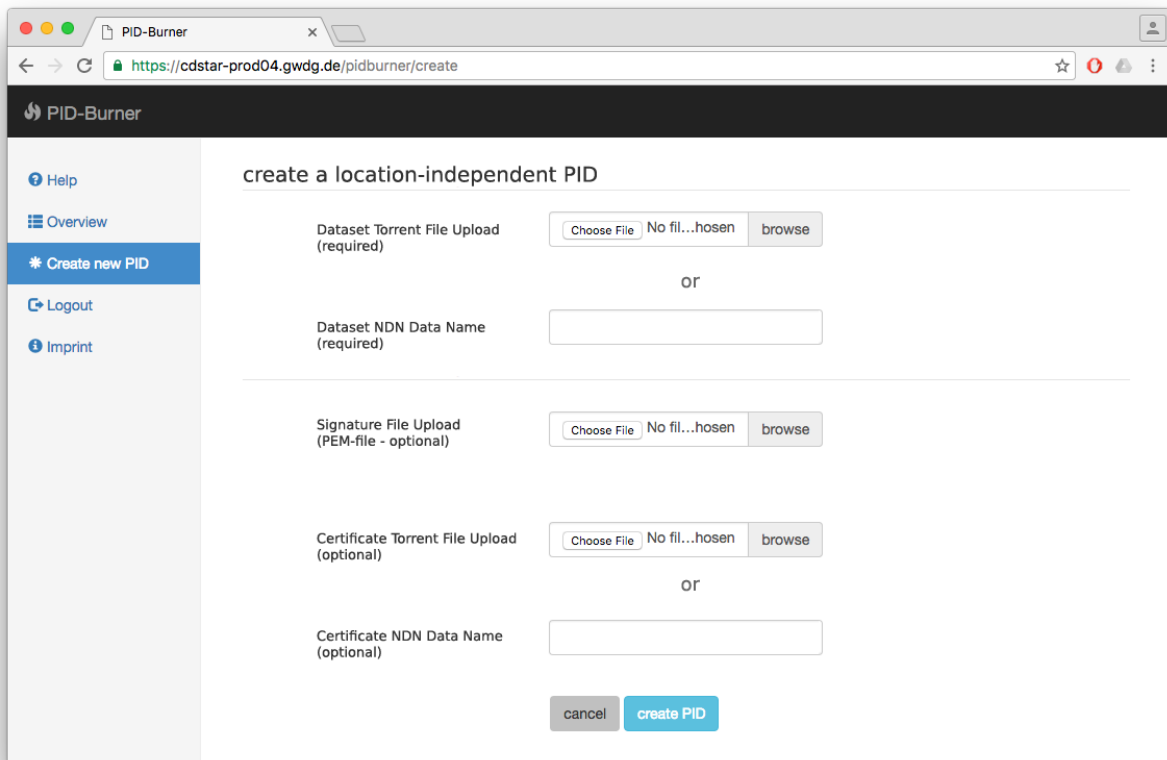

Figure A.1: Creating a Magnet Link-enabled PID in PID-Burner

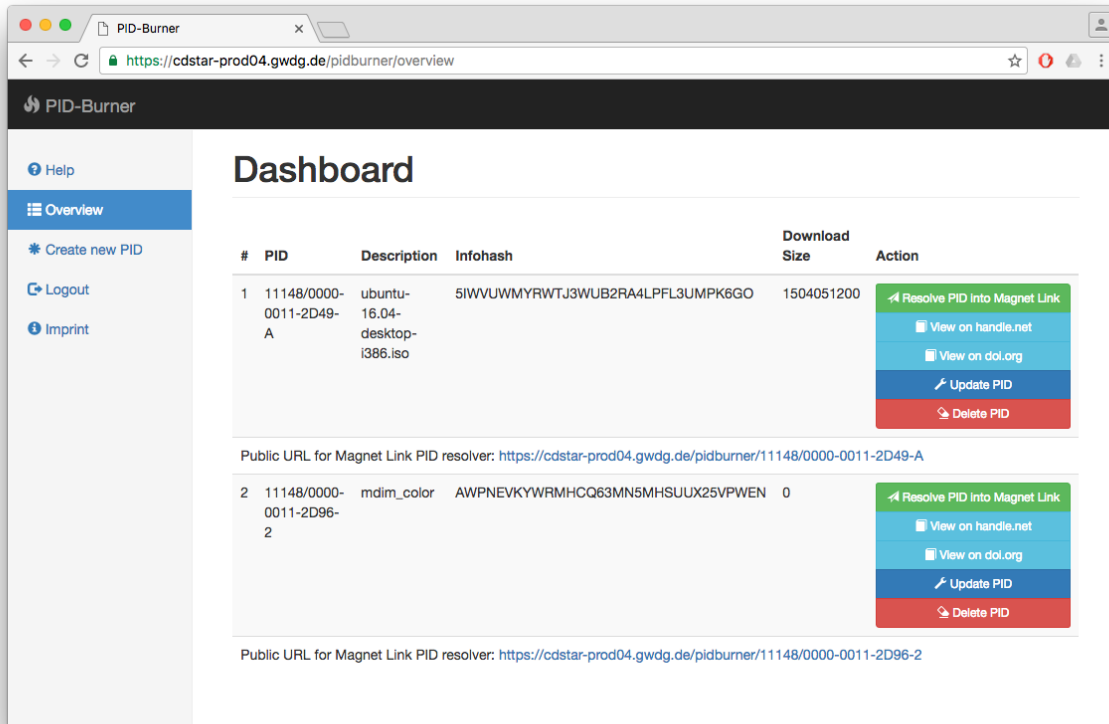Figure A.2: PID Management Interface in PID-Burner
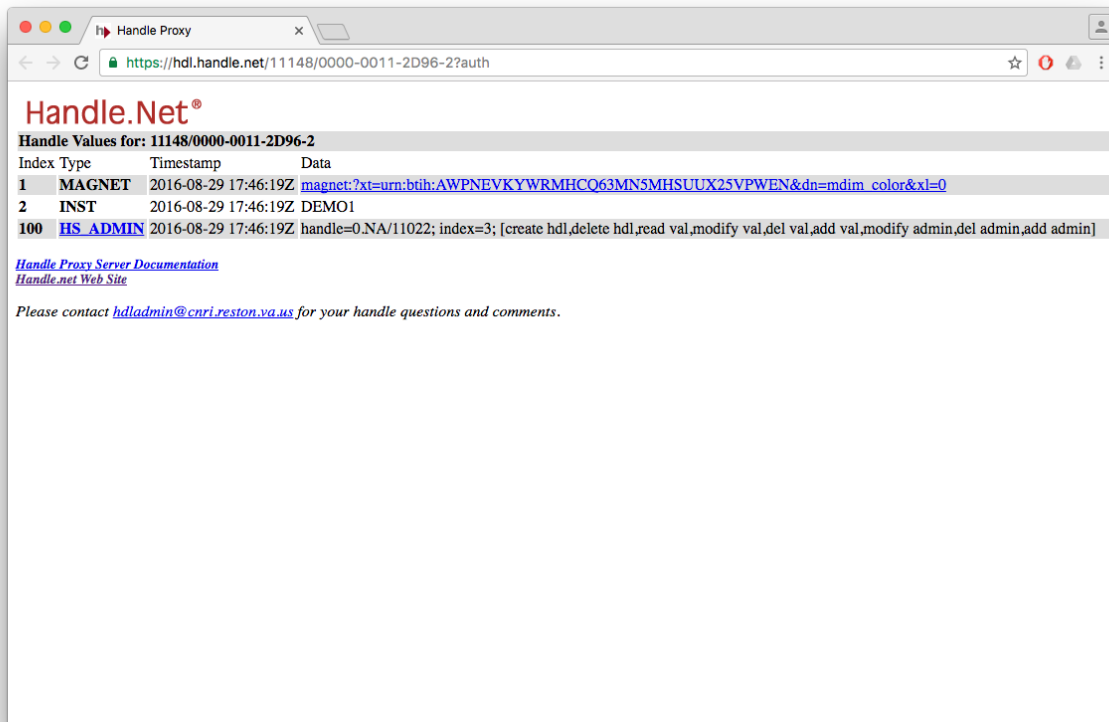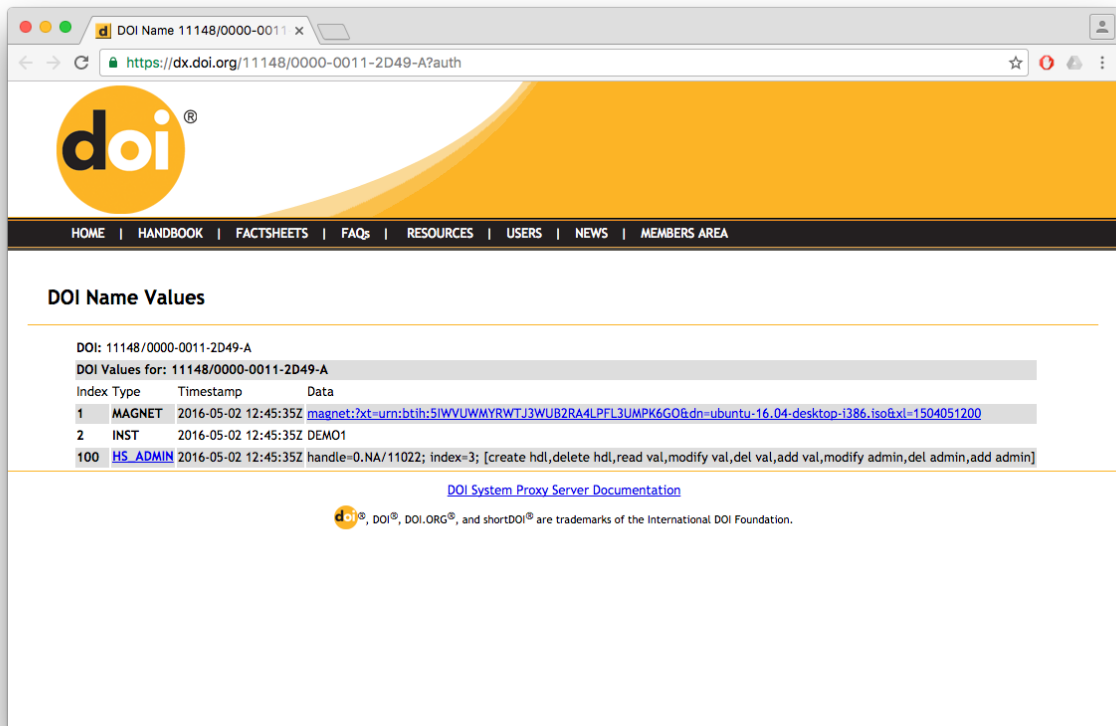
Figure A.3: PID with BitTorrent Access Data on hdl.handle.net

Figure A.4: PID with BitTorrent Access Data on dx.doi.org

A.5.2 Source Code

In the following, we provide excerpts of the source code from the web application PID Burner. The excerpts depict the management of Magnet Links in conjunction with the Handle system using the EPIC REST interface.

Listing A.39: Interaction with the EPIC PID Rest Interface for creating and maintaining PIDs

```
 1  import httplib2
    import os
 3  import json
    import base64
 5  import logging
    from pprint import pprint
 7
    WEBROOT = os.path.abspath(os.path.dirname(__file__))
 9  logging.basicConfig(filename=WEBROOT + '/log/log.txt', format=logging
        .BASIC_FORMAT)
11
    class epicclient(object):
13      def __init__(self):
            userdetails = self.read_users_details()
15          self.username = userdetails['username']
            self.password = userdetails['password']
17          self.host = userdetails['host']
            self.prefix = userdetails['prefix']
19
        def create_pid(self, magnet_link):
21          self.__error50xworkaround()
            h = httplib2.Http()
23          auth = base64.encodestring(self.username + ':' + self.
                password)
            uploaddata = self.create_upload_body(magnet_link)
25          response, content = h.request(
                self.host + '/handles/' + self.prefix + '/',
27              'POST',
                headers={'Authorization': 'Basic ' + auth, 'User-Agent':
                    "curl/7.43.0", 'Accept': "application/json",
29                  'Content-Type': "application/json"},
                body=uploaddata
31          )
            if response['status'] == '201':
33              parse_content = json.loads(content)
                parse_content = parse_content['epic-pid']
35              suffix = parse_content.split('/')[1]
                logging.info("[CREATE MAGNET LINK] 201 PID Created -
                    suffix: " + suffix + " magnet-link: " + magnet_link)
37              return suffix
            if response['status'][0] == '5':
```

```
39              logging.info(
                    "[CREATE MAGNET LINK] Server failed - status: " +
                        response['status'] + " magnet-link: " +
                        magnet_link)
41          raise SystemError('[CREATE MAGNET LINK] EPIC Server Error
                5xx')
        logging.info(
43          "[CREATE MAGNET LINK] Unknown error - status: " +
                response['status'] + " magnet-link: " + magnet_link)
        raise KeyError('[CREATE MAGNET LINK] unknown error')
45

    @staticmethod
47  def create_upload_body(magnet_link):
        # type: (object) -> str
49      body = []
        body.append(dict(type="MAGNET", parsed_data=magnet_link))
51      body.append(dict(type="INST", parsed_data="DEMO1"))
        return json.dumps(body)
53

    def update_pid(self, suffix, magnet_link):
55      self.__error50xworkaround()
        h = httplib2.Http()
57      auth = base64.encodestring(self.username + ':' + self.
            password)
        uploaddata = self.create_upload_body(magnet_link)
59      response, content = h.request(
            self.host + '/handles/' + self.prefix + '/' + suffix,
61          'PUT',
            headers={'Authorization': 'Basic ' + auth, 'User-Agent':
                "curl/7.43.0", 'Accept': "application/json",
63                  'Content-Type': "application/json"},
            body=uploaddata
65      )
        # EPIC developers? 204 is WRONG - What about using 200?!
67      if response['status'] == '204':
            logging.info("[UPDATE MAGNET LINK] 204 PID updated -
                suffix: " + suffix + " magnet-link: " + magnet_link)
69          return suffix
        if response['status'] == '404':
71          logging.info("[UPDATE MAGNET LINK] 404 Not found - suffix
                : " + suffix + " magnet-link: " + magnet_link)
            raise KeyError('[UPDATE MAGNET LINK] PID not found')
73      if response['status'][0] == '5':
            logging.info(
75              "[UPDATE MAGNET LINK] Server failed - status: " +
                    response['status'] + " magnet-link: " +
                    magnet_link)
            raise SystemError('[UPDATE MAGNET LINK] EPIC Server Error
                5xx')
77      logging.info(
            "[UPDATE MAGNET LINK] Unknown error - status: " +
```

```python
                    response['status'] + " magnet-link: " + magnet_link)
79          raise KeyError('[UPDATE MAGNET LINK] unknown error')

81      def delete_pid(self, suffix):
            self.__error50xworkaround()
83          h = httplib2.Http()
            auth = base64.encodestring(self.username + ':' + self.
                password)
85          response, content = h.request(
                self.host + '/handles/' + self.prefix + '/' + suffix,
87              'DELETE',
                headers={'Authorization': 'Basic ' + auth, 'User-Agent':
                    "curl/7.43.0", 'Accept': "application/json"}
89          )
            if response['status'] == '204':
91              logging.info("[DELETE PID] 204 PID deleted - suffix: " +
                    suffix)
                return 'OK'
93          if response['status'] == '404':
                logging.info("[DELETE PID] 404 Not found - suffix: " +
                    suffix)
95              raise KeyError('[DELETE PID] PID not found')
            if response['status'][0] == '5':
97              logging.info("[DELETE PID] Server failed - status: " +
                    response['status'] + " - suffix: " + suffix)
                raise SystemError('[DELETE PID] EPIC Server Error 5xx')
99          logging.info("[DELETE PID] Unknown error - status: " +
                response['status'] + " - suffix: " + suffix)
            raise KeyError('[DELETE PID] unknown error')
101
        def get_magnet_link_from_pid(self, suffix):
103         self.__error50xworkaround()
            h = httplib2.Http()
105         auth = base64.encodestring(self.username + ':' + self.
                password)
            response, content = h.request(
107             self.host + '/handles/' + self.prefix + '/' + suffix,
                'GET',
109             headers={'Authorization': 'Basic ' + auth, 'User-Agent':
                    "curl/7.43.0", 'Accept': "application/json"}
            )
111         if response['status'] == '200':
                data = json.loads(content)
113             for field in data:
                    if field['type'] == 'MAGNET':
115                     return field['parsed_data']
                logging.info("GET MAGNET LINK] PID does not contain a
                    Magnet Link - suffix: " + suffix)
117             raise KeyError('[GET MAGNET LINK] PID does not contain a
                    Magnet Link')
            if response['status'] == '404':
```

```
119              logging.info("[GET MAGNET LINK] 404 Not found - suffix: "
                     + suffix)
                 raise KeyError('[GET MAGNET LINK] PID not found')
121          if response['status'][0] == '5':
                 logging.info("[GET MAGNET LINK] Server failed - status: "
                     + response['status'] + " suffix: " + suffix)
123              raise SystemError('[GET MAGNET LINK] EPIC Server Error 5
                     xx')
             logging.info("[GET MAGNET LINK] Unknown error - status: " +
                 response['status'] + " - suffix: " + suffix)
125          raise KeyError('[GET MAGNET LINK] unknown error')

127      def get_pid_target_from_handle(self, prefix, suffix):
             h = httplib2.Http()
129          response, content = h.request(
                 'https://hdl.handle.net/api/handles/' + prefix + '/' +
                     suffix + '?auth=true',
131              'GET',
                 headers={'User-Agent': "curl/7.43.0", 'Accept': "
                     application/json"}
133          )
             if response['status'] == '200':
135              data = json.loads(content)
                 for field in data['values']:
137                  if field['type'] == 'MAGNET' or field['type'] == 'URL
                         ':
                         return field['data']['value']
139              logging.info("GET MAGNET LINK] PID does not contain a
                     Magnet Link - suffix: " + suffix)
                 raise KeyError('[GET PID FROM HANDLE_NET] PID does not
                     contain a Magnet Link')
141          if response['status'] == '404':
                 logging.info("[GET PID FROM HANDLE_NET] 404 Not found -
                     suffix: " + suffix)
143              raise KeyError('[GET PID FROM HANDLE_NET] PID not found')
             if response['status'][0] == '5':
145              logging.info("[GET MAGNET LINK] Server failed - status: "
                     + response['status'] + " suffix: " + suffix)
                 raise SystemError('[GET PID FROM HANDLE_NET] EPIC Server
                     Error 5xx')
147          logging.info("[GET PID FROM HANDLE_NET] Unknown error -
                 status: " + response['status'] + " - suffix: " + suffix)
             raise KeyError('[GET PID FROM HANDLE_NET] unknown error')
149
         def __error50xworkaround(self):
151
             def __exec_workaround():
153              attempts = 0
                 while not __is_workaround_working():
155                  if attempts > 3:
                         logging.info("[50x EPIC WORKAROUND] four attempts
```

```
                                    failed. Giving up.")
157                     return
                    attempts += 1
159

         def __is_workaround_working():
161         try:
                h = httplib2.Http()
163             auth = base64.encodestring(self.username + ':' + self
                    .password)
                uploaddata = self.create_upload_body('testmagnet')
165             response, content = h.request(
                    self.host + '/handles/' + self.prefix + '/' + '
                        0000-0011-2C12-8',
167                 'PUT',
                    headers={'Authorization': 'Basic ' + auth, 'User-
                        Agent': "curl/7.43.0",
169                         'Accept': "application/json",
                            'Content-Type': "application/json"},
171             body=uploaddata
                )
173             if response['status'][0] == '5':
                    logging.info("[50x EPIC WORKAROUND] saved request
                         from being scrapped")
175                 return False
                else:
177                 return True
            except:
179             return False

181     __exec_workaround()

183 def read_users_details(self):
        webroot = os.path.abspath(os.path.dirname(__file__))
185     filepath = os.path.join(webroot, 'config', 'epic_login.json')
        with open(filepath) as data_file:
187         return json.load(data_file)
```

Listing A.40: Extraction of BitTorrent Access Information

```python
from epicclient import epicclient
from storage import storage
import bencode
import hashlib
import base64
import time
import urllib


class pidObject():
    def __init__(self):
        expires = time.time()
        self.timestamp = time.strftime("%a, %d-%b-%Y %T GMT", time.
            gmtime(expires))


class torrentObject(object):
    def __init__(self, name, length, infohash):
        self.name = name
        self.length = length
        self.infohash = infohash

    def get_name(self):
        return self.name

    def get_length(self):
        return self.length

    def get_infohash(self):
        return self.infohash


class pidstorage(object):
    def __init__(self, webroot):
        self.eclient = epicclient()
        self.storage = storage(webroot)

    def create_pid_with_torrent_file(self, filepath):
        storage_data = pidObject()
        magnet_link = self.convert_torrent_file_to_magnet_link(
            filepath)
        suffix = self.eclient.create_pid(magnet_link)
        self.storage.write_object_data_to_disk(suffix, storage_data)
        self.storage.archive_torrent_file(suffix, filepath)
        return suffix

    def get_pid_target_from_handle(self, prefix, suffix):
        return self.eclient.get_pid_target_from_handle(prefix, suffix
            )
```

```
     def update_pid_with_torrent_file(self, suffix, filepath):
49       if not self.storage.is_object_existing(suffix):
             raise KeyError('[PID STORAGE] Only PIDs managed by PID-
                 Burner can be updated.')
51       magnet_link = self.convert_torrent_file_to_magnet_link(
             filepath)
         retval = self.eclient.update_pid(suffix, magnet_link)
53       self.storage.write_object_data_to_disk(suffix, pidObject())
         self.storage.archive_torrent_file(suffix, filepath)
55       return retval

57   def delete_pid(self, suffix):
         if not self.storage.is_object_existing(suffix):
59           raise KeyError('[PID STORAGE] Only PIDs managed by PID-
                 Burner can be deleted.')
         self.eclient.delete_pid(suffix)
61       self.storage.delete_object_from_disk(suffix)
         self.storage.delete_torrent_file_safe(suffix)
63

     def convert_torrent_file_to_magnet_link(self, filepath):
65       torrent_properties = self.
             extract_properties_from_torrent_file(filepath)
         xt_param = 'xt=urn:btih:%s' % torrent_properties.get_infohash
             ()
67       dn_param = urllib.urlencode({'dn': torrent_properties.
             get_name()})
         xl_param = urllib.urlencode({'xl': torrent_properties.
             get_length()})
69       magneturi = 'magnet:?' + xt_param + '&' + dn_param + '&' +
             xl_param
         return magneturi
71

     def is_suffix_in_database(self, suffix):
73       return self.storage.is_object_existing(suffix)

75   def extract_properties_from_torrent_file(self, filepath):
         torrent = open(filepath, 'r').read()
77       metadata = bencode.bdecode(torrent)
         hashcontents = bencode.bencode(metadata['info'])
79       digest = hashlib.sha1(hashcontents).digest()
         infohash = base64.b32encode(digest)
81       try:
             name = metadata['info']['name']
83       except:
             name = 'name not set'
85       try:
             length = metadata['info']['length']
87       except:
             length = 0
89       return torrentObject(name, length, infohash)
```

A.5.3 PID Resolution Measurements For Various PID Sizes

Listing A.41: Script for Measuring PID Resolution Times With Various target URL Sizes

```python
import urllib2
import base64
from pprint import pprint
import os
import time
import random
import string
import sys

def updateGWDGPid(username, password, prefix, suffix, targeturl):
    opener = urllib2.build_opener(urllib2.HTTPHandler)
    request = urllib2.Request('http://pid.gwdg.de/handles/' + prefix
        + '/' + suffix, data='[{"type":"URL","parsed_data":"' +
        targeturl +'"}]')
    request.add_header('User-Agent', 'EPIC PID Reservation Tool -
        powered by GWDG (c) 2015')
    request.add_header('Content-Type', 'application/json')
    base64string = base64.encodestring('%s:%s' % (username, password)
        ).replace('\n', '')
    request.add_header("Authorization", "Basic %s" % base64string)
    request.get_method = lambda: 'PUT'
    response = opener.open(request)
    etag = response.info().getheader('Etag').replace('"','')
    return etag, prefix, suffix, 'https://hdl.handle.net/' + prefix +
        '/' + suffix, targeturl

def measureTimeForPIDResolving(prefix, suffix):
    # Prepare the request
    host = 'ec2-XXX-XXX-XXX-XXX.eu-west-1.compute.amazonaws.com'
    opener = urllib2.build_opener(urllib2.HTTPHandler)
    # Set ?auth parameter to turn off caching completely in the
        resolution chain
    request_address = 'http://' + host + '/api/handles/' + prefix + '
        /' + suffix + '?auth'
    request = urllib2.Request(request_address)
    request.add_header('User-Agent', 'Handle Benchmark Tool')
    request.add_header('Content-Type', 'application/json')
    request.get_method = lambda: 'GET'

    # Do the measurement
    starttime = time.time()
    response = opener.open(request)
    endtime = time.time()

    # Check if everthing went well, if not return -1 for the
        measurement
    statuscode = str(response.getcode())
    if statuscode != "200":
```

```
41          print "Resolving problem - Host: %s Statuscode: %s " % (
                request_address , statuscode )
            return -1
43      else:
            return ( endtime - starttime )
45
   def getRandomString ( length ):
47      return ''.join ( random . choice ( string . ascii_lowercase ) for _ in
            xrange (0 , length ))

49 def measurePID ( username , password , prefix , suffix ):
       # we test target URL length from 1 to 65536
51      sys . stdout . write ("[ Measurement ] " + prefix + '/' + suffix + ' ')
       results = []
53      for exp in xrange (0 , 16): # 2^15 max chars.
           target_url_length = 2 ** exp
55          random_string = getRandomString ( target_url_length )
           try:
57              # update the PID
               updateGWDGPid ( username , password , prefix , suffix ,
                   random_string )
59              # measure
               measured_time = measureTimeForPIDResolving ( prefix , suffix
                   )
61              results . append ( measured_time )
               sys . stdout . write ('..' + str (2 ** exp ))
63              sys . stdout . flush ()
               # Clean the PID up after using
65              updateGWDGPid ( username , password , prefix , suffix , 'http
                   :// www . gwdg . de ')
           except Exception , e:
67              print " "
               print "[ Measurement PROBLEM ] %s/%s - STL: %s " % ( prefix ,
                   suffix , str ( target_url_length ))
69              print e
               # Clean the PID up after using
71              updateGWDGPid ( username , password , prefix , suffix , 'http
                   :// www . gwdg . de ')
               continue
73      sys . stdout . write ('.. DONE \n ')
       sys . stdout . flush ()
75      return results

77 if __name__ == '__main__ ':
       username = os . environ . get ('PID_TOOL_USERNAME ')
79      password = os . environ . get ('PID_TOOL_PASSWORD ')
       if username is None or password is None:
81          print "Cannot extract username and password from ENV
               variables . Exiting ..."
           exit ( -1)
83      with open ('/ tmp / measurement . csv ','a ') as outputpointer :
```

```
        output = ''
85      # write header of file
        outputpointer.write('"Numer",')
87      for exp in xrange(0, 16):
            outputpointer.write('"STL-' + str(2 ** exp) + '",')
89      outputpointer.write('\n')
        # do the measurements
91      for counter in xrange(1, 10000):
            results = measurePID(username, password, '11022', '
                testpid-' + str(counter))
93          output += str(counter) + ','
            for value in results:
95              output += str(value) + ','
            output = output[: len(output) - 2]
97          outputpointer.write(output + '\n')
            outputpointer.flush()
99          output = ''
```

A.5.4 Magnet Link Size Growth Caused By Content Signatures

Listing A.42: Script for Measuring URL-encoded Content Signatures

```
1  #!/bin/bash

3  for ((i=1;i<=10000;i++));
   do
5  openssl req -nodes -x509 -sha256 -newkey rsa:4096 -keyout "$(whoami)s
        Sign Key.key" -out "$(whoami)s Sign Key.crt" -days 365 -subj "/C=
      DE/ST=Lower Saxony/L=Goettingen/O=GWDG/OU=AG E/CN=$(whoami)s Sign
      Key"
   LENGTH=`shuf -i1-10000000 -n1`
7  echo $LENGTH
   NEW_UUID=$(cat /dev/urandom | tr -dc 'a-zA-Z0-9' | fold -w $LENGTH |
      head -n 1)
9  echo $NEW_UUID > sign.txt
   openssl dgst -sha256 -sign "$(whoami)s Sign Key.key" -out sign.txt.
      sha256 sign.txt
11 base64 sign.txt.sha256 > sign.txt.sha256.txt
   cat sign.txt.sha256.txt|perl -MURI::Escape -lne 'print uri_escape($_)
      '|wc -m >> counter.txt
13 done
```

After drawing 10.000 samples, following character counts can be observed for a size increase of Magnet Links caused by SHA256 asymmetric content signatures:

Average	736.92
Variance (population)	83.90
Standard deviation (population)	9.16
Variance (sample)	83.98
Standard deviation (sample)	9.16

Table A.3: Character Count Increase Caused By Content Signatures

Curriculum Vitae