UNIVERSITY OF AMSTERDAM

MSc Security and Network Engineering
Research Project II

Planning and Scaling a Named Data Network with Persistent Identifier Interoperability

by
Kees de Jong & Anas Younis
August 15, 2019

Supervisor:
Dr. Zhiming Zhao

Assessor:
Prof. dr. ir. Cees de Laat

Contents

Abstract

Research clouds contain diverse and large datasets, this data is published by the use of Persistent Identifiers (PIDs). The current paradigm utilizes data transmissions by the means of host-to-host connections (IP), where every data request from the consumer is answered with a data transfer from the source (the producer). This approach can potentially cause congestion and delays with many data consumers. Named Data Networking (NDN) is a data centric approach where unique data, once requested, is stored on intermediate hops in the network. Consecutive requests for that unique data object are then made available by these intermediate hops (caching). This approach distributes traffic load more efficient and reliable compared to host-to-host connection oriented techniques [69]. Furthermore, one of the most important technical challenges is to incorporate interoperability between NDN and the different PID naming schemas. These naming schemas are used by data providers within these data infrastructures for sharing and identifying digital objects. This research investigated how identification services and transmission services can be better integrated by the use of NDN. In order to create this integration, a translation between the different naming schemas was developed and made extendable for future PID types. Furthermore, a method was researched and applied for planning and scaling an NDN with scalability in mind.

1 Introduction

1.1 Background

Research clouds such as proposed in the European Open Science Cloud (EOSC) will offer Europe's 1.7 million researchers and 70 million science and technology professionals the means to store, share and re-use large volumes of information generated by the big data revolution [63]. Research clouds publish datasets from distributed sources, identified by a PID (more details in section 3). These datasets are not accessed via e.g. a simple web portal like traditional internet objects on the web. Instead these requests are processed by e.g. a data catalog of the federated research cloud. Therefore, research clouds face the challenge of identifying distributed data in a federated cloud, provide data provenance and provide a scalable service to serve many users that request large datasets.

Furthermore, research clouds face a trend of increased data production and consumption, which is expected to grow. This general trend in research clouds calls for a data distribution solution that better supports the scale and complexity. In this section we will briefly discuss the technical challenges.

An example of such a research cloud is SeaDataNet, which is a distributed marine data infrastructure network for managing the large and diverse data sets collected by the oceanographic fleets and the automatic observation systems. SeaDataNet started the SeaDataCloud project in 2016. Their aim is to advance and increase the usage of SeaDataNet's services by adopting cloud and high performance computing technology for better performance [56]. Their current infrastructure is a centralized solution (figure 1), as the current cache is a central catalogue which duplicates the entire data repository.

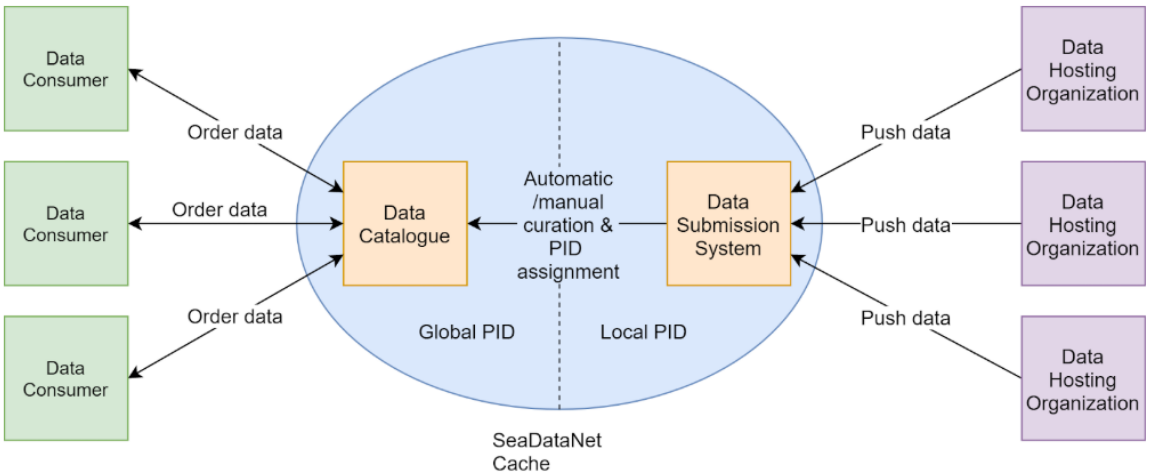

Figure 1: SeaDataCloud's current infrastructure.

Data consumers use host-to-host-connections (IP) to pull data from this cache, which could cause congestion with many concurrent data consumers. SeaDataCloud aims at considerably advancing SeaDataNet services and increasing their usage, more users are expected to be engaged and for longer sessions [57].

Another research cloud example is Cloud Storage Synchronization and Sharing Services (CS3), which is a data infrastructure that brings together users, researchers, developers, technology, and service providers of on-premise cloud services as well as large cloud service companies [36]. The

CS3 community has been growing for the last three years and currently includes around one hundred institutions and companies from around the world.

The general problem that can be subtracted from figure 1 is that research clouds face a potential congestion problem, and thus network and data processing performance degradation when many concurrent data consumers are active. In the field of big science (big data), several data distribution technologies are being experimented with, one of these technologies is NDN (more details in section 3.2). NDN could be a solution to distribute network load and data distribution more efficiently.

1.2 Research question

With the technical challenges and problem statement in mind, as described in section 1.1, we will answer the following questions which are divided into two main research questions.

1. A translation is needed between the PID and NDN namespace in order to make these solutions compatible, thus; how to make the PID and NDN namespace interoperable?

We will research different PID types to find a feasible approach to implement NDN interoperability with extensibility in mind for PID types.

- How to support different PID types?
- How to incorporate extensibility for future PID schemas?

The problem statement discussed in 1.1 calls for a data distribution solution. Therefore, we will research the planning and deployment of an NDN with scalability in mind. To achieve this, the following research question needs to be answered.

2. How to plan and manage an NDN's life cycle with scalability in mind?

To answer this research question we need to analyze the known scalability problems in NDN. Furthermore, the term scalability needs to apply to manageability as well. If scaling in or out in terms of resources is made uncomplicated, the efforts needed to manage this infrastructure needs to stay the same.

- Which NDN scaling problems are known?
- Which method can be used to plan an NDN?
- How to deploy an NDN with scalability in mind?

1.3 Scope

Based on the research question we defined the following scope. We will develop a method to make the PID types Uniform Resource Name (URN), Handle and Digital Object Identifier (DOI) interoperable with the NDN namespace. This will be done to demonstrate that interoperability of different PID types is possible within NDN. The reason we do not cover more PID types in our methodology is because with the aforementioned PID types we can already prove that PID interoperability is possible. Therefore, the interoperability goals are to demonstrate that different PID types can be used within NDN.

The planning of the NDN has the following goals. We did not have access to performance monitoring data from research clouds such as SeaDataCloud or CS3. Thus, a starting baseline will be determined based on the known performance limitations (section 2.2) and the problem statement discussed in section 1.1. Using this starting baseline, a method will be developed for NDN planning and deployment. This includes a proof of concept where, based on NDN's adaptability, the network can be adjusted for scale in both performance but also in terms of manageability. The proof of concept will be limited to existing software solutions. Therefore, solutions that are still in a research or development phase and not yet implemented or made public will not be explored in the proof of concept. The proof of concept will be setup in a small scale and is used to demonstrate the methods and train of thought used to deploy the NDN with the mentioned scalability properties. However, the methods developed should be applicable to also larger deployments (research clouds) without alternation.

1.4 Report structure

This report is structured as follows; after the background of this research is discussed, the research questions are presented. This research is divided into two research parts, each with their own scope and results. Related work will be discussed in section 2. A technical background is provided in section 3. These technical details assist in understanding our methods and technology choices we made to answer the research questions. These technologies are used in our methodology sections; 4 and 5. These methods are tested in a proof of concept. The results of both methods in the proof of concept will be summarized and the novelty evaluated in the discussion (section 6). Furthermore, we also discuss preliminary performance results based on our proof of concept in section 6.1. We will conclude our findings and provide future work in section 7. Uncommon acronyms are listed in the appendix in section 8.

2 Related work

In this section we will provide an overview of existing solutions for PID interoperability and known NDN performance and scalability issues and solutions.

2.1 PID interoperability

In 2014, Karakannas researched the efficiency of Information Centric Networking (ICN) for delivering big data with PIDs [26]. PIDs are used in big data infrastructures to identify digital content and research data. The research proposed a mapping architecture for resolving PIDs to ICN names. Karakannas proposed to use a PID to NDN mapping server for every PID type instead of implementing the translation on the clients browser. For the latter, the researcher states that translation of a PID to NDN name will be highly depended on the clients NDN browser, which needs to be updated every time a new PID schema would be introduced [26].

In 2017, Mousa researched the fetching and sharing of DOI objects with ICN such as NDN. The researcher's approach focused only on DOI identified objects within NDN's, which was possible. Mousa stated that there are many other different PID systems that can also be identified and translated for compatibility with the NDN naming schema. The patterns of most of these different PID systems are maintained in the ePIC Data Type Registry (DTR) [13]. Furthermore, the researcher explained that the difference in NDN naming of different PID providers must be taken into account, such that the correct prefix is used within NDN to identify specific PID types. For example, using the prefix `"doi/"` within NDN for DOI identified objects. In the researcher's design, the translation happens in the consumers browsers. The consumer has the choice to either request the digital object by its NDN name or PID [37].

Zhao et al. progressed the research done by Karakannas [26] and Mousa [37] and proposed an architecture to map PIDs into the naming schema of NDN. Their proposed solution is called NDN-as-a-service for PID data objects (NaaS4PID) and supports one PID type. This solution is composed out of three key components [27]:

- PID2NDN gateway; primarily responsible for resolving PIDs to NDN names.
- NDN4PID router image; an NDN node that implements a virtualized NDN router.
- NDN4PID manager; automates the management of the NDN overlay in cloud or e-infrastructure.

Olschanowsky et. al. researched translating climate modeling schema filenames, used by Center for Multiscale Modeling of Atmospheric Processes (CMMAP) applications at Colorado State University, to an NDN compatible naming schema. The researchers designed an NDN name translator to support the data management needs of CMMAP. The NDN name translator incorporates a naming schema that is designed to be compatible with existing climate modeling schemas, like the Coupled Model Intercomparison Project (CMIP5). While allowing for some flexibility for project-specific requirements. They explained that NDN names for climate data should be both expressive and human readable. The advantages of long expressive names versus short easy to remember names have to be balanced. The researchers stated that every data provider should publish a data prefix which acts as a globally routable prefix within NDN covering all of the data made available by the provider. This complies with Mousa, who stated that NDN naming of different PID providers must be taken into account. The researchers consulted with CMMAP

climate scientists and confirmed that these naming rules are acceptable for their data sets and are appropriate for global distribution. However, the translation is not as simple as taking only the filename and translate it into an NDN name. To construct appropriate NDN names to uniquely identify data of a specific model, the filename to NDN translator needs information gleaned from the filesystem path, filename and metadata mined from the data itself. This information is processed by a filename to NDN name mapping schema to use it for CMMAP filenames. Therefore, some intelligence is expected in their translators. Figure 2, taken from Olschanowsky et. al., shows the translation of a CMMAP to an NDN name. In this example it is shown that after the prefix `"/coupled/control/CMMAP/"`, which identifies the provider, the NDN translated name contains additional information next to the input filename [45]. They further stated that since NDN naming is flexible and virtually any appropriate translation schema can be plugged into, it is possible to support many naming schemas as long as the name can be broken down in hierarchical components.

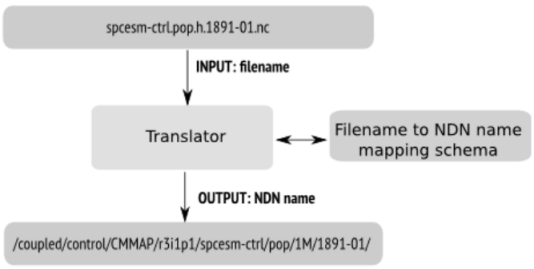

Figure 2: CMMAP to NDN name translator.

The research done by Fan et. al. utilizes a catalog which contains the translation of NDN names. Users query the catalogue to receive an NDN name. This process is fairly lightweight since the catalog's only concern is to add or remove names rather than data set synchronization. The researchers used data sets of the CMIP5 climate modeling schema for their research and encourage to use the climate modeling schema as a prefix for an NDN name. This is comparable and also complies with the previous researches, where NDN naming of different PID providers must be taken into account. Their catalog design can support the needs of any community, which can be seen as a data provider described earlier. The catalog is able to define and use consistent naming schemas. The researchers concluded that their catalog facilitates NDN name discovery and that data management can be made easier for scientific communities as long as they converge on common naming schemas [16].

2.2 NDN performance and scalability

ICN is the common architectural idea of forwarding packets based on their name rather than destination (IP) [25]. Research done by Karakannas [26] concluded that NDN was the only ICN approach that published a specification and workable software to experiment with. As of writing this report, we concluded that this conclusion is still valid. Therefore, this research used NDN for experimentation. The reasoning behind the choice for NDN compared to other solutions is discussed in more detail in section 3.2.

James McCabe's "Network Analysis, Architecture, and Design" [35] book offers a systems methodology approach towards network design. The approach is described in McCabe as "seeing the network part of a larger system, with interactions and dependencies between the network and

its users, applications and devices". The methodology put forward by McCabe is to design a network based on several inputs and outputs:

- Analysis: What do users/services require? How are the relationships composed between these components?
- Architecture: Which technology and topology choices support the users/services needs? Based on this, create high-level, end-to-end structure of the network.
- Design: Determine the physical detail of the architecture based on output of the previous steps.

Section 3.3 provides a more detailed overview of this method and section 5 describes how it is applied for this research.

NDN is not yet implemented on a large scale, with the exception of several large testbeds. In order to plan an NDN, it would be necessary to know what works and does not work. However, there is sufficient research available that focused on particular design choices of NDN. Research done by Lim et al. [33] at a large existing testbed located in the USA [43] highlighted a few lessons learned. Lim et al. setup the first intercontinental NDN testbed with the intent to see the benefits for big science. They implemented a name translator for climate-science data, which translated CMIP5 files to an NDN compatible name. They concluded that NDN provided performance improvements compared to classical climate data delivery techniques based on TCP/IP. An experimental network between universities was used for testing which provided direct links across the USA. The research concluded that NDN over UDP does not properly support large NDN packets (>64kB). According to the research this was due to UDP's properties of not applying congestion control and proper packet retransmission and resulted in unsatisfying performance. The loss of a single UDP segment resulted in the retransmission of a large NDN data packet, consisting out of many UDP packets. However, they did not consider the lack of flow and congestion control in the NDN software. The researchers also didn't take into account the theoretical limit of UDP packets which with the IPv4 protocol is 65,507 bytes (65,535 - 8 byte UDP header - 20 byte IP header), thus fragmentation is expected. Furthermore, NDN over TCP demonstrated a more reliable and faster performance due to the allowance of larger dynamic window sizes and congestion control. Native NDN congestion control is still an open research area [51], as of writing this report, no proposal has been implemented.

The performance benefits of NDN concluded by Lim et al. correlates with research done by Shannigrahi et al. at the USA-based large hadron collider network [59]. The researchers achieved a 71% reduction in the average delay per data chunk compared to a no-caching case. They conducted NDN caching simulations [58], where they concluded that a small cache of several gigabytes already reduces the network load. However, as expected, increasing the cache towards 1TB reduced the network load even further. However, this reduction of load wasn't proportional to the cache size. They concluded in their research that a 1GB NDN cache at the edge of the NDN can already significantly improve data distribution and reduce the network load. The average file size in use was 1.3GB.

Furthermore, in NDN several caching, cache replacement and forwarding strategies can be used to fine-tune performance. Koulouzis et al. researched these strategies and concluded that the 'leave copy everywhere' caching strategy provided the best performance ratio between cache size to data object size for generic data usage. However, 'leave copy down' and 'leaving copies with probability' performed best for delivering big data objects. Koulouzis et al. also concluded

that the ascending ordering of data objects enhances network performance when combined with the 'least recently used' caching strategy. This was concluded based on observations that if large objects were requested first, the cache replacement algorithm would make room by overwriting files already in the cache, resulting in cache misses. When small files were requested first, more cache hits were measured. As for cache size, the researchers recommended a cache size at least twice the size of the biggest data object in the network [27].

Yuan et al. [68] researched the performance of NDN forwarding. They used the Content-Centric Networking (CCNx) application[1] to perform experiments. Their research concluded that packets with long names degraded performance. After profiling the software they measured that the NDN name decode operation took 35.46% of the entire program running time. So et al. [60] developed a method to achieve fixed lookup times with variable length names. In order to achieve this goal they explored the application of hash tables to do name lookup. Furthermore, they explored the possibility to do this in hardware by using a Cisco ASR9000 router with the integrated service module running 64-bit Linux. With their experiments they managed to forward 20Gbps of real NDN traffic. Tortelli et al. [64] researched the effectiveness of two opposite forward strategies; flooding and best-route (with and without caching). Several experiments lead them to the conclusion that there are pros and cons in each forwarding strategy. But that it is difficult to determine the best performing forward strategy.

The heterogeneous nature in cloud environments can make deployment automation and management complicated. Topology and Orchestration Specification for Cloud Applications (TOSCA) [44] is a method to describe the deployment and management of a cloud infrastructure based on reusable templates. These descriptions are then used by orchestrators to facilitate the deployment. SeaClouds [55] (European Seventh Framework Programme (EU FP7) funded project), not to be confused with SeaDataCloud, is a research cloud that already manages infrastructures based on TOSCA. Brogi et al. [6] concluded that SeaClouds can now realize their multiple cloud infrastructures in an automated arrangement with central coordination, deployment and management without the need to make custom deployment strategies for each cloud environment. Furthermore, several large companies such as Google, Red Hat, Canonical and IBM are also involved with the development of TOSCA, signifying that broader adoption may follow in the future.

[1] `https://wiki.fd.io/view/Cicn`

3 Technical background

In this section we provide an overview of the technologies used in our methodology in sections 4 and 5. We will briefly analyze and explain related technologies based on the general problem statement in research clouds from section 1.1. We will compare key properties of these technologies and based on the analysis and the process of elimination we will define the relevant technologies for this research.

Shortly after the introduction of the WWW, Tim Berners Lee (its creator) proposed to the IETF to use the Uniform Resource Identifier (URI) as the naming schema for identifying contents on the web. This proposal got rejected by the IETF, due to the fact that it does not allow users of the web to change the URI of the web contents when moving the web contents to another location. This lead to the use of Uniform Resource Locators (URLs) as the naming schema to identify web content [26].

The use of URLs did not seem to create issues in the early stages of the web. However, several studies have found out that approximately 50% of the URLs in scholarly publications fail to retrieve the digital object after a period of seven to ten years. Web users began to experience the problem of a broken link, or the so called "link rot" problem [2, 61]. This is a situation where a user fails to retrieve the web content by its URL, due to the fact that the location of the web content has been changed [26, 30].

Therefore, the first PID systems emerged in the mid-1990s shortly after the introduction of the web itself. A PID is a long-lasting, permanent reference to digital objects. Traditional identifiers, such as the bibliographic International Standard Book Number (ISBN) will not be interpreted as a hyperlink by web browsers. For example the string `ISBN 951-45-9942-X` has to be expressed as a HTTP URI in the form of `http://urn.fi/URN:ISBN:951-45-9942-X` to be a persistent link to the resource. As shown in the persistent link, the term URN is used, which is an example of a PID type that is used for object identification. PIDs such as URN are names which do not imply a location. A URL identifies the location of a resource identified by a URN. The resource may reside in one or more locations, may move or might not be available at all. A PID needs to be mapped to a URL, which is the role of resolvers. Resolvers assist or provide persistent identifier to URL mapping [52]. In section 3.1 we will describe PID types and their resolvers in more detail. When using a PID, a user who requests the digital resource can trust that the correct digital resource is retrieved, even if the location where it resides has been changed [22].

3.1 PID types and naming schema

Over the years, different kinds of PIDs have emerged. The most widely used PID types are Handle (first major PID type introduced in 1994), DOI, Persistent Uniform Resource Locator (PURL), URN and Archival Resource Key (ARK) [22, 17, 23]. For our proof of concept described in section 4, we will describe Handle, DOI and URN. For a technical overview of the aforementioned PID types we refer to the research done by Karakannas [26].

The most widely used PID types make use of the same hierarchical naming schema, which starts with a PID type identifier such as `"urn"`, `"handle"` or `"doi"`, followed by some kind of delimiter. The PID type identifier is usually embedded in the URL of the PID resolver naming authority. Such as `"http://hdl.handle.net/"` for resolving Handle PIDs, `"https://doi.pangaea.de/"` for resolving DOI PIDs or `"http://resolver.kb.nl/resolve?urn="` for resolving URN PIDs, followed by the PID of the digital object. This means that PIDs are usually associated with a resolver via a URL [54, 26]. A PID web resolver redirects to the location of the digital object.

After the first delimiter, the naming authority is defined (this can be seen as a prefix of a digital object), followed again by some kind of delimiter. Finally there is a specific string for indicating the namespace, which is the identity of a digital object within the naming authority. Table 1 below shows an overview of the mentioned PID types.

Table 1: Hierarchical schema of PID standards [26].

PID types	PID Type ID	Delimiter	Authority	Delimiter	Name
URN	urn	:	<NID>	:	<NSS>
Handle	hdl	:	<Handle Naming Authority>	/	<Handle Local Name>
DOI	doi	:	10.<Naming Authority>	/	<doi name syntax>

3.1.1 URN

The URN PID standard was first introduced in Request for Comments (RFC) 1737 [52]. It is based on the URI syntax and its syntax is shown in figure 1. The Namespace Identifier (<NID>) part identifies the namespace or to be more specific the authority that publishes the specific URN. The syntax of the Namespace Specific String (<NSS>) part depends on the authority identified by the <NID>. This part can be used from the authority for further delegation to sub-authorities [26].

The national library of the Netherlands uses URNs. They maintain a resolver for URNs, which denote objects in a specific collection. This collection contains Algemeen Nederlands Persbureau (ANP) material. This resolver can be used to retrieve the digital object[2] in the ANP collection. The resolver translates the PID to the physical file location and forwards the requested file to the user [21]. An Registration Authority (RA) is responsible for publishing and maintaining URNs. An RA is an organization or institution which ensures specific quality standards in order to make use of URNs and is responsible for assigning URNs to digital objects. The national library of the Netherlands got assigned the prefix `URN:NBN:NL` for publishing objects. This is further delegated to a National Resolver (NR). Furthermore, an Institutional Repository (IR) deposits each assigned URN together with the URL that points to the location of the digital object in the IR with the NR [50]. IRs are digital collections, which capture and preserve the intellectual output of a single or multi-university community [67].

The national library of the Netherlands stores the metadata of a digital object in an XML schema. Descriptive metadata uses the Dublin Core (dcx) formatting, while structural metadata utilizes the MPEG21-DIDL format. The metadata can be requested via an API call[3] and is stored in the `http://services.kb.nl` data catalogue [62]. The metadata can be parsed to fill in possible naming gaps in an NDN name as discussed by Olschanowsky et. al. [45]. This method applies to all upcoming PID types discussed in this section.

3.1.2 Handle

The Handle system was developed by Bob Kahn at the Corporation for National Research Initiatives (CNRI) in 1994, and is currently administered and maintained by the Digital Object

[2] `http://resolver.kb.nl/resolve?urn=anp:1938:10:01:2:mpeg21`

[3] `http://services.kb.nl/mdo/oai?verb=GetRecord&identifier=anp:anp:1938:10:01:2:mpeg21&metadataPrefix=didl`

Numbering Authority (DONA) Foundation. The CNRI is the root server of the Handle System and maintains all the Handle naming authorities, where each Handle naming authority can establish its own resolution infrastructure. This makes it possible for the Global Handle Registry (GHR) to delegate queries for Handle resolution. Its main functionalities are specified in RFC 3650. Like uniqueness, which means that every handle is globally unique within the Handle System and persistence, which means that Handles may be used as persistent identifiers for internet resources [53]. In 2015 the Handle system supported, on average, 68 million resolution requests per month where the largest single user being the DOI system [3], which will be discussed in more detail in section 3.1.3.

The Handle syntax is shown in table 1. The <Handle Naming Authority> (HNA) part of the syntax is a prefix that it is assigned by the GHR and its hierarchical structure is similar to DNS domain names. The HNA is a sequence of decimals that are separated by the dot (".") character. For example, one can get the prefix 20.4000.581 assigned. The slash ("/") delimiter separates the HNA from the <Local Handle Name> (LHN) syntax. In this part, a PID provider can specify the identity of a digital object within the HNA it gets assigned. The only limitation for the LHN syntax is that it can only contain printable characters from Unicode's Universal Coded Character Set-2 (UCS-2). The path of a Handle PID is read from the left to the right and the dot (".") character defines the hierarchy of the naming authorities. The naming hierarchy does not imply any technical implication. Which means that the HNA "20.5000.481" can be independent of the HNA "20.5000" [26]. For Handles, a web accessed URN resolver can also be used to resolve a Handle PID to a URL[4]. Metadata is stored in the JSON format. The Handle System makes use of the restful JSON API for retrieving metadata [24]. At default when requesting the object by its PID, the metadata is served. To request the object itself (payload of the PID) one has to give up the payload parameter in the PID link. This default behaviour can be changed by serving the payload instead of the metadata. This is accomplished by changing the payload to primary in the global Handle records configuration file. The Handle system is based on a Digital Object (DO) architecture and has three core components. The first is the identifier/resolution system, which allots unique identifiers to information in digital form structured as digital objects, regardless of the location of such information. The second is the repository system, which manages digital objects. This includes the provision of access to such objects based on the use of identifiers. The third component is the registry system. This is a specialized repository system which is intended to store metadata about the digital objects stored in the repository system, rather than the digital object itself [9].

3.1.3 DOI

DOI utilizes the Handle system as described in section 3.1.2. The Handle System was selected for resolving DOI's because it matched the resolution requirements identified for the DOI concept [19], and thus DOI is based on Handle. It is managed and controlled by the International DOI Foundation (IDF). The DOI system has been assigned the <Handle Naming Authority> value 10 in the Global Handle System (GHS) as shown in table 1. Just like Handle, it uses a slash ("/") delimiter to separate the PID authority from the PID name.

The DOI system is mostly an administrative framework for assuring common practices and standards for publishing and maintaining handles between the RAs. The <doi name syntax> part identifies an object within the DOI naming authority. The DOI Resolver is the apex in the

[4] http://hdl.handle.net/20.5000.481/data/objects/object1

hierarchy and it can resolve any RA's DOI to a URL from which the digital object can be retrieved [26]. This makes the resolution of a DOI to the digital object also achievable by a web accessed resolver[5]. The PANGAEA information system, aimed at archiving, publishing and distributing georeferenced data from earth system research, uses DOI [47]. Which means that PANGAEA utilizes the Handle web resolver. Therefore, DOI also makes use of the DO architecture and objects and metadata are stored in the same way as the Handle system by using a data repository and registry system [20]. Furthermore, metadata is available in many formats in the DOI system. Including BibTeX, Citeproc-JSON, Resource Description Framework (RDF) as well as XML and JSON [49]. It depends on the PID provider which format to use.

3.2 NDN

When the internet was conceived it was designed for host-to-host communication. This means that in order to retrieve data, a host needs to retrieve the data from an IP address. Nowadays the internet is increasingly data-oriented rather than host-oriented. For example, in 2018 15% of the worldwide average bandwidth use was consumed by Netflix, with regional peaks often reaching 40% [18]. When many data consumers request the same video (or content in general), congestion can occur, degrading the user experience. Data locality sits on top of that problem, since data needs to cross distance as well, resulting in latency. Several technologies and methods have been developed to assist in efficient data distribution to improve performance. A few notable examples are Content Delivery Networks (CDNs), Information Centric Networks (ICNs) and BitTorrent.

CDNs have become a popular solution to decrease network latency [31]. In essence a Content Delivery Network (CDN) places content closer to its users. This is done by caching data in multiple geographical locations which are reachable over IP (often via anycast). These caches provide faster delivery of e.g. HTML pages, JavaScript files, style sheets, images, and videos. The key benefits of this solution are improved web content load times and increasing the availability and redundancy. Furthermore, CDNs can serve as a mitigation for Distributed Denial of Service (DDoS) attacks [7].

ICNs are another solution for data distribution [25]. However, ICNs employ a different networking methodology. An overlay is created on top of e.g. IP. In this overlay, data is identified and forwarded based on a globally unique hierarchical name rather than an IP (location). This removes the need to retrieve data from a specific IP address. The data distribution is made possible by the use of caching. Intermediary ICN nodes may cache data objects along the network path. The ICN communication model is consumer-driven, meaning that the consumer initiates queries and thus the in-network caching. ICN nodes can be of any class such as mobile phones, laptops or dedicated ICN nodes with specific hardware for increased performance. This means that once data is retrieved from the original publisher and cached in a local network, it can be shared with neighboring peers. Furthermore, due to cryptographic signatures, the data can be verified in the network and/or application layer. Thus, providing provenance authentication of the data, even if the data came from a third party. Since data is retrieved by name and not location, traditional DDoS attacks are mitigated as well as long as the requested data is available in the NDN caches.

BitTorrent shares similarities with ICN [34]. The information needed to download data from the BitTorrent network is included in a so called torrent file, which does not contain the distribution content. This torrent file contains metadata about the content that describes the file names, sizes, folder structure and cryptographic hashes to verify the integrity. This torrent file also includes a

[5] `https://doi.org/10.1594/PANGAEA.339110`

list of trackers, which are network hosts that help establish a distribution group to exchange the data of interest. The torrent file can be downloaded from a website or from e.g. a distributed hash table. However, BitTorrent tries to determine the best peers based on trail-and-error. The protocol 'chokes' and 'unchokes' peers in order to guess the best quality peers to download from. This method is applied because BitTorrent has no knowledge of the network topology and routing policies. Furthermore, data can only be verified in the application layer. This means that corrupted data is only identified and discarded when it reaches the consumer.

ICN offers the best solution based on the problem statement from section 1.1 and the analyses of data distribution related technologies. This is due to that ICN provides distribution of data by being able to forward data by name rather than IP and makes use of in-network caching. Thus, when a climate scientist downloads a large data set from his or her workstation, that data may then be cached locally. Other scientists interested in the data can then take advantage of this local cache. Thus, the original data publisher will not have to send the same data twice, lowering the chance of congestion. Furthermore, ICN has the ability to detect corrupt data in the network layer by the use of cryptographic hashes. Therefore, corrupt data isn't forwarded once corruption is detected and thus saving bandwidth. ICN is the common architectural idea which has spawned several different implementations. We decided to use the NDN [42] implementation of ICN based on related work discussed in section 2.

In an NDN packet the name can be anything; a named chunk of a video, book, command to turn on lights, or data set, which can be forwarded by any node in the network that has this named data [41]. NDN's minimal functionality includes support for consumer-driven data delivery, built-in data security, and use of in-network caching. NDN provides support for scaling data distribution, balancing data flows for congestion control and retrieving data via multiple paths. When data is sent across the NDN, it may be cached at intermediary hops. Consecutive request for the same data can then be provided by these local in-network caches.

Data is named in NDN by typically a hierarchical name, in which a prefix can be used to match a specific tree of an organization. Figure 3, taken from Jacobson et al. [25] is an example of an NDN named video file. This named data object is made available under the prefix `/parc.com`, which is the unique globally-routable name. This globally-routable name is in practice assigned by an organization (operations similar to DNS allocation of today). All data under this prefix are automatically published and available in NDN. These can then be downloaded from the original publisher, a repository, a router cache or a neighboring peer in a local network. NDN can be used as an overlay on any type of network e.g. IP, but also Bluetooth. All content can be authenticated by the use of integrity checks to ensure untainted copies of the data. Efficient distribution is achieved by caching at intermediary hops in the NDN. These hops can be NDN routers, but also cellphones and laptops. This distributed nature of NDN provides parallel transfers such as bulk data distribution of climate change data or a movie.

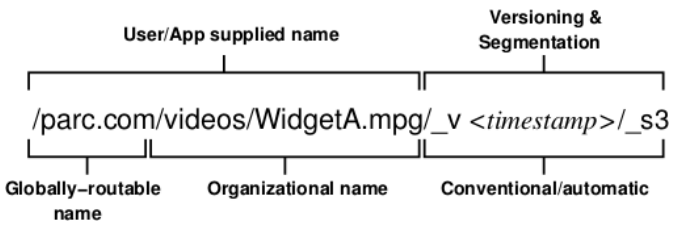

Figure 3: NDN name schema example.

Two distinct packets are used to drive communication; interest and data packets. In order to

query for data names in the NDN, the interest packet is used by the consumer. When this interest packet is received by a node in the NDN that has the data (producer), a data packet is returned. This data packets is send back over the same route as the interest packet was sent, resulting in symmetric forwarding. As discussed earlier, data may be cached on the intermediary hops in the NDN.

NDN has an advantages over IP, it is able to use multiple paths and unlike IP it is able to handle loops at the forwarding layer. A loop-free topology is realized by inserting a nonce in every interest packet which allows to identify duplicate interests for named data, allowing multiple paths to be used which could increase throughput efficiency. Information about paths in the network are maintained in a layer called the strategy layer. This layer keeps track of two-way traffic and changes local forwarding decisions based on traffic observations. The term 'face' is used in NDN to describe a connection to a forwarder, which may be of any class of the underlay (e.g. IP, Bluetooth or any other supported underlay).

The NDN packet forwarding engine is composed out of three data structures; The Forwarding Information Base (FIB), Content Store (CS) and Pending Interest Table (PIT). The FIB is used to forward interest packets towards potential source(s) of matching data. The FIB allows a list of outgoing faces towards a single prefix. In contrast to an IP FIB, a list for a single destination is not allowed due to network loop problems. The behavior of the forwarding can be changed by using different forwarding strategies. Before an interest packet is send out to the FIB it is first stored in the PIT along with the requesting face. An interest packet will remain in the PIT until a user defined timeout is reached, or when a matching data packet has returned from a source. The prefix and requesting face in the PIT define a return path back to the requester at each hop, this is referred to as the breadcrumbs path. The CS acts as the cache in NDN, which is used to provide the distributed property in NDN. This cache also has strategies, one for evicting data objects from the cache and one for stashing data objects in the cache.

3.2.1 NDN strategies

As described in section 3.2 there are two kinds of caching strategies; a caching decision and a cache replacement strategy. A caching decision strategy is used to determine in which router along the reverse path of an interest packet will cache the data [27]. The following caching decision strategies are the most well-known.

- Leave copy everywhere; used to cache data object packets along each hop in the NDN path.
- Leave copy down; used only by the first router in the NDN path after either the original producer or an NDN cache.
- Leaving copies with probability; used to cache data objects with the probability of 1/(hop count).
- Leaving copies with uniform probability; used to cache data objects with uniform probability.

A cache replacement strategy is used to determine which data objects to evict from the cache in order to make room for new data objects. The following cache eviction strategies are the most well-known.

- First in first out; used to evict the first data object that was inserted from the cache.

- Random replacement; used to evict random data objects from the cache.
- Least recently used; used to evict the least recently used data object from the cache.
- Least frequently used; used to evict the least frequently used data object from the cache.

The information stored in the PIT and FIB can be used to determine how to forward an interest packet out to one or more faces. These strategies are meant to give adaptive decisions based on network conditions. The following forward strategies are the most well-known.

- Flooding; used to forward received interest packets towards every face, excluding the face the interest originated from.
- Best-route with caching; used to calculate the best path with Dijkstra's algorithm (least amount of hops) to reach a data object cache.
- Best-route without caching; used to only satisfy interest packets towards the original data publishers, thus excluding intermediary caches along the NDN path.

3.3 McCabe

As briefly described in section 2, McCabe's book "Network Analysis, Architecture, and Design" [35] is about applying a systems methodology approach towards network design. McCabe's approach consists out of three core phases; analysis, architecture and design. These phases in McCabe describe how to make technology and topology decisions in a network. These decisions are guided based on inputs for these three core phases, the initial input may be from users and/or from network metrics. Consecutive processes use the output of previous processes as input, thus these processes are interconnected. In this section we will describe McCabe's method in more detail.

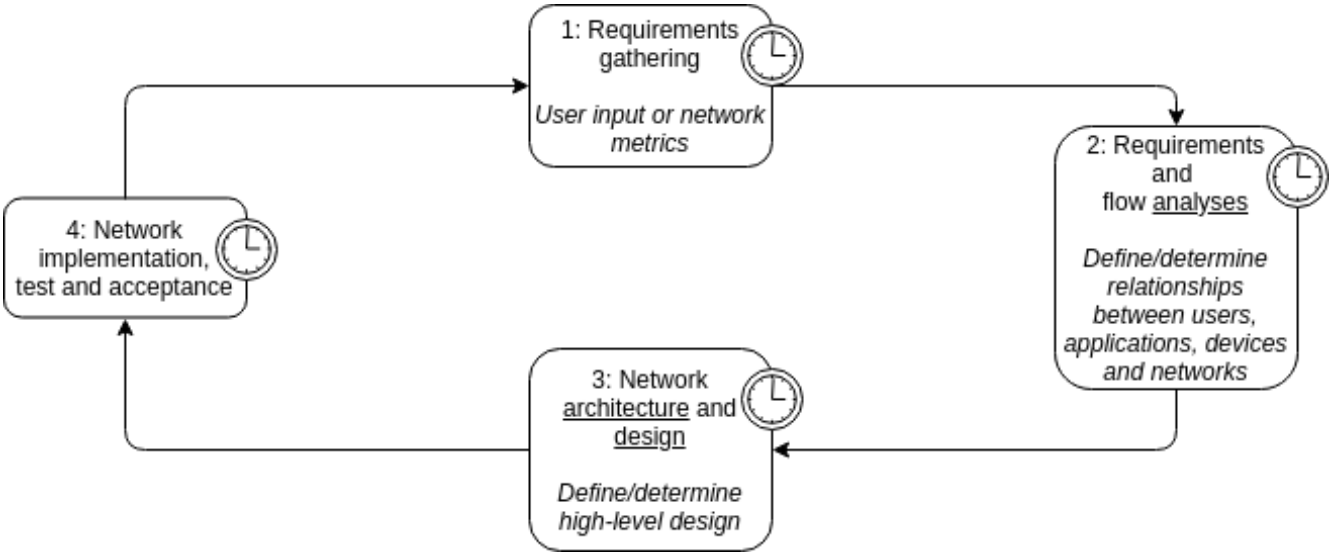

Figure 4: Cyclic and iterative nature of McCabe's processes and phases.

In figure 4 the three core phases are illustrated in *italic* while the processes are in normal text. In order to highlight the core phases by name they are underlined in the figure. The first phase

(analysis) has as goal to understand the network and potential problems in terms of performance and efficiency in order to determine network requirements. This is done by developing sets of problem statements and objectives that describe what the target network should address. Therefore, historical data from network management (monitoring), requirements gathered from the network users, staff and management are included in the analysis phase. Furthermore, these metrics are then compared to the relationships between users, applications, devices and other networks in order to determine if requirements match with the user and network expectations. The second and third phase (architecture and design) uses the output of the first phase to establish a high-level design of the network. This network design determines which technology and topology choices are justified to improve the network requirements established in the first phase. The fourth process is implementing the design, test if requirements are met and finally accept the implementation. These phases are intended to be iterative and by no means define a final architecture design. This is due to the fact that requirements, technology and user behavior can change and with that the network design.

3.4 TOSCA

Organization for the Advancement of Structured Information Standards (OASIS), an organization which is a global nonprofit consortium that works on open-standards, published the first TOSCA standard in 2014. The TOSCA standard is meant to standardize data modeling for cloud orchestration environments and applications. The latest version 1.1 [44], was released in 2018. In this section we will highlight the important features of TOSCA, relevant to our research scope. Section 5.3 will merge TOSCA with McCabe's method (section 3.3).

Portability and automated management of enterprise IT infrastructures are major concerns. Infrastructures may need to run on heterogeneous (cloud) components due to application needs. Having different descriptions of deployment and management for each environment complicates a cloud infrastructure. This challenge introduces new requirements and concepts for deployment, configuration, operation and termination of components that make up an entire infrastructure.

TOSCA provides a single description for a cloud infrastructure consisting out of templates, which is then implemented by an orchestrator. TOSCA orchestrators are e.g. Dynamic Real-time Infrastructure Planner (DRIP) and OpenStack. DRIP is currently a prototype which uses the open cloud computing interface and currently supports EC2 [4], European Grid Infrastructure (EGI) FedCloud [12] and Exo Global Environment for Network Innovation (ExoGENI) [14] clouds. Furthermore, DRIP also supports Ansible playbooks [5] for configuration management and contains a deployment agent for e.g. Kubernetes (discussed in section 3.5). This results into a portable (can be run by any orchestrator that understands TOSCA) and automated (implementation carried out by an orchestrator) method of infrastructure management. This allows interoperability and reusability of TOSCA template descriptions on different cloud providers. Reusability of templates is made possible by using variables to substitute e.g. IPs or hostnames. These variables can be retrieved with the 'get_attribute' function and can be declared by a reference node or relationship template [65].

TOSCA template descriptions consist out of the following core components; nodes, relationships and interfaces. Nodes can be a host, container or VM and are connected to each other through relationships such as 'dependsOn', 'hostedOn' and 'connectsTo'. These relationships can be used to describe that a VM is 'hosted on' a host (e.g. a bare metal machine). Or that a set of containers 'depend on' each other for functionality and 'connect to' e.g. a database. Such as a containerized web application that requires a database, facilitated by another container. Interfaces

are used to control the life cycle of a component and consist as a set of hooks to trigger actions, these actions are create, configure, start, stop or delete. These hooks can be triggered to e.g. configure and create containers, stop or start a service or do system maintenance such as delete artifacts after a service is stopped. Furthermore, constraints can be set for the input values in these template descriptions. These can be e.g. that the amount of CPU's should be defined with integers and should contain a value more than one. However, the orchestrator is responsible for verifying these constraints, TOSCA is merely a means to describe components in an infrastructure.

3.5 Virtualization

Virtualization provides more efficient use of resources and flexibility because multiple VMs and containers can be deployed on any host that supports them. VMs are virtual operating systems that run in complete isolation from its host. Containers on the other hand share the kernel of its host and offer user-space isolation, which is relatively less isolation when compared to a VM. However, containers are more resource friendly than VMs in terms of needed compute power, memory and disk space. This section will briefly describe these two forms of virtualization. Section 5 further applies these virtualization techniques.

Cloud providers such as Amazon provide easy access to compute resources, one way to access these resources is by deploying a VM on their platform. VMs and its resources can be added or removed via management interfaces or APIs. This allows an infrastructure, that e.g. runs on the Amazon cloud platform, to flexibly scale in or out.

Containers offer the ability to package software components together with all dependencies (e.g. libraries and binaries), such a package is called a container image. Docker is a popular platform to build and manage container-based virtualization. Once a Docker image is built it can be distributed through so-called Docker registries, such as Docker Hub [10]. An important property of containers is that they are not persistent, this means that containers revert back to their original image state and that data created inside the container is lost when the container has stopped or crashed. In order to make containers persistent, a data volume can be mapped onto the container's host, thus storing data outside the container.

Kubernetes [29] can be used to automate container deployment, scaling, and management of containers. In Kubernetes a container is called a pod, this may be a Docker container. These Kubernetes pods can be deployed in a cluster of Kubernetes nodes, spanning over e.g. different data centers. This offers great flexibility by packaging an application in a pod and then scale the application in or out through different data centers with Kubernetes. Kubernetes also offers high-availability features, such as automatically restart a pod when a crash occurs or load-balance requests by using a pool of identical pods in e.g. a round-robin fashion. This can be done by configuring a logical set of pods as a service. In order to custom tailor pods, environment variables can be used. These environment variables are then available inside the pod's namespace. Scripts that are executed inside the pod can then use these environment variables to setup e.g. a network configuration by specifying routes and a gateway. In Docker these scripts can be executed via a so called `ENTRYPOINT` [11].

Network Functions Virtualization (NFV) is a network architecture concept which uses virtualization to create and manage higher level network functions, as software, on commodity hardware. These functions may be interconnected to create a network service such as load-balancers, firewalls or intrusion detection. This architecture differs from traditional network architectures, where network functions are provided by hardware devices. NFV provides the ability to easily duplicate network functionality and expand the network locally or into other data centers. NFV is usually

managed by an orchestrator to automate deployment, thus providing less overhead than traditional network management with hardware devices.

4 PID interoperability with NDN

In this section, we will propose a design, which achieves PID interoperability with the NDN namespace and makes it feasible to add future PID types to answer our research question in section 1.2. Our design is based on related work done by Karakannas [26], by avoiding the PID to NDN translation on client side [26]. The pattern matching method was based on related work by Mousa for identifying different PID type schemas [37]. The research done by Olschanowsky et. al. was used for deriving NDN names from metadata [16]. We have reimplemented the principles of NaaS4PID in order to demonstrate the PID interoperability extensibility feature. We came up with our own solution, as the NaaS4PID was nor available or public to extend it. We combined the aforementioned related work and extended it with an extendable interoperability feature, we attempt to make the translation transparent to the user and support multiple PID types. The proposed design for our solution is illustrated in figure 5. Our design adheres to the following aforementioned principles, which will be discussed in more detail in this section.

- Translation is transparent to the user.
- Support for multiple PID types.
- Extensible with future PID types with different naming schemas.

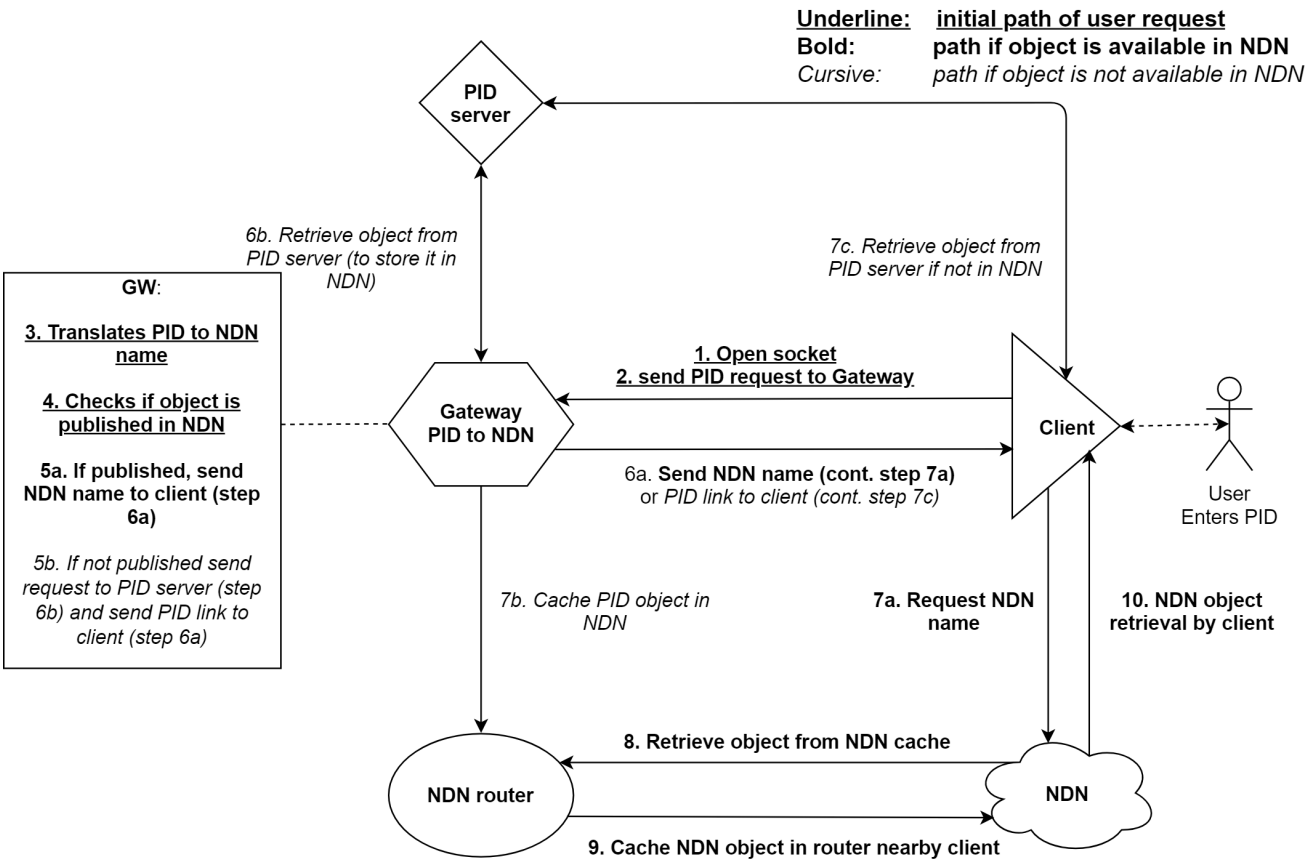

Figure 5: NDN virtual function based planning for achieving PID interoperability

Our proposed design consists of the following components, each with their own functionality; the PID server, the PID to NDN gateway and the client. The general idea is that a user enters a PID of the object that the user want to retrieve at the client and gets back the requested object as shown in figure 5. The retrieval of an object depends if the object is already published in the NDN or not, which is further described in sections 4.1.2 and 4.1.3.

4.1 Proof of concept

This section discusses the implementation of our design in a proof of concept. In our proof of concept, the components mentioned in section 4 are conceptually part of the NDN we have setup in our high-level network design, which will be discussed in section 5.2.

4.1.1 PID server

The PID server is maintained by the PID provider, which identifies objects of a particular PID type. The PID types we cover in our proof of concept are highlighted in section 3.1. For our proof of concept we have set up a Handle PID server with Cordra software [8] on our lab environment server 'nimes' to store and identify our data. We got allotted the Handle prefix `20.5000.481` by the Handle registry. We use this prefix for object identification. The Handles stored on our Handle PID server are resolved by the Handle System (`http://hdl.handle.net`). In addition to this, we also used the resolver of the national library of the Netherlands, which contains ANP material for resolving URNs as well as PANGAEA, which resolves DOIs.

4.1.2 Client

The role of the client is to transparently make the user either retrieve the requested object from the corresponding PID provider or from the NDN. This will be further described in section 4.1.3. The client receives the PID based on the user's input, which can be any kind of PID type. The client then opens a socket and sends the user's request to the gateway, which corresponds with step 1 and 2 in figure 5. After the gateway does the translation, it send back either a translated NDN name from the PID or a link to the PID server to the client. This depends whether the object is published in the NDN or not. If it is, the client receives the NDN name. If not, it receives the link to the PID server for retrieving the object. This corresponds to step 6a in figure 5.

If an NDN name is sent back to the client, the client will request the object from the NDN[6]. This is shown as step 7a and step 8 till 10 in figure 5. For object retrieval in NDN we used the tool `ndncatchunks` part of the `ndn-tools` software, which makes use of the libraries of the NDN C++ library with eXperimental eXtensions (NDN-CXX) application [40]. If the object is not in NDN and a PID link is sent back, the client requests the object by its PID link by sending the request to the PID server[7]. This corresponds to step 7c in figure 5. Object retrieval is done at client side, otherwise the gateway has to retrieve the object first before sending it to the client. The client would have to wait for the gateway till it has retrieved and cached the object in NDN. In the case that the object is already published in NDN, the gateway server only concerns itself with translation and sending back the NDN name to the client. This eliminates unnecessary load on the gateway.

[6] `https://github.com/AquaL1te/rp2/blob/master/Scripts/ndn_client.py`
[7] `https://github.com/AquaL1te/rp2/blob/master/Scripts/pid_client.py`

Retrieving an object either from the PID server or from NDN should happen transparently for the user. It is possible to accomplish this by combining the code for retrieving the object from the PID server with the code we have used to retrieve the object from NDN at the client side. Furthermore, a conditional statement needs to be added at the gateway to check if the object is already published in NDN. This is further described in section 4.1.3.

As a result, if transparency is implemented the user might not even be aware of an NDN. The user only specifies a PID as input for the client, without specifying the link of the web resolver that resolves the PID. The user gets redirected automatically by the client to either the PID server or NDN. No further user input is needed as everything is taken care of by the gateway, which will also be discussed in section 4.1.3.

4.1.3 Gateway

The gateway used in our design follows the principles of Karakannas by avoid doing the PID to NDN translation on client side. In the case of doing the translation on client side, the client software needs to be updated every time a new PID type is introduced. For our proof of concept, we designed a translation server called the PID to NDN gateway. The PID to NDN gateway implements the translation of different PID types and sends the translated name back to the client. Furthermore, we identify PID types based on pattern matching as described by Mousa [37]. We have also looked at how the Names to Things (N2T) resolver deals with different PID types. The N2T resolves different PID types by stating the PID type that needs to be implemented along with the pattern of the PIDs' schema [39].

The gateway is responsible for translating a PID to NDN name and checks if the requested object is already published in NDN. The first responsibility is translating the PID it receives from the client to an NDN name, which can be of any PID type. In our proof of concept we have implemented the Handle PID type schema of the Handle PID server we setup. In addition to this, we have also implemented the URN PID type schema of the national library of the Netherlands, as well as the DOI type schema of PANGAEA. The gateway receives a PID from the client, without the link of the corresponding web resolver of the PID type. Based on pattern matching of the PID type schema[8], the gateway detects what kind of PID type it has to deal with. Then, the associated function is called to translate the PID to NDN name[9] and appends the corresponding link of the web resolver of the PID type it receives. The patterns of most standardized PID type schemas are maintained in the ePIC DTR [13] and can be used for implementation in the gateway we propose.

Furthermore, the second responsibility of the gateway is to check if the object is already published in NDN. If the object is available in NDN, the gateway sends the translated NDN name back to the client. The client then retrieves the object from NDN. This is shown in figure 7, where the Handle PID type is used as an example. If the object is not available in NDN, the gateway sends back the PID link to the client and caches the object in NDN. The PID link that is sent back to the client contains the PID and the link to the corresponding PID web resolver. This happens before the gateway retrieves the object to cache it in NDN, otherwise the client needs to wait for this as discussed in section 4.1.2. For our proof of concept we use the tool `ndnputchunks`, which is part of the `ndn-tools` software for caching objects in the NDN [40]. Transparency can be achieved, if the code we used in our setup to retrieve the object from NDN is combined with the code we used to retrieve the object from the PID server. This is done by including a conditional

[8] `https://github.com/AquaL1te/rp2/blob/master/Scripts/pid_server.py#L58-L62`

[9] `https://github.com/AquaL1te/rp2/blob/master/Scripts/pid_server.py#L17-L37`

statement to check if the object is already published in NDN. This can be done with `ndnping` for example, which is also part of the `ndn-tools` testing software that we have utilized. To use `ndnping`, a `ndnpingserver` has to be setup for the objects. Another option is to catch the error, which is thrown by `ndncatchunks` if the object is not published by `ndnputchunks`. To translate the matched PID type to an NDN name, one has to take into account the schema of the PID type and the hierarchical way it has to be divided in. In our proof of concept a prefix is added before the PID name, for deriving an NDN name from each discussed PID type. Such as `/ndn/handle` for Handle objects, `/ndn/doi` for DOI objects and `/ndn/urn` for URN objects. This complies with the related work we have discussed in section 2. The delimiters of the PID types are replaced by a slash (`"/"`). In PANGAEA, specific columns and parameters can be requested to retrieve a particular part from an object[10]. The columns and parameters can also be translated to an NDN name[11] [37]. This shows that not only the delimiters are replaced to translate it into an NDN name. This is done by hierarchically dividing it in `/attrib+ndn` and appending the columns and parameters with a plus (`"+"`) (where the requested columns and parameters are separated by a comma (`","`)). Afterwards, the requested data chunk is stored in NDN with the parameter and column attributes. The next time someone wants to retrieve that part, it will be already available. Another solution would be to request the whole object and hierarchically divide the attributes in an NDN name by some kind of rules. This leads to having all chunks of a object available in NDN, so the next time a part of the object can be requested. Furthermore, the web resolver link is also stripped after translation.

The web resolver link is not used for deriving the NDN name. By excluding the web resolver link from the NDN name, duplications will not occur in NDN as the NDN name is only derived from the PID. A PID always translates to the same name in NDN this way. If the PID object is moved to another web resolver, only the link of the web resolver has to be updated in the gateway.

In listing 1, the translation of a Handle PID to an NDN name is shown, the Handle name is hierarchically divided into its PID type, authority and sub-authority.

Listing 1: Handle PID to NDN name

```
<user>@consumer-1:~/python-ndn$ python3 server_pid.py
Waiting for client
PID from connected user: 20.500.481/sub-auth/object1
PID type: Handle
NDN name from Handle: /ndn/handle/20.500.481/sub-auth/object1
```

The translation of a URN PID to an NDN name is shown in listing 2 for the ANP collection maintained by the national library of the Netherlands as discussed in section 3.1.1. The national library of the Netherlands chose to assign a PID based on the year when the object has been published. This way, objects in NDN can be hierarchically divided in year, months or days for example.

Listing 2: URN PID to NDN name

```
<user>@consumer-1:~/python-ndn$ python3 server_pid.py
Waiting for client
PID from connected user: anp:1938:10:01:2:mpeg21
PID type: URN
NDN name from URN: /ndn/urn/anp/1938/10/01/2/mpeg21
```

[10] `https://doi.pangaea.de/10.1594/PANGAEA.842227&columns=1,,2,3&filterParameterValue=Station,TARA_100`

[11] `/ndn/doi/10.1594/PANGAEA.842227/attrib+ndn+1,2,3+Station,TARA_100`

Metadata can be used if missing gaps need to be filled in for dividing the PID in the NDN hierarchy, as described by Olschanowsky et. al. [45] in section 2. Filling in missing gaps highly depends on which way the metadata is served by the providers of the different PID types as discussed in section 3.1. Thus, if metadata is used, a parser has to be implemented in the gateway. This could be a XML, JSON or an other kind of parser depending on the PID provider. In our proof of concept we have implemented a XML parser for URNs and a JSON parser for Handles.

There is no address exhaustion problem in NDN as the NDN namespace is unbounded [38]. But worth mentioning is to keep in mind that using long NDN names degrades performance with many interests as described by Yuan et al. [68] in section 2.

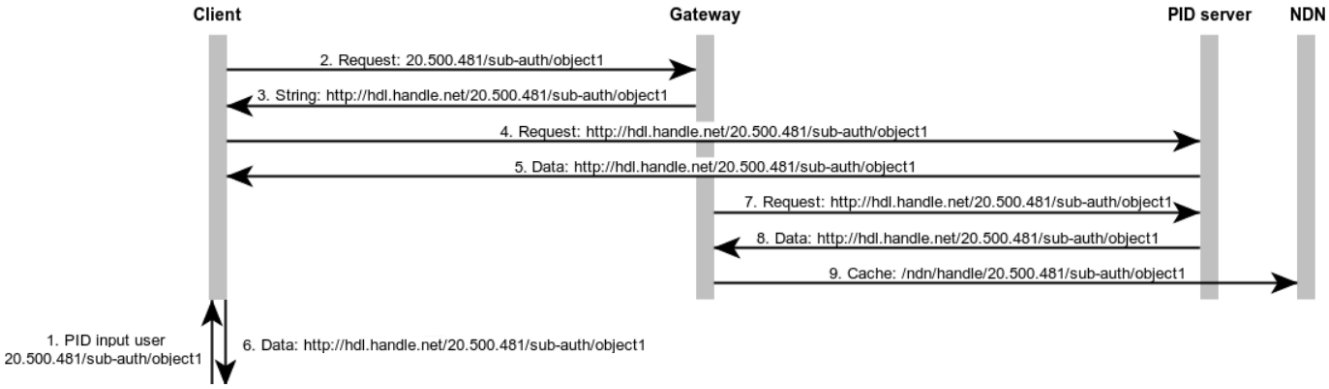

Figure 6: Handle PID request

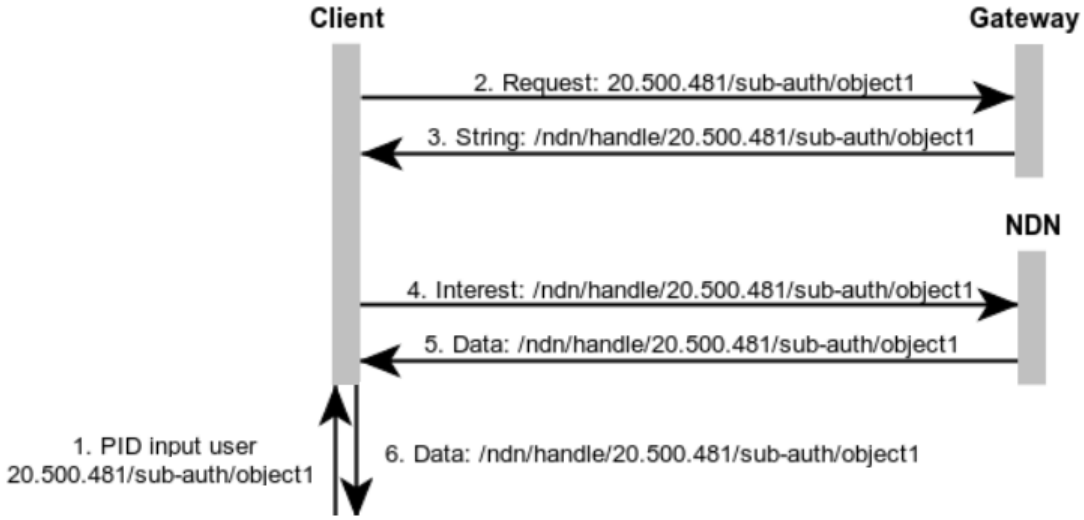

Figure 7: Handle NDN request

4.2 Results

The outcome of implementing our design in a proof of concept shows that our principles can be adhered. Since the NaaS4PID solution was not available to extend or improve it, we came up with our own solution and made it publicly available. Making the translation and object retrieval

transparent to the user is possible. Users should not have to concern themselves whether to retrieve the object through NDN or the PID server. This is due to gateway's responsibility for PID to NDN translation and the object retrieval, which is taken care of by the client. Translation is achieved by first recognizing the PID type based on pattern matching and then hierarchically divide the PID to an NDN name. Support for multiple PID types is also achieved by adding the schema of the PID types at the gateway, which makes it also easily extensible to support future PID types. By adding PID types at the gateway, we overcome the hurdle of updating the client software with the schema of newly introduced PID types each time when a new PID type is introduced.

5 Planning an NDN

This section will discuss the method we used in order to plan an NDN with scalability in mind. Scalability is defined as capacity to be changed in size or scale. In order to plan an NDN with this goal in mind we will use a combination of McCabe's method (section 3.3) and TOSCA (section 3.4). McCabe will function as a method to guide design choices with scalability and performance requirements in mind. While TOSCA will function as an implementation method that provides the means to control the whole life cycle of the network design and deployment. Furthermore, a proof of concept will be discussed that we implemented in a limited scope, in order to proof the method in practice. Furthermore, the method described in this section is deployed in a small scale proof of concept. However, the train of thought used for the NDN design, planning and management can be applied to larger scale deployments, such as research clouds.

5.1 Design requirements and analysis (McCabe)

In this section we will apply McCabe's approach in order to establish the design requirements and analyze the properties of NDN and how it can be applied to solve the problem stated from section 1.1. A requirement is flexible scalability, this will be addressed by deploying and managing NDN in an NFV-style. Therefore, NDN will be deployed as virtual functions and managed centrally via virtualization techniques as discussed in section 3.5. The first step is to make an overview of the requirements and known NDN scalability and performance issues.

The following analysis is based on the design requirements and will provide a starting baseline for the high-level network design. As discussed in the related work (section 2), several key scalability and performance metrics were addressed. Several NDN-specific design choices need to be made. These include the consideration that NDN is an overlay, on top of e.g. IP. Within this context it was concluded that TCP provided the most satisfying performance when compared to UDP. Furthermore, there are several NDN strategies to choose from. The 'leave copy everywhere' cache decision strategy and the 'least recently used' cache replacement strategy were considered to be the overall best performing choices. However, for the forward strategies there was no decisive conclusion on which a selection could be based on. Therefore, we will use the default forwarding strategy; best-route. In terms of cache size, Koulouzis et al. recommends a cache size twice the size of the largest data object residing in the used repositories. The data catalogue or registry system provides metadata of digital objects, where the size and PID type is available. This information may be used to sort the files in the catalogue by maximum size. The performance configurations mentioned can be configured in the `nfd.conf` file of the NDN-CXX application. NDN-CXX is one of the most mature software implementations of NDN and therefore used in our design. Other performance optimization solutions which were mentioned in the related work require changes in the source-code. These source-code optimizations were not made public or are not yet integrated in existing software and therefore not used.

The following recommendations are merely a baseline for research clouds and act as a starting point for performance optimizations, which is an iterative process. This process in practice should be based on the input values needed for McCabe's method discussed in section 3.3. However, these hardware configurations can be applied dynamically on cloud environments and adjusted to custom requirements. TOSCA can be used to apply these changes from a central deployment and management description.

The following hardware requirements were determined based on the software documentation, the problem statement from section 1.1 and related work from section 2. For VM memory re-

quirements, 8-12GB and 2 CPUs or more are recommended. This was determined based on the recommended system requirements for Kubernetes [28] and the fact that the use case may be I/O intensive. Therefore, I/O caching in memory benefits performance. In order to have sufficient disk space to cache NDN data objects, install software, store logs and containers, a minimum of 100GB of storage is recommended.

5.2 Architecture (McCabe)

With the design requirements established, we can develop a high-level network design. As mentioned in section 3.5, virtualization allows for flexible allocation of cloud resources via VMs while scalability of software applications can be realized by the use of containers in a NFV-style. If more cloud resources are required, then this can be done by deploying more VMs. Furthermore, if needed, VMs can be deployed in specific geographically located cloud providers, expanding the data distribution availability. The virtual NDN functions running inside these cloud providers, can then provide locally cached copies of data objects. And thus providing data distribution which lowers the chance of network congestion. In figure 8 we illustrate our high-level network design. In this illustration there are two conceptual cloud providers; 'mulhouse' and 'nimes'. These two nodes each are equipped with 12GB RAM and an Intel Xeon CPU E3-1240L v5 @ 2.10GHz with 1TB disks. In both of these conceptual cloud providers a VM will be deployed in order to allocate the resources needed.

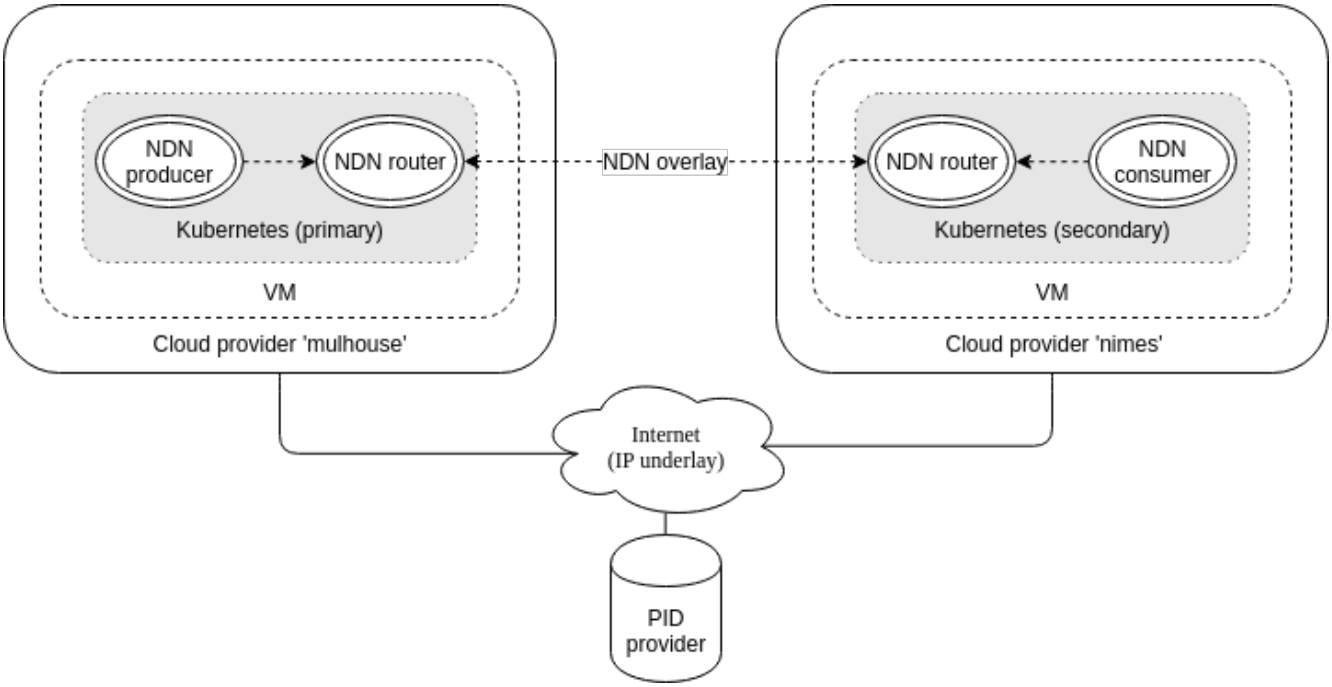

Figure 8: High-level network design.

Our high-level network design (figure 8) contains three different virtual functions for NDN nodes. The producer is assigned the function to make data available in the NDN. The consumer is assigned to request data from the producer. However, in NDN, any node that has named data, can reply to interest packets. So the producer and consumer functions can be interchangeable. The

router's function is to forward interest packets between the two cloud providers. This forwarding is done in the NDN overlay, which runs on top of the internet (underlay). NDN is designed in such a way that any node can exchange named data, thus sharing between neighbours without a router is possible. However, a dedicated NDN router offers a persistent presence in the NDN, allowing forwarding control and maintain a stable dedicated cache.

These NDN functions are deployed via a Kubernetes cluster (section 3.5), spread over the two conceptual cloud providers; ('mulhouse' and 'nimes'). These cloud providers are spread geographically in order to provide users with their regional NDN cache. Kubernetes can be used to keep a central control over the NDN, where pods (containers) can be created, removed and managed. These pods are spawned from images and custom tailored for their specific network function. This is done by the use of scripts running inside the containers, which configure the NDN based on environment variables provided by Kubernetes, as described in section 3.5. Cache misses are expensive since they require an update of the cache, which puts load on the original publisher of the data, e.g. on SeaDataCloud or any other research cloud. Which is what we want to prevent. Therefore, pods preferably are configured with persistent data volumes (section 3.5), on which the cached data can be stored outside of the container. Kubernetes can also load-balance requests between a set of identical pods. However, if these pods do not share the same persistent cached data, cache misses may occur, which results in performance degradation.

5.3 Deploying an NDN (McCabe and TOSCA)

Now that the network analysis, design and architecture are defined, a deployment strategy is needed. The high-level design (figure 8) needs to become deployable with a scalable method. Scalable in this context means that a single deployment strategy can be used for different cloud providers. As described in section 3.4, TOSCA is a standard to describe the complete life cycle of an infrastructure. Having a single set of template descriptions for deployment benefits portability and reproducibility of an infrastructure on different cloud providers.

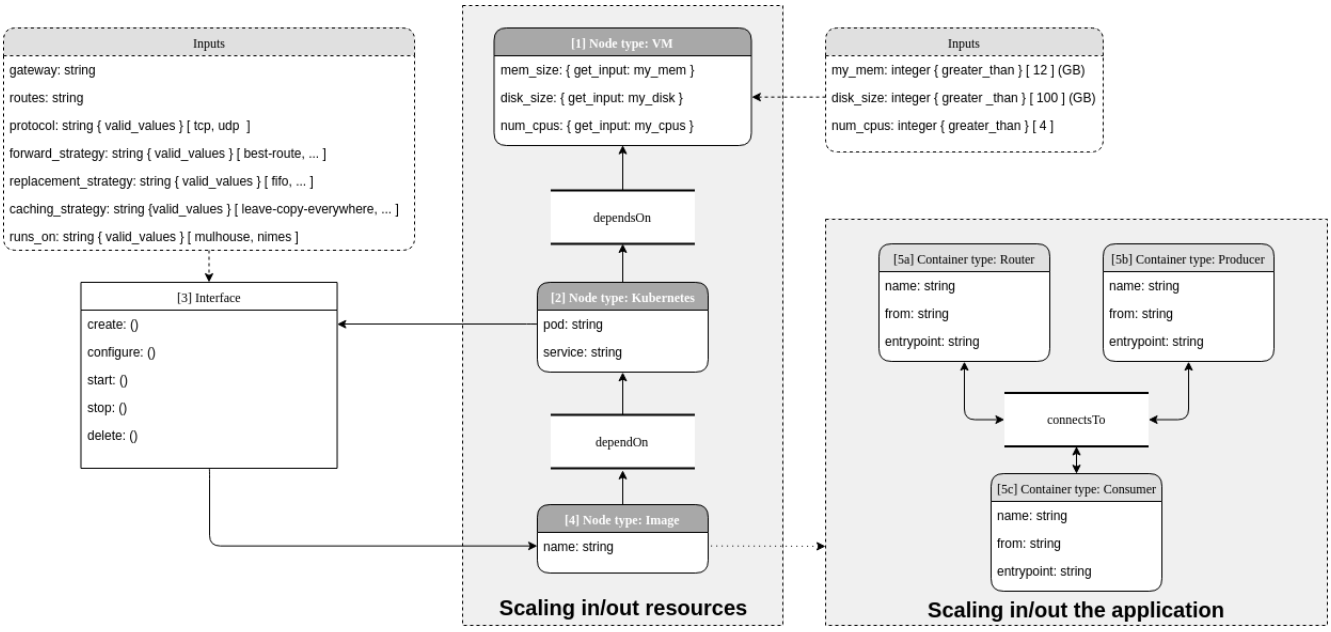

Figure 9: TOSCA diagram.

In figure 9, a TOSCA diagram is illustrated. This diagram represents an abstract template description of the TOSCA relationships, in which the grey rectangular boxes are the core scalability factors. As described in section 3.4, TOSCA consists out of several types; nodes, relationships and interfaces. The scaling properties are highlighted in the rectangular areas. The left area, highlighted as 'scaling in/out resources' contains a dependency chain of several virtual NDN functions. This dependency chain is also depicted numerically. Before a pod can be deployed on Kubernetes (step 2 to 5), a VM needs to exist (step 1). This is described by the 'dependsOn' relationship. Furthermore, with the requirements defined in section 5.1, input constraints are described. These constraints are used by the orchestrator to make sure that the NDN infrastructure has sufficient resources available to operate. Once a VM is deployed, the dependency for Kubernetes is satisfied, thus Kubernetes can then be setup (step 2). Kubernetes can then deploy pods by the use of interfaces (step 3). These interfaces feed the containers with environment variables such as the gateway, a list of routes, the transport protocol for NDN, the NDN strategies and on which Kubernetes node this pod should run. The environment variables are given to the interface via the TOSCA inputs. These environment variables are then used by scripts that run inside the pods to setup NDN. Several constraints are set for these environment variables such as which valid transport protocols can be used for NDN, which NDN strategies are valid and which nodes are available. These constraints are defined with e.g. 'valid_values' or 'greater_than' definitions. These constraints help to guide the orchestrator to verify the inputs that are given for the template description. As illustrated in the second gray area 'scaling in/out the application', several pods can be instantiated (step 5a, 5b and 5c) from the image (step 4). These pods enable the virtual NDN functions as described in section 5.2. These pods establish the NDN and therefore are connected via the 'connectsTo' relationship. This network expands over to other Kubernetes nodes in the cluster by the use of the Kubernetes built-in overlay network.

5.4 Proof of concept

With the methodology defined, in which scalability and performance requirements are met and a method for deployment is described, a proof of concept was used to test the methodology. The orchestrators mentioned in section 3.4 are still in a prototype phase when combined with a TOSCA parser. Therefore, in our proof of concept we deployed the VMs and Kubernetes nodes manually. In practice the life cycle of also the Kubernetes pods are managed by a TOSCA orchestrator. Without having a TOSCA-ready orchestrator available, steps 2 through 5 in figure 9 were be carried out by Kubernetes exclusively. This was done by defining the configuration properties[12] of the pods manually. These properties include the NDN function name, e.g. router, producer or consumer. And also includes the routes (NDN prefixes) and the associated NDN face with the transport protocol to use (TCP or UDP). These parameters were then inserted into the NDN FIB by the scripts that were executed inside the pod[13]. The NDN strategies were also configured by these scripts. Furthermore, if it is not defined where a pod should be running, Kubernetes will make this decision itself, based on the known resources in the Kubernetes cluster. If for example a Kubernetes node has more memory to spare than other nodes, then Kubernetes will likely decide to spawn the pod there. This Kubernetes node could potentially run in a cloud provider, located in another geographical area. Since the purpose is to provide data distribution through the use of NDN, locality becomes a key factor. Therefore, a pod is specifically assigned to a Kubernetes

[12] `https://github.com/AquaL1te/rp2/blob/master/Kubernetes/expanded-cluster.yml`
[13] `https://github.com/AquaL1te/rp2/blob/master/Docker/producer/docker-entrypoint.sh`

node in order to provide in-network caching in a specific geographical area.

5.5 Results

The methods discussed in this section offer a combined way to plan and deploy a data distribution network. McCabe can be used to methodologically adjust the environment to match performance requirements. For the size of this proof of concept the McCabe method is extra overhead. However, the proof of concept is only used to demonstrate the train of thought. The methods discussed can be applied to larger deployments where the McCabe method will be more valuable.

Furthermore, the NDN infrastructure life cycle can be managed from Kubernetes. Our proof of concept lacks a TOSCA-ready orchestrator. Therefore, scaling in or out resources to other cloud providers is not demonstrated. However, scaling in or out the NDN application is demonstrated. This method allows to reconfigure an NDN infrastructure in NFV-style by interacting with Kubernetes as the orchestrator. Therefore, the YAML configuration of Kubernetes acts as the TOSCA template description and Kubernetes, which executes this configuration, acts as the orchestrator. In practice the Kubernetes configuration would be generated based on the TOSCA template descriptions and e.g. DRIP would act as the orchestrator. In effect, the NDN containers are spawned as NFVs which provides scalable management of the NDN. These NDN NFVs may be deployed on a continental or global scale, facilitating a managed distribution network.

In contrast, if NDN was deployed without the discussed methods, an alternative would be to have a custom deployments for each cloud environment. Which does not scale in terms of manageability and may create inconsistent deployments and errors. Therefore, that approach also realizes a data distribution network by the use of NDN, but does not scale in terms of management and deployment.

6 Discussion and experimental results

In this section we will discuss our proposed solutions that we explored in order to answer the research questions. Furthermore, we will discuss some preliminary NDN performance of our proof of concept in section 6.1.

Research clouds make large datasets available to many users, with growth expected in the future. Lim et al. researched and proved the effectiveness of NDN for big science workflows, as discussed in section 2. However, the data distribution benefits provided by NDN would become unscalable without the means to maintain the life cycle. Therefore, our research developed and tested a method for planning and managing the NDN life cycle on a larger scale by utilizing cloud providers.

The method is a general solution for planning and deploying the NDN regardless of the research cloud its size. As demonstrated in section 5.3, we have allowed the ability to flexibly scale a data distribution network. Which, with the use of the TOSCA standard, could be deployed in different TOSCA-ready cloud providers without alteration. This flexibility allows the data distribution network to scale easily and make management uncomplicated. However, TOSCA-ready cloud providers are still rare, for our method to be more relevant, a wider adoption is needed.

The McCabe method [35] is utilized to establish the requirements and high-level design of the NDN in TOSCA. McCabe offers a proven, yet simple methodological approach for defining the design goals, which are then used to define the TOSCA descriptions.

Furthermore, in research clouds identification services are used and utilize different PID schemas. Our research created a better integration between the identification services and the data transmission services. Where traditionally IP is used for host-to-host communication, our solution utilizes NDN.

NDN requires a different namespace format than PIDs, this is due to the incompatible nature of these namespaces. Thus a translation was needed from the PID to the NDN namespace, this solution was demonstrated in section 4 and integrated in our proof of concept as discussed in section 5.

Furthermore, the focus was on developing an extendable solution for new PID schemas. Our solution succeeded by making use of regular expressions to match with a certain PID type, and then call the associated function in order to do the translation to an NDN name. Furthermore, this solution was integrated into our NDN proof of concept. We were unable to implement our solution into the existing NaaS4PID software, as this was not made publicly available.

To cache an object in our NDN proof of concept, the user is required to first request the object from the PID server. The object can then be requested in NDN afterwards. Therefore, providing transparency for the user by not requiring any further user input, has not been implemented but can be achieved as described in section 4.2. In a previous study, user input was required after translation, which is also the case in our current proof of concept. In section 4 we describe how transparency for a user can be achieved. This is accomplished due to the gateway's responsibility for PID to NDN translation and the object retrieval which is taken care of by the client. Our solutions are made freely available under the General Public License (GPLv3)[14].

An alternative, less complicated design is also possible, where the client does not have to support NDN. This is accomplished by making the gateway communicate over the NDN protocol to an NDN and IP to the client. A client that wants to retrieve a object, uses this gateway as their resolver. The gateway retrieves the object from NDN (or from the PID server if it is not already

[14] `https://github.com/AquaL1te/rp2`

in NDN) and sends this back to the client through IP. However, the gateway needs to cache the object before sending it to the client, which can cause delay for the client when retrieving an object. This design does not require the client to support NDN as it can retrieve objects through the gateway with regular web browsers, which communicate with the gateway over HTTP.

6.1 Preliminary performance measurements

In this section we will briefly discuss the preliminary performance of our NDN with PID interoperability proof of concept. The results gathered were merely based on best-effort test scenarios and are inconclusive. Therefore, further and more detailed research is required, which we will discuss in more detail in future work (section 7).

We used for both TCP/IP and NDN the default values. For TCP/IP, we did not tweak the MTU values and kept the TCP parameters to their defaults, such as the default Linux TCP congestion algorithm; Cubic. For the TCP/IP benchmarks we used the the `urllib.request` Python module, which uses the HTTP 1.1 protocol over TCP [15]. For NDN, we used the default MTU setting when creating a face (8800) with `nfdc` (part of the NDN-CXX application) and did not tweak any of the parameters, which are highlighted in section 7. For publishing objects in NDN, we used the default settings of `ndnputchunks`, which has a MTU of 4400. The underlying technology is still IP in NDN. However, NDN uses hop-by-hop fragmentation. This technology enables fragmentation at each hop if needed. Which means that the whole network no longer needs to limit its MTU size to the smallest denominator [1].

The architecture used in our experiments is as follows. The NDN consumer and router container reside on 'nimes', while the NDN producer and router reside on 'mulhouse'. Furthermore, they are interconnected via the internet, using the Kubernetes overlay network as shown in section 5.2 figure 8. As the connection is routed over the internet, latency can occur as it depends on the path a packet takes. The tool `traceroute` shows that there are no intermediate hops between 'nimes' and 'mulhouse'. The disk throughput of the server where the containers reside has a write speed of 180MB/s. Network latency is around 200 microseconds, measured by the tool `arping`. The preliminary performance results of our Handle PID server and the NDN are illustrated in boxplots (figure 10, 11, 12, 13 and 14). We ran the following performance tests within our proof of concept using a 10MB, 100MB, 250MB, 500MB and a 1000MB data object. We performed the performance tests ten times for each object size and protocol and are processed in the illustrated boxplots. The different object sizes were chosen in order to determine if there is a certain trend between the object size and performance. This performance trend is illustrated in figure 15 and shows the average of the test runs. We can observe that NDN over UDP outperforms the TCP/IP connection used for retrieving data objects from the Handle PID server that we setup with all chosen object sizes. Furthermore, NDN over TCP outperforms NDN over UDP. This result correlates with the research done by Lim et al., as discussed in section 2.2. The line chart shown in figure 15 shows that the lines converge after the 250MB mark. Which means that the relative difference between NDN and TCP/IP becomes smaller with object sizes bigger than 250MB. This is due to NDN's nature of handling big object sizes. These can cause a performance problem, because the cost of retransmission when interests are retransmitted (or re-issued) becomes unsustainably high [46].

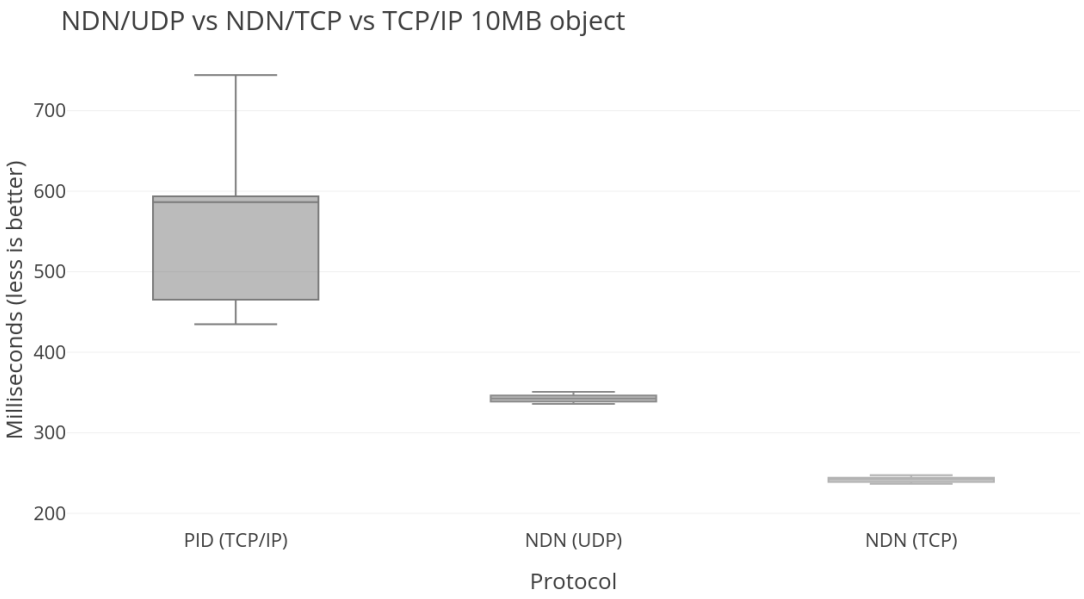

Figure 10: Performance test TCP/IP vs NDN with a 10MB object.

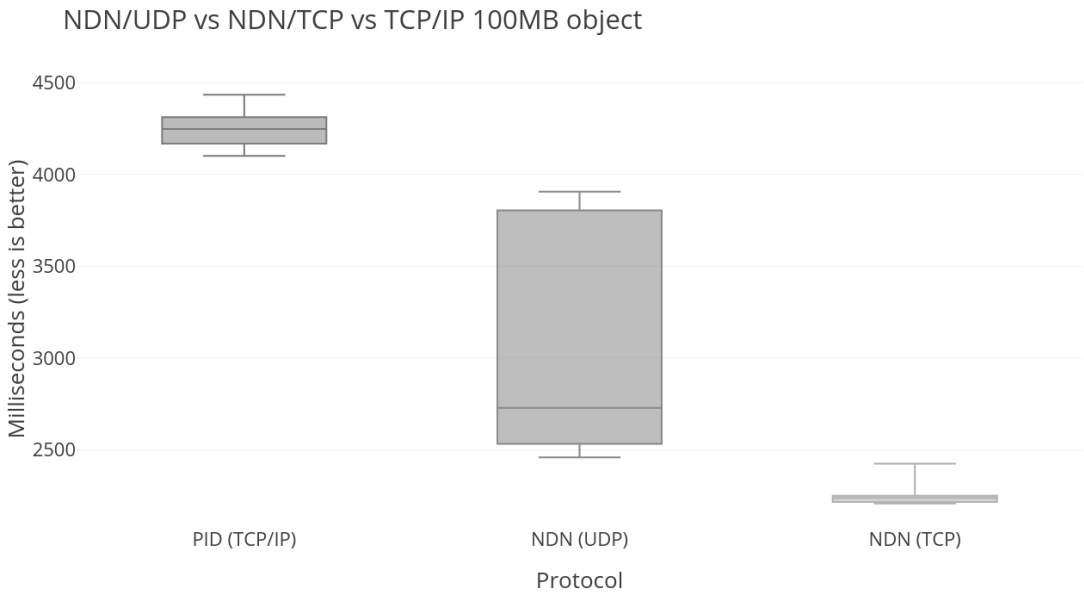

Figure 11: Performance test TCP/IP vs NDN with a 100MB object.

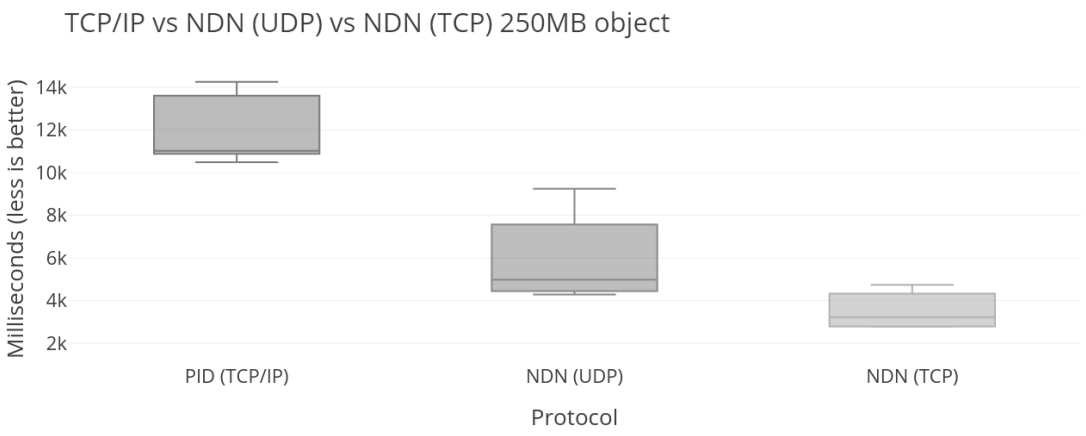

Figure 12: Performance test TCP/IP vs NDN with a 250MB object.

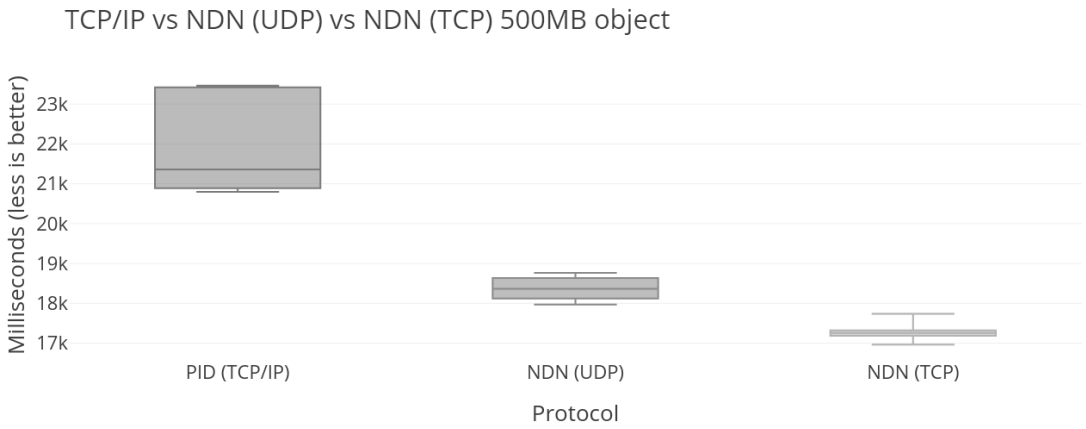

Figure 13: Performance test TCP/IP vs NDN with a 500MB object.

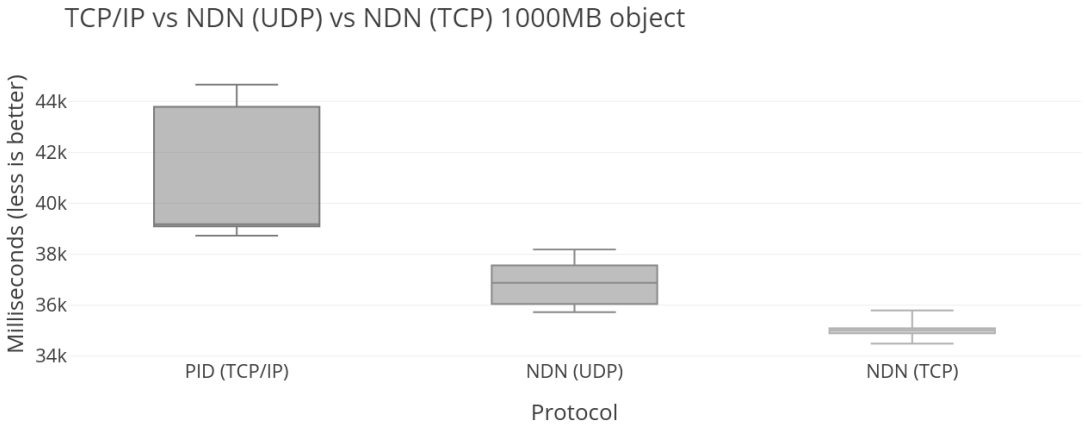

Figure 14: Performance test TCP/IP vs NDN with a 1000MB object.

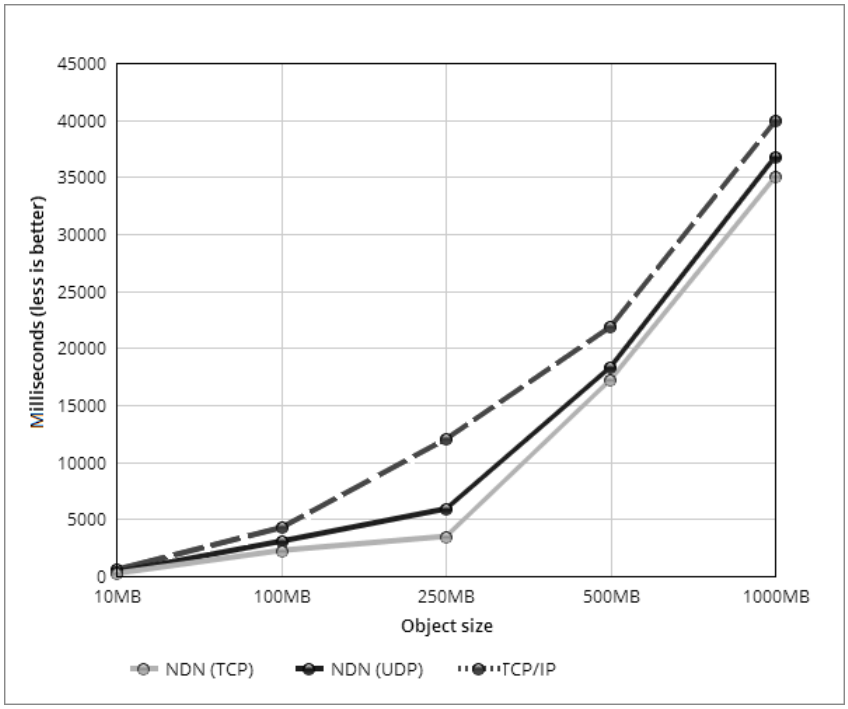

Figure 15: Object size to performance relation.

When observing the results, we can see that it takes roughly more than four seconds to retrieve a 100MB object over TCP/IP, which does not seem to be optimal. It seems that our Handle PID server, which is set up with `cordra-1.0.7` software is the bottleneck. Since we got faster results when retrieving the Linux kernel for example. We can retrieve a 102MB Linux kernel in ∼1 second within a container in our proof of concept. This seems to be faster than the default `ndnputchunks` settings with a MTU of 4400. When raising the MTU to 8800 for publishing a object in NDN with `ndnputchunks`, to reduce the number of signatures needed to transmit a object, we can observe that the average time to retrieve an object through NDN takes less than a second (∼850 milliseconds). When retrieving a object over NDN UDP with a MTU set to 8800 in `ndnputchunks`, we cannot observe any difference with our results shown in figure 11 with a MTU of 4400 over NDN UDP. This may be caused by UDP's limitations in NDN as described in section 2.2.

7 Conclusion and future work

Our research investigated solutions for the increasing trend of data producers and consumers in research clouds. Our methods are designed to be used for large research clouds. However, the methods were only tested in a small scale proof of concept. Therefore, further experimentation is needed on a larger scale to fully prove our methods. Furthermore, data producers and consumers can make use of different PID standards, in order to retrieve this data by name in NDN, a compatible translation was needed between these namespaces. The solution we developed for this problem is designed to be easily extensible for new PID types.

Research clouds are in general a federated cloud, where each federation is responsible for its own budget and infrastructure. However, our solution assumes central control over the NDN. Our solution could be approached in several ways. The solutions could be deployed as an internal data-sharing platform per infrastructure and interconnect those NDN's, thus maintaining the federated model. Or, it could be deployed as a third party data-sharing platform, where one can deploy and operate the NDN for multiple infrastructures. Our research did not explore these subjects. However, it provides the technical possibilities to deploy these solutions in such a manner.

Based on our findings we acknowledge the following subjects for further research. As described in section 3.4, TOSCA orchestrators are still in development. Therefore, more experimentation is needed when these orchestrators become more mature. This can give more insight in the flexibility and utilization of these orchestrators. As discussed in section 5.4, due to the nature of Kubernetes, pods are deployed based on the resource requirements of a pod and the resource availability inside a cluster. Pods that run NDN functions are not only interested in Kubernetes resources, but their true value is mainly determined on locality. This is due to the distributed nature of NDN, which depends on in-network caching along the network path between data producers and consumers. If Kubernetes doesn't deploy a pod where NDN resources are needed, then human intervention is needed to specify on which Kubernetes node a pod must run. Therefore, if Kubernetes could be extended with the intelligence of also NDN's resource needs, it could deploy pods automatically in a certain geographical area in order to increase cache hits. Furthermore, the NDN faces were configured manually as well, this was due to the lack of mature NDN routing protocols. However there are two promising routing protocols in development; Open Shortest Path First for NDN (OSPFN) by Lan Wang et. al. [48, 66] and Named-data Link State Routing (NLSR) [32]. With a routing protocol, the NDN management process would become less complicated and more resilient.

Furthermore, metadata can be parsed to fill in possible naming gaps in an NDN name. An NDN name is unbounded, however, the length of NDN names have to be taken into account since long NDN names can degrade performance (as discussed in section 2.2). So et al. researched and developed a solution for fixed lookup times. However, this research was not made public. Integrating such a solution could prove valuable for NDN performance. Lastly, we discussed preliminary performance measurements based on a best-effort testing scenario. Our results show that NDN over TCP gives the best performance in comparison to NDN over UDP, which correlates with related work.

Furthermore, more performance measurements and analyses is needed to gather a better profile of NDN scalability with cloud resources. The performance results we discussed in section 6 are preliminary. We recommend the following test scenarios. Performance can be improved with the use of different MTU sizes (this can be configured when creating a face, as well as with the `ndnputchunks` tool) and configuring congestion threshold and pipeline types (for example fixed, Cubic or AIMD for configuring window sizes). Where the latter can also be configured when using the `ndnputchunks` tool. Furthermore, the NDN can be configured to use TCP or UDP,

which can also be configured when creating a face. We recommend running different tests with the aforementioned parameters with multiple consumers in order to determine when congestion occurs. The parameters can also be combined with the most optimal NDN caching mechanism and the most optimal ordering method [27] to run benchmarks. These performance tests would be interesting by running it in a cloud environment, where resources can be scaled up and down.

The PID interoperability solution we developed exists outside of the NDN source code. For the best application of this functionality, we recommend the integration of these interoperability functionalities into the native NDN source code. This would remove the need to run a translation gateway.

8 Appendix

Acronyms

ANP Algemeen Nederlands Persbureau.

ARK Archival Resource Key.

CCNx Content-Centric Networking.

CDN Content Delivery Network.

CMIP5 Coupled Model Intercomparison Project.

CMMAP Center for Multiscale Modeling of Atmospheric Processes.

CNRI Corporation for National Research Initiatives.

CS Content Store.

CS3 Cloud Storage Synchronization and Sharing Services.

dcx Dublin Core.

DDoS Distributed Denial of Service.

DO Digital Object.

DOI Digital Object Identifier.

DONA Digital Object Numbering Authority.

DRIP Dynamic Real-time Infrastructure Planner.

DTR Data Type Registry.

EGI European Grid Infrastructure.

EOSC European Open Science Cloud.

EU FP7 European Seventh Framework Programme.

ExoGENI Exo Global Environment for Network Innovation.

FIB Forwarding Information Base.

GHR Global Handle Registry.

GHS Global Handle System.

GPLv3 General Public License.

ICN Information Centric Networking.

IDF International DOI Foundation.

IR Institutional Repository.

ISBN International Standard Book Number.

N2T Names to Things.

NaaS4PID NDN-as-a-service for PID data objects.

NDN Named Data Networking.

NDN-CXX NDN C++ library with eXperimental eXtensions.

NFV Network Functions Virtualization.

NLSR Named-data Link State Routing.

NR National Resolver.

OASIS Organization for the Advancement of Structured Information Standards.

OSPFN Open Shortest Path First for NDN.

PID Persistent Identifier.

PIT Pending Interest Table.

PURL Persistent Uniform Resource Locator.

RA Registration Authority.

RDF Resource Description Framework.

RFC Request for Comments.

TOSCA Topology and Orchestration Specification for Cloud Applications.

UCS Universal Coded Character Set.

URI Uniform Resource Identifier.

URL Uniform Resource Locator.

URN Uniform Resource Name.

References

[1] Alexander Afanasyev et al. "Packet Fragmentation in NDN: Why NDN Uses Hop-By-Hop Fragmentation". In: *Technical Report NDN-0032* (2015). URL: https://named-data.net/wp-content/uploads/2015/05/ndn-0032-1-ndn-memo-fragmentation.pdf (visited on 08/14/2019).

[2] Frank McCown et. al. "The Availability and Persistence of Web References in D-Lib Magazine". In: *arXiv preprint cs/0511077* (2005). URL: https://www.cs.odu.edu/~mln/pubs/iwaw-05/dlib-persistence-iwaw.pdf (visited on 07/01/2019).

[3] Joan Starr et. al. "Achieving human and machine accessibility of cited data in scholarly publications". In: *PeerJ Computer Science* 1 (2015). URL: https://peerj.com/preprints/697.pdf (visited on 07/02/2019).

[4] *Amazon EC2 website*. Amazon Web Services, Inc. URL: https://aws.amazon.com/ec2/ (visited on 07/15/2019).

[5] *Ansible website*. Red Hat, Inc. URL: https://www.ansible.com/ (visited on 07/15/2019).

[6] Antonio Brogi et al. "Adaptive management of applications across multiple clouds: The SeaClouds Approach". In: *CLEI electronic journal* 18.1 (2015), pp. 2–2.

[7] Cloudflare, Inc. *What Is a CDN?* 2019. URL: https://www.cloudflare.com/learning/cdn/what-is-a-cdn/ (visited on 07/14/2019).

[8] *Cordra*. URL: https://cordra.org (visited on 06/04/2019).

[9] *Digital Object Repository*. The DONA foundation. URL: https://www.dona.net/digital%20objectarchitecture (visited on 08/13/2019).

[10] *Docker Hub website*. Docker Inc. URL: https://hub.docker.com/ (visited on 07/15/2019).

[11] *Dockerfile reference*. Docker Inc. URL: https://docs.docker.com/engine/reference/builder/%5C#entrypoint (visited on 07/15/2019).

[12] *EGI website*. EGI. URL: https://www.egi.eu/services/cloud-compute/ (visited on 07/15/2019).

[13] *ePIC Data Type Registry*. URL: http://dtr-test.pidconsortium.eu (visited on 06/10/2019).

[14] *ExoGENI website*. ExoGENI. URL: http://www.exogeni.net/ (visited on 07/15/2019).

[15] *Extensible library for opening URLs*. Python Software Foundation. URL: https://docs.python.org/3/library/urllib.request.html (visited on 08/14/2019).

[16] Chengyu Fan, Susmit Shannigrahi, and Catherine Olschanowsky et. al. "Managing Scientific Data with Named Data Networking". In: *NDM'15* (2015). URL: https://named-data.net/wp-content/uploads/2015/11/Managing_Scientific_Data.pdf (visited on 06/18/2019).

[17] Martin Fenner, Laurel L. Haake, and Gudmundur A. Thorisson et. al. "ODIN: The ORCID and DataCite interoperability network". In: *International Journal of Knowledge and Learning (IJKL)* 9.4 (2014). URL: https://doi.org/10.1504/IJKL.2014.069537 (visited on 07/01/2019).

[18] Fortune Media IP Limited. *Netflix Consumes 15% of the World's Internet Bandwidth*. 2018. URL: https://fortune.com/2018/10/02/netflix-consumes-15-percent-of-global-internet-bandwidth/ (visited on 07/07/2019).

[19] International DOI Foundation. *DOI Resolution*. 2017. URL: https://www.doi.org/doi_handbook/3_Resolution.html (visited on 06/08/2019).

[20] *Frequently asked questions*. The DONA foundation. URL: https://www.dona.net/faq (visited on 08/13/2019).

[21] *Gebruikte standaarden bij digitalisering*. The National Library of the Netherlands. URL: https://www.kb.nl/organisatie/onderzoek-expertise/digitaliseringsprojecten-in-de-kb/beleid-documentatie-en-techniek-van-digitalisering/gebruikte-standaarden-bij-digitalisering (visited on 08/13/2019).

[22] Juha Hakala. "Persistent identifiers - an overview". In: *persid* (2010). URL: http://www.persid.org/downloads/PI-intro-2010-09-22.pdf (visited on 07/01/2019).

[23] *Handle.net Registry*. 2019. URL: http://handle.net (visited on 06/02/2019).

[24] *HDL.NET Proxy Server System*. 2018. URL: https://www.handle.net/proxy_servlet.html (visited on 06/18/2019).

[25] Van Jacobson et al. "Networking named content". In: *Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM. 2009, pp. 1–12.

[26] Andreas Karakannas and Zhiming Zhao. "Information centric networking for delivering big data with persistent identifiers". In: *University of Amsterdam* (2014).

[27] Spiros Koulouzis et al. "Information centric networking for sharing and accessing digital objects with persistent identifiers on data infrastructures". In: *2018 18th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE. 2018, pp. 661–668.

[28] *Kubernetes system requirements*. Kubernetes. URL: https://kubernetes.io/docs/setup/production-environment/tools/kubeadm/install-kubeadm/ (visited on 07/15/2019).

[29] *Kubernetes website*. Kubernetes. URL: https://kubernetes.io/ (visited on 07/15/2019).

[30] John A. Kunze. *Towards Electronic Persistence Using ARK Identifiers*. 2003. URL: https://confluence.ucop.edu/download/attachments/16744455/arkcdl.pdf (visited on 07/01/2019).

[31] Byungjoon Lee et al. "Towards a CDN over ICN." In: *DCNET/ICE-B/OPTICS*. 2012, pp. 46–51.

[32] Vince Lehman et al. "A Secure Link State Routing Protocol for NDN". In: *Technical Report NDN-0037* (2016). URL: https://named-data.net/wp-content/uploads/2016/01/ndn-0037-1-nlsr.pdf (visited on 07/07/2019).

[33] Huhnkuk Lim et al. "NDN Construction for Big Science: Lessons Learned from Establishing a Testbed". In: *IEEE Network* 32.6 (2018), pp. 124–136.

[34] Spyridon Mastorakis et al. "ntorrent: Peer-to-peer file sharing in named data networking". In: *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE. 2017, pp. 1–10.

[35] James D McCabe. *Network analysis, architecture, and design*. Elsevier, 2010.

[36] Jakub T. Moscicki and Luca Mascetti. "Cloud storage services for file synchronization and sharing in science, education and research". In: *Future Generation Computer Systems* 78 (2018), pp. 1052–1054. URL: http://dx.doi.org/10.1016/j.future.2017.09.019 (visited on 07/07/2019).

[37] Rahaf Mousa. "Application aware digital objects access and distribution using Named Data Networking". In: *University of Amsterdam* (2017).

[38] *Named Data Networking: Motivation & Details*. URL: https://named-data.net/project/archoverview (visited on 06/12/2019).

[39] *Name-to-Things resolver*. URL: https://n2t.net (visited on 06/28/2019).

[40] *NDN Essential Tools*. URL: https://github.com/named-data/ndn-tools (visited on 06/16/2019).

[41] *NDN Frequently Asked Questions (FAQ)*. National Science Foundation. URL: https://named-data.net/project/faq/ (visited on 08/13/2019).

[42] NDN Project. *Named Data Networking: Executive Summary*. 2019. URL: http://ndndemo.arl.wustl.edu/ (visited on 07/06/2019).

[43] NDN Project. *NDN Testbed Status*. 2019. URL: http://ndndemo.arl.wustl.edu/ (visited on 07/06/2019).

[44] *OASIS*. OASIS. URL: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.2/os/TOSCA-Simple-Profile-YAML-v1.2-os.pdf (visited on 07/09/2019).

[45] Catherine Olschanowsky, Susmit Shannigrahi, and Christos Papadopoulos. "Supporting climate research using named data networking". In: (2014), pp. 1–6. URL: https://named-data.net/wp-content/uploads/2015/08/supporting_climate_research_ndn.pdf (visited on 06/18/2019).

[46] D. Oran. *Maintaining CCNx or NDN flow balance with highly variable data object sizes*. 2019. URL: https://tools.ietf.org/id/draft-oran-icnrg-flowbalance-01.html (visited on 08/14/2019).

[47] *PANGAEA. Data Publisher for Earth & Environmental Science*. URL: https://www.pangaea.de (visited on 06/05/2019).

[48] *Progress Summary*. URL: https://named-data.net/project/ndn-ar2011-html (visited on 07/04/2019).

[49] *Querying DOI metadata*. URL: https://project-thor.readme.io/docs/accessing-doi-metadata (visited on 07/06/2019).

[50] *Registration Agency-NBN: Principes*. The National Library of the Netherlands. URL: https://www.kb.nl/organisatie/onderzoek-expertise/informatie-infrastructuur-diensten-voor-bibliotheken/registration-agency-nbn/principes (visited on 08/13/2019).

[51] Yongmao Ren et al. "Congestion control in named data networking–a survey". In: *Computer Communications* 86 (2016), pp. 1–11.

[52] *RFC 1737: Functional Requirements for Uniform Resource Names*. URL: https://www.ietf.org/rfc/rfc1737.txt (visited on 07/02/2019).

[53] *RFC 3650: Handle System Overview*. URL: https://www.ietf.org/rfc/rfc3650.txt (visited on 07/02/2019).

[54] Ryan Scherle. *Identifiers*. 2011. URL: https://wiki.dlib.indiana.edu/display/INF/Identifiers (visited on 06/20/2019).

[55] *SeaClouds website*. SeaClouds. URL: http://www.seaclouds-project.eu/ (visited on 07/15/2019).

[56] *SeaDataCloud (2016 - 2020)*. SeaDataCloud. URL: https://www.seadatanet.org/About-us/SeaDataCloud (visited on 06/04/2019).

[57] *SeaDataCloud - Further developing the pan-European infrastructure for marine and ocean data management*. EU Commision - Cordis. URL: https://cordis.europa.eu/project/rcn/207433/factsheet/en (visited on 08/14/2019).

[58] Susmit Shannigrahi, Chengyu Fan, and Christos Papadopoulos. "Request aggregation, caching, and forwarding strategies for improving large climate data distribution with NDN: a case study". In: *Proceedings of the 4th ACM Conference on Information-Centric Networking*. ACM. 2017, pp. 54–65.

[59] Susmit Shannigrahi et al. "Named data networking in climate research and hep applications". In: *Journal of Physics: Conference Series*. Vol. 664. 5. IOP Publishing. 2015, p. 052033.

[60] Won So et al. "Named data networking on a router: forwarding at 20gbps and beyond". In: *ACM SIGCOMM Computer Communication Review*. Vol. 43. 4. ACM. 2013, pp. 495–496.

[61] Diomidis Spinellis. "The Decay and Failures of Web References". In: *Communications of the ACM* 46.1 (2003), pp. 71–77. URL: https://www2.dmst.aueb.gr/dds/pubs/jrnl/2003-CACM-URLcite/html/urlcite.html (visited on 07/01/2019).

[62] *Technische uitleg dataset ANP*. The National Library of the Netherlands. 2012. URL: https://www.kb.nl/sites/default/files/docs/techniek-anp.pdf (visited on 06/16/2019).

[63] *The European Cloud Initiative*. European Commission. URL: https://ec.europa.eu/digital-single-market/en/%20european-cloud-initiative (visited on 08/15/2019).

[64] Michele Tortelli, Luigi Alfredo Grieco, and Gennaro Boggia. "Performance assessment of routing strategies in named data networking". In: *IEEE ICNP*. 2013.

[65] *TOSCA Attribute functions*. OASIS. URL: https://docs.oasis-open.org/tosca/TOSCA-Simple-Profile-YAML/v1.1/csprd01/TOSCA-Simple-Profile-YAML-v1.1-csprd01.html#_Toc464060429 (visited on 08/13/2019).

[66] Lan Wang et al. "OSPFN: An OSPF Based Routing Protocol for Named Data Networking". In: *Technical Report NDN-0003* (2012). URL: https://www.named-data.net/techreport/TR003-OSPFN.pdf (visited on 07/04/2019).

[67] *What is an Institutional Repository?* Centrum Wiskunde en Informatica. URL: https://www.cwi.nl/about/library/tutorial/13-what-is-an-institutional-repository (visited on 08/13/2019).

[68] Haowei Yuan, Tian Song, and Patrick Crowley. "Scalable NDN forwarding: Concepts, issues and principles". In: *2012 21st International Conference on computer communications and networks (ICCCN)*. IEEE. 2012, pp. 1–9.

[69] Lixia Zhang et al. "Named Data Networking". In: *ACM SIGCOMM Computer Communication Review* 44.3 (2014). URL: https://named-data.net/wp-content/uploads/2014/10/named_data_networking_ccr.pdf (visited on 06/03/2019).