UNIVERSITY OF AMSTERDAM

Faculty of Science

MSc System and Network Engineering

Master of Science Thesis

Application aware digital objects access and distribution using Named Data Networking (NDN)

Author:

Rahaf Mousa

Supervisor:

dr. Zhiming Zhao

Amsterdam July 14, 2017

Acknowledgements

To my thesis supervisor dr. Zhiming Zhao for his efforts guiding me in this challenging journey.

To my husband and family for their love and support.

To my colleagues who were always there to help.

Abstract

In big data infrastructures, Persistent Identifiers (PIDs) are widely used to identify digital content and research data. A typical example of PIDs is the Digital Object Identifier (DOI). In a data centric application (such as a scientific workflow) it is often required to fetch different data objects from multiple locations. When reproducing a workflow published by community, data objects involved in the workflow often have PIDs. In this project we investigated how to optimize the fetching and sharing of DOI identified objects with Information centric networking paradigm such as Named Data Networking (NDN). In order to achieve that goal, first we presented an approach for integrating PIDs with Named Data Networking (NDN) networks. NDN identifies digital objects with their names and route them also based on their names. In addition, we proposed an approach for optimizing the NDN network's performance using application level knowledge, such as the size, number, and order of the requested objects. We investigated the effect of ordering a group of objects in ascending or descending order according to their sizes before requesting them one by one. The results showed that the order of the requests can dramatically influence performance of fetching objects from NDN networks.

Contents

List of Figures

Acronyms

CCN Content Centric Networking 4, 6

CS Content Store 10, 11, 13, 14

DOI Digital Object Identifier 1, 4–6, 8, 9, 15–18, 23, 25, 26

DS Decision Strategy 13

FIB Forwarding Information Base 10, 13

FIFO First In First Out 2, 19–21, 25, 30, 32

ICN Information Centric Networking 4, 6

LFU Least Frequently Used 2, 19–23, 31

LRU Least Recently Used 2, 19–21, 25, 31, 33

NDN Named Data Networking 1, 2, 4–15, 17–19, 23–26

PID Persistent Identifier 1, 4, 15

PIT Pending Interest Table 10, 11, 13

RR Random Replacement 2, 19–21, 25, 32

RS Replacement Strategy 13, 14

1. Introduction

In big data infrastructures (such as the research infrastructures in ENVRIplus [1]), research data objects often have a PID and can be directly retrieved using its PID. A PID is an enduring reference to a document, file, web page, or other object[1]. An advantage of using PIDs is providing a persistent way of retrieving objects, even when their location changes. A typical PID is the DOI, with the DOI of an object you can request the Handle System[2] for the metadata of that object which contain the location (url) where the object is currently available.
A data centric application (such as a scientific workflow) often requires fetching different data objects from multiple locations. An example of that is when a researcher wants to reproduce the results of a scientific paper. In this case he/she would need to retrieve the cited references mentioned in the scientific paper, together with the data involved in the experiments. Optimizing the access of multiple data objects in such cases is crucial for the system's performance. The problem that the current big data infrastructures face, is that its demand for distributing large content and sharing data among large communities does not fit easily in the Internet architecture as it is today. Looking at the way the Internet was first implemented, we see that it was for creating point-to-point communications between two ends, while a big data infrastructure is a content distribution infrastructure.

A suitable solution for big data infrastructure can be Information Centric Networking (ICN). "ICN is an approach to evolve the Internet infrastructure away from a host-centric paradigm based on perpetual connectivity and the end-to-end principle, to a network architecture in which the focal point is "named information""[3]. In this project, we will investigate how to use ICN approach to enhance the distribution of digital objects. We will specifically focus on NDN [4], the successor of the original Content Centric Networking (CCN) [5]. The main idea behind NDN is identifying digital objects with persistent identifiers that is independent of its location, unlike with IP where data is identified with its location. Digital objects are routed in NDN based on its identifier. While the current Internet secures the data container (i.e., the communication channel), NDN assures securing the contents, that dissociates trust in data from trust in hosts. It also enables several radically salable communication mechanisms such as automatic caching to optimize bandwidth[6]. The NDN project was funded by the National Science Foundation (NSF) in September 2010 as one of the four projects under NSF's Future Internet Architecture Program[6]. NDN have gained a lot of attention in the research

[1]http://www.envriplus.eu

community with around four hundred papers about it. We propose NDN as a solution for big data infrastructures for several reasons, such as that security is built in data itself by signing each piece of data together with its name. Additionally, NDN provides in-network caching, meaning that objects are cached in the network level (in the routers). Such in-network caching can decrease the amount of traffic and increase the response's speed. These characteristics facilitate achieving the speed, availability, and security required in a big data infrastructure.

In order to efficiently utilize NDN environments in big data infrastructures, DOI identified data objects have to be imported and efficiently distributed using NDN mechanisms. In this context, we highlight two research questions:

1. How can we facilitate the fetching of DOI identified objects via NDN network?
2. How can we optimize NDN network's performance using application level knowledge?

In order to answer the second question, we specifically focus on how to optimize the performance by using application level information, such as the number of the requested objects, their sizes and order.

This report is constructed in the following way: first we will discuss the scientific work related to our research. Then we will explain the basic concepts this report deals with. Following, we will present the two proposed approaches in addition to the experiments we conducted for evaluating the second approach. Lastly, we will present a discussion of our findings, the conclusion of this work and the suggested future work.

2. Related Work

The growth in the number of science and engineering applications that involve big data and intensive computing is introducing higher demand for network performance. That have pushed researchers for finding new solutions for meet these demands. Among the existing work, several research focuses can be highlighted: adopting ICN as a solution[7][8], enhancing data distribution performance with different big data infrastructures[7], and Integrating these proposed approaches with the services widely used in the scientific field[9].
Chen *et.al.* reviewed different Data Center Networking technologies for future cloud infrastructures to efficiently manage the huge amount of resources in data centers[7]. In their paper, they proposed leveraging the existing NDN technology for distributed file system (DFS) management and VM migration.
Tantatsanawong *et.al.* presented an approach for using different future internet models, such as CCN and Software define Network (SDN) to improve the Hadoop based Big data architecture on research and education networks (REN) to deliver high-performance and scalable big data analytic[8].

Schmitt, Majchrzak and Bingert presented a Persistent Identifier infrastructure stack for NDN. The stack provides fetching mechanisms for objects from the handle system in general (not only DOI identified objects as the case is in our approach). An NDN-enabled Handle server is proposed in the NDN network, such server can react and send the specific Handle value as NDN data packet response. Moreover, Handle gateways are included in the NDN network to forward resolution requests wrapped in interests to existing Handle servers located in the classic network environment[9]. Compared to this approach, we approach the problem in a much simpler way, by adding a script to the consumer's browser that is running on both IP and NDN networks. That script sends requests to the Handle System in order to fetch DOI identified objects, and publishes them in the NDN network for future possible requests.

In-network caching in novel proposed Internet architectures was tested and evaluated by Jing. In his paper, he looked into in-network caching in two architectures: NDN and MobilityFirst. The results of his experiment showed that LFU (Least Frequently Used) is the most effective cache replacement strategy[10].

Xiaoke Jiang and Jun Bi proposed creating an Interest Set packet for interests that share the same prefix. This approach resulted in reducing the size of Pending Interest Table (PIT), additionally, when the network traffic is heavy, it can decrease the round trip delay. Nevertheless, it has the downside of leading to inflexibility to data plane and data burst[11]. Their research was focused on optimizing the performance of application that is based on receiving bursts of packets from a specific producer, such as in live video and large file transfer applications. In our case, such approach limits objects names that can be added in the set of Interest packet to having the same prefix, while our motivation is facilitating fetching a set of objects often from different producers.

We can see that using NDN in big data infrastructure have attracted lots of research attention, however more research need to be done in order to have a generic implementation for such integration. Additionally, enhancing some features in NDN networks according to the different demands of big data infrastructures may produce an increase in performance.

3. Background

In this chapter, we give a short introduction of the key concepts used in this report. First, we will explain what DOI is and how this system works. Next, we will explain the basic concept of the Handle System, which DOI implements. Eventually, we will throw light on the fundamentals of the NDN approach.

3.1. Digital Object Identifiers

The Digital Object Identifier or DOI is a persistent identifier system that provides an infrastructure for persistent and unique identification of objects of any type. DOI is standardized by the ISO [1][12]. "Digital Object Identifier" means that it is a "digital identifier of an object" rather than an “identifier of a digital object”[13]. The DOI system implements the Handle System; a general-purpose global name service enabling secure name resolution over the Internet[14][13].
A DOI name is permanently assigned to an object and it provides a link to current information about the object. These information include the object’s metadata, which describes the content of that object and its location. While the object’s information might change, its DOI will not change.

The DOI system enables constructing automated services and transactions. Managing information and documentation location and access, managing metadata, and facilitating electronic transactions are some of the DOI system applications[13]. The DOI system is implemented by Registration Agencies, these RA’s provide domain-specific identifiers for various applications using the underlying DOI framework. For example, Crossref[2] manages DOIs for the scientific publishing industry, DataCite[3] provides DOIs for referencing and sharing scientific data sets[13].

The DOI system provides a ready-to-use packaged system which contains several components; specified standard numbering syntax, a resolution service, a data model incorporating a data dictionary, and an implementation mechanism through a social infrastructure of organizations, policies and procedures for the governance and registration of DOI names[13].

[1]The International Organization for Standardization
[2]https://www.crossref.org/
[3]https://www.datacite.org/

3.1.1. DOI name syntax

The DOI name has two components; the prefix and the suffix with a slash "/" separating them. The prefix is composed of two parts; a directory indicator and a registrant code, which are separated by a period ".". The directory indicator for DOI is "10", this way it indicated that the provided name is a DOI within the Handle System. The registrant code is a unique string assigned to a registrant. The suffix consists of a character string of any length. Assigning the suffix is left for the naming authority to decide[15].
The prefix should be globally unique, as it represents a naming authority(the registrant). On the other hand, the suffix has to be only unique within the naming authority space.
An example of a DOI is 10.1000/123456, where "10.1000" is the prefix and "123456" is the suffix. In the prefix "10" is the directory indicator that identifies the DOI system and "1000" the registrant code that identifies a naming authority.

3.2. The Handle System

The Handle System,is a distributed information system that is designed for providing extensible and secured global name service for use on networks such as the Internet[16]. It was designed by the Corporation for National Research Initiatives (CNRI)[4][16] and currently administered and maintained by the DONA Foundation [5][17]. The Handle System includes an open protocol, a namespace, and a reference implementation of the protocol. Its protocol enables a distributed computer system to store names, or handles, of digital resources and resolve those handles into the information necessary to locate, access, and otherwise make use of the resources[2]. The associated information describe the current state of the identified object and they may change, yet the handle does not change.
Each handle may have its own administrator(s) and administration can be done in a distributed environment. The Handle System provides a confederated name service that allows any existing local namespace to join the global handle namespace by obtaining a unique Handle System naming authority. Local names and their value-binding(s) does not change after joining the Handle System. Handle requests to any local space are processed by a service interface speaking the Handle system interface. Because the naming authority is unique in the handle, any local name is unique under the global handle namespace.[2]

3.3. NDN

"NDN retains the Internet's hourglass architecture, but evolves the thin waist to allow the creation of completely general distribution networks."[18] Eliminating the restriction that packets can only name communication endpoint is the most important element of this evolution. The name in a NDN packet can be anything, e.g., an endpoint, a book, a command to

[4]https://www.cnri.reston.va.us/
[5]https://www.dona.net/

turn on some lights, etc[18]. Communication in NDN is driven by the consumer. In order to re-

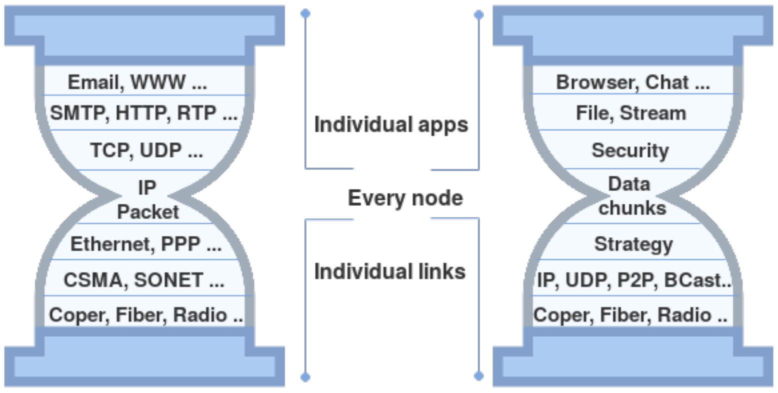


Figure 3.1: Internet and NDN Hourglass Architectures

ceive data, a consumer sends out an Interest packet, which contains the name of the requested object, for example /uva/os3/rp2/report123.pdf, along with some additional data. Figure 3.2 shows the content of an Interest packet. When the router receives the Interest packet, it looks in its Content Store (CS)[6], which contains the stored objects, if the object is found then it is sent to the consumer in a Data packet with a signature by the producer's key. The content of a Data Packet is shown in 3.2.

If the object is not found in the CS of the router, the router remembers the interface from

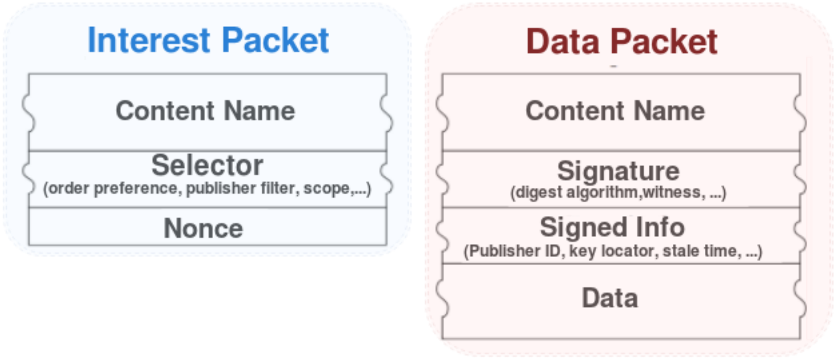

Interest Packet
Content Name
Selector
(order preference, publisher filter, scope,...)
Nonce
Data Packet
Content Name
Signature
(digest algorithm,witness, ...)
Signed Info
(Publisher ID, key locator, stale time, ...)
Data

Figure 3.2: Packets in NDN

which the request comes in, this information is stored in what is called Pending Interest Table (PIT). Then the router forwards the Interest packet by looking up the name in its Forwarding Information Base (FIB), this table is populated by a name-based routing protocol, the aforementioned process is shown in Figure 3.3(a). As long as the requested object is not found, Interest packets are forwarded to the next router till it is found in one of the routers, or the producer of the object is reached.

When the object is found, it traces back to the consumer through the reverse path taken by the Interest Packet, this process is shown in Figure 3.3(b). Note that neither Interest nor Data pack-

[6]The Content Store is basically the router's buffer memory subject to a cache replacement policy

ets carry any host or interface addresses (such as IP addresses); Interest packets are routed towards data producers based on the names carried in the Interest packets, and Data packets are returned based on the state information set up by the Interests at each router hop[19]. Both

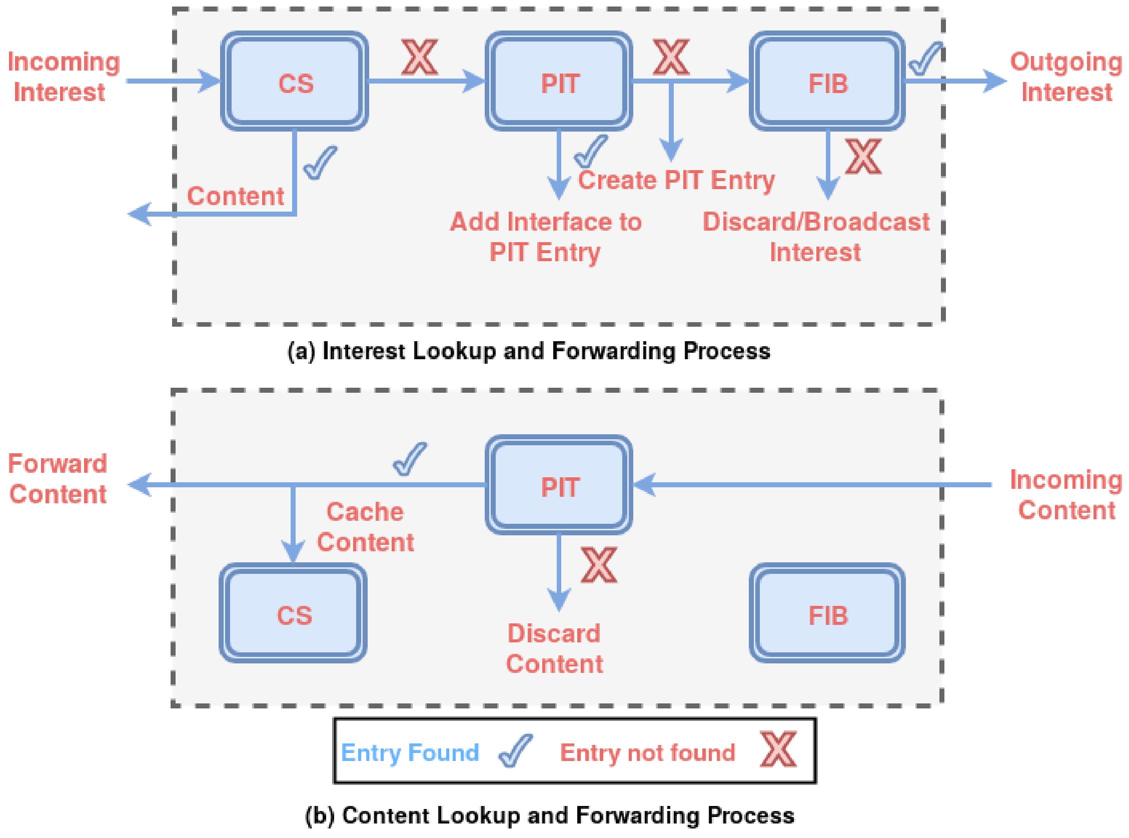


Figure 3.3: Interest and Content lookup and forwarding process in the NDN router

Interests and Data are kept in the routers for a period of time. When multiple Interests for the same Data are received from downstream, only the first Interest is sent upstream towards the data source. The router then stores the Interest in the PIT, where each entry contains the name of the Interest and a set of interfaces from which the matching Interests have been received[19], as it is shown in Figure 3.3. When the requested data arrives to the router, the router looks for a matching entry in its PIT. If a match is found, the data is forwarded through the corresponding interfaces stored in the PIT, the entry is removed, and the data is cached in the CS. If a match is not found in the PIT, then the Data packet is immediately dropped. As a consequence, traffic is pull-based, meaning that no contents flow unless a consumer has explicitly asked for them and there is a 1-to-1 matching between Interest and Data. This significantly reduces the amount of unwanted data transfers (e.g., spam) and facilitates the deployment of accountability and control mechanisms on the content routers that manage Interest signaling (e.g., aggregation mechanisms for massive data access, congestion control)[20].

Caching the objects that pass through the router allows it to satisfy potential future requests for these objects. That achieves independence from the original location of the object and reduces the round trip time for fetching cached objects. Therefore it enables NDN to automatically support various functionality without extra infrastructure, including content distribution[7], multicast, mobility, and delay-tolerant networking[19].

3.3.1. Naming

The design of NDN assumes hierarchically structured names, such as /uva/os3/rp2/report123.pdf. These names are opaque to the network, meaning that the router does not know the meaning of the name, yet, it knows the boundaries between each component, which is the slash "/". This design allows each application to choose the naming scheme that fits its needs and allows the naming schemes to evolve independently from the network[4].
This hierarchical structure is useful for applications to represent relationships between pieces of data. For example, segment 3 of version 1 of the video might be named /uva/os3/rp2/video/3. Additionally, the hierarchy allows routing to scale because it enables aggregation which is essential in scaling today's routing system.
Common structures necessary to allow programs to operate over NDN names can be achieved by conventions agreed between data producers and consumers, e.g., name conventions indicating versioning and segmentation. Name conventions are also specific to applications and opaque to networks[4].

To retrieve dynamically generated data, consumers must be able to deterministically construct the name for a desired piece of data without having previously seen the name or data. Either a deterministic algorithm allows both the producer and consumer to arrive at the same name based on data available to both, and/or consumers can retrieve data based on partial names[4]. For example a consumer can request a video file without specifying the segment, but the data that is received is the first segment, based on the name of the object received, the consumer can specify which parts of the objects to request next.

3.3.2. Data-Centric Security

Content security is crucial in any data centric network. In NDN, security is built into data itself, rather than being a function of where, or how, it is obtained[19]. Data is signed together with its name with the producer's public key. The signature, in addition to the producer's information, proves the authenticity of data. That allows the consumer to trust the data no matter where it is fetched from. It also makes data immutable, meaning that making changes on an object would create a new object, that new object would have a new signature and a change to the name would be applied, for example, by adding a version number at the end of the name. The signature field in each NDN Data packet includes a key locator field, as shown in Figure 3.2, which specifies either a name that points to another Data packet containing certificate or a public key or KeyDigest to identify the public key within a specific trust model[21]. Keys are also communicated in NDN as data, which as well contains its own key

[7]Which is the case with big data infrastructures

locator. These key locators form a certification chain up to a trust anchor key that a consumer must have obtained and trust a priori[22].
Secure binding of names to data provides a basis for a wide range of trust models, e.g., if a piece of data is a public key, a binding is effectively a public key certificate.
NDN 's data-centric security can be extended to content access control and infrastructure security. Applications can control access to data using encryption and distribute keys as encrypted NDN data, limiting the data security boundary to the context of a single application[19].

3.3.3. Routing and Forwarding

As we mentioned before, Interest packets contain the name of the requested object. Based on that name, the forwarding of Interest packets is done in NDN. As the Interest packet is forwarded, the NDN routers keep state of the pending interests in its PIT.
Routing is done in a way that is similar to IP routing, except for that it is based on names. An NDN router announces the prefixes that it can serve, these prefixes are announced using a routing protocol. Using these announcement, each router builds its own FIB. Conventional routing protocols can be used in NDN also, for example, OSPFN [23] is an extended version of OSPF that distributes name prefixes and calculates routes to name prefixes.
NDN routers do component-wise longest prefix match in order to match the content name requested in an Interest packet.
NDN inherently supports multipath routing, in the contrary to IP routing, that adopts a single best path to prevent loops. In NDN, Interests can not loop persistently, that is because Interest packets contain a random nonce, as shown in Figure 3.2. The random nonce, in addition to the name, form a combination that allows detecting duplicate packets. Data does not loop as they take the reverse path of Interests. Having this feature in NDN, allows the router to send out Interest packets using multiple interfaces. The first Data coming back will satisfy the Interest and be cached locally, later arriving copies will be discarded.

3.3.4. Caching

When a router receives a Data packet matching an entry in its PIT, it caches the data in its CS to satisfy possible future requests, this is called in-network caching.
Caching in NDN provides a lot of benefits; first of all, it helps dissociating objects from their producer. It also reduces the overhead at the producer side and avoids a single point of failure by caching multiple copies of the same content in the network. Additionally, dynamic content can benefit from caching in the case of multicast (e.g., realtime teleconferencing) or retransmission after a packet loss[24]. Also caching the content can reduce the network load and data retrieval latency.
There are two main questions to be answered when dealing with caching: where to cache and what previous cached content should be replaced first? These two questioned are answered with the Decision Strategy (DS) and Replacement Strategy (RS) consequently. The DS is used

to determine in which router along the reverse path of an Interest packet the data will be cached. The most-well known and used ones are as follows:

- Leave Copy Everywhere (LCE), where each router on the path of the Data packet caches the object.
- Leave Copy Down (LCD), in which only the first router (in the path of the Data packet) after the producer or the router where the object was found caches the object.
- Leaving Copies with Probability (LCProb) caches the objects at the router using the caching probability *1/(hop count)*.
- Leaving Copies with Uniform Probability (LCUniP) caches the the objects at the router using uniform probability.

RS is used to determine which object in the CS to replace when it is full and a new object needs to be cached. The most-well known and used strategies are as follows:

- First In First Out (FIFO), Where the first pushed object into the CS is replaced first.
- Random Replacement (RR), where objects to be replaced are chosen randomly.
- Least Recently Used (LRU), where the least recently used object is replaced first.
- Least Frequently Used (LFU), where the least frequently used object is replaced first.

4. Fetching DOI identified objects via NDN networks

In this chapter we present our approach for facilitating the fetching of DOI identified objects via NDN networks.
Considering that research objects in a big data infrastructure are identified with a PID, it is important to be able to fetch objects using DOIs in a NDN network.

4.1. requirements

The approach we are proposing assumes that a fully functional NDN network is operating separately from the IP network. The aim of this approach is to support requesting an object either with a DOI or with an NDN name.

4.2. Design

The flow chart of the proposed approach is shown in Figure 4.1. When the requested name is a DOI, the corresponding NDN name would be first derived from it and an Interest packet with that name would be sent to the NDN network. Transforming the name to NDN name is done first in order to check if the object has been requested before and is still available in the NDN network. If the object is found in the network then it is returned in a Data packet to the user, or else a request is sent to the DOI system to fetch the object. When the object is returned, a NDN name is derived from it and it is published in the NDN network.
If a non-DOI is requested, then it is either a NDN name or a name derived from DOI, if neither, then it might be an illegal name. If it is a NDN name, then an Interest packet is sent with that name to the NDN network. If the object is found then it is returned in a Data packet to the user, or else it passes through a conditional to test if it was derived from a DOI, if not, then it is an illegal name. If the NDN name is derived from a DOI then a request is sent to the DOI system. This last step is used because the NDN name provided might be for an object that is originally fetched using a DOI but at some point it was replaced in the routers' caches.

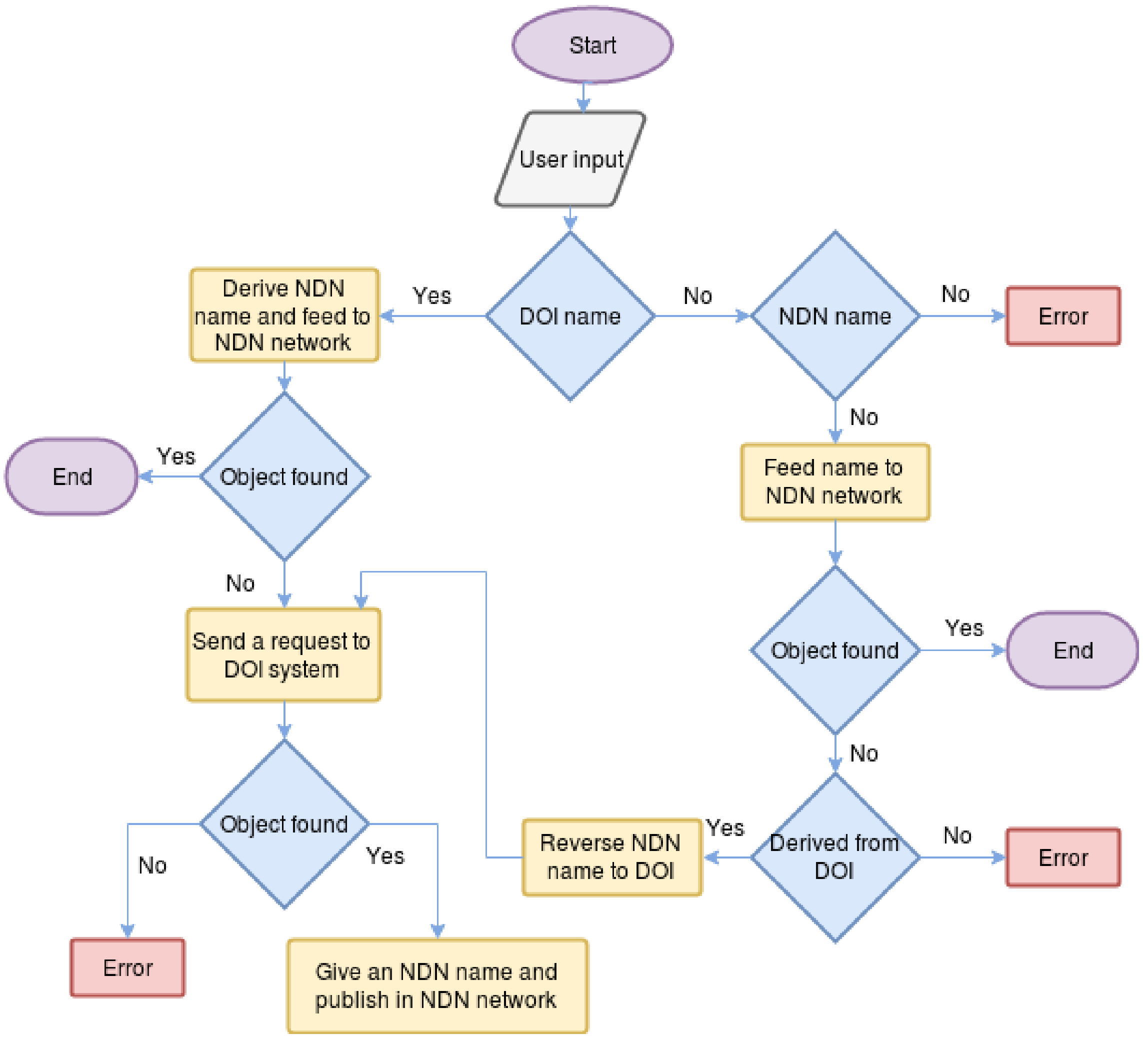


Figure 4.1: Flowchart of the proposed approach

4.3. Prototype

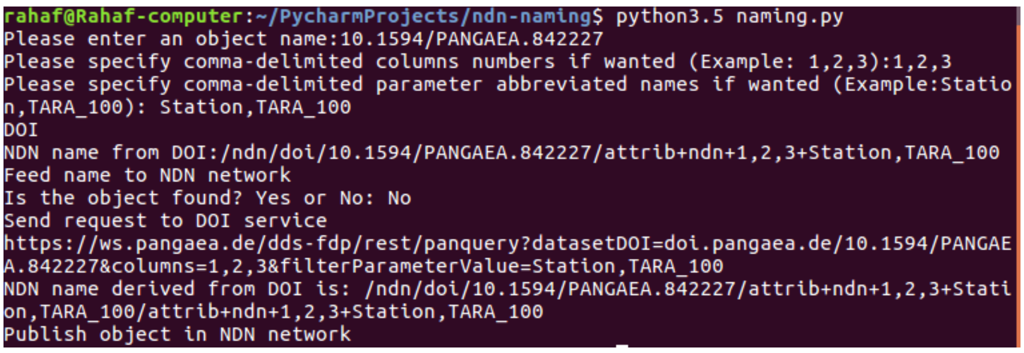
To achieve our goal, we have written a python script. In this script, we used the information system PANGAEA [1]. PANGAEA is an open access library aimed at archiving, publishing and distributing georeferenced data from earth system research. Data published on this system can be retrieved using DOIs. In addition, parts of objects can be retrieved; such as specific columns of data sets, or sampling events from a specific station. PANGAEA has a download service that can be used for direct download of objects[2]. We used that service for retrieving the objects requested by the used and also added to possibility of retrieving parts of objects. For example a user can ask for DOI "10.1594/PANGAEA.842227", and specify that the wanted columns are 1,2,3 and all sampling events from station TARA_100. The request that would be sent to the download service is as following:

[1]https://www.pangaea.de/
[2]https://ws.pangaea.de/dds-fdp/

```
https://ws.pangaea.de/dds-fdp/rest/panquery?datasetDOI=doi.pangaea.de/10.1594/
    ↪ PANGAEA.842227&columns=1,,2,3&filterParameterValue=Station,TARA_100
```

Deriving the NDN name from DOI is done as following: a prefix is added before the DOI name such as "/ndn/doi/10.1594/PANGAEA.842227". Reversing the DOI name to NDN name would be done by removing the prefix.
This whole process can run as a browser extension. In this case, the naming of the published DOI objects would change from a device to another depending on the prefix it can publish with. Yet it is possible to assign only the DOI objects names containing "/doi/10.*/" prefix, this way it would make matching the NDN names derived from a DOI easier.

A demonstration of the functionality of the script is shown in Figure 4.2. The object requested in the demonstration has the DOI name:"10.1594/PANGAEA.842227". Specific columns and parameters are requested, meaning that part of the object would be retrieved. First, the NDN name is derived from the DOI name. When the object is not found in the NDN network, a GET request is sent to the shown link. When the object is received from the download service, it is given a NDN name and published in the network.



```
rahaf@Rahaf-computer:~/PycharmProjects/ndn-naming$ python3.5 naming.py
Please enter an object name:10.1594/PANGAEA.842227
Please specify comma-delimited columns numbers if wanted (Example: 1,2,3):1,2,3
Please specify comma-delimited parameter abbreviated names if wanted (Example:Statio
n,TARA_100): Station,TARA_100
DOI
NDN name from DOI:/ndn/doi/10.1594/PANGAEA.842227/attrib+ndn+1,2,3+Station,TARA_100
Feed name to NDN network
Is the object found? Yes or No: No
Send request to DOI service
https://ws.pangaea.de/dds-fdp/rest/panquery?datasetDOI=doi.pangaea.de/10.1594/PANGAE
A.842227&columns=1,2,3&filterParameterValue=Station,TARA_100
NDN name derived from DOI is: /ndn/doi/10.1594/PANGAEA.842227/attrib+ndn+1,2,3+Stati
on,TARA_100/attrib+ndn+1,2,3+Station,TARA_100
Publish object in NDN network
```

Figure 4.2: Demonstration of the script functionality

5. Application aware NDN performance optimization

In the previous chapter, we proposed a light solution for fetching DOI identified objects via NDN networks, which facilitates NDN integration with big data infrastructures. In this chapter we present an approach for optimizing the performance of NDN network using application level knowledge, e.g., objects' sizes, number, and ordering. Such an optimization is aimed to serve the integration of NDN networks with big data infrastructures.
Data centric applications (such as a scientific workflow) often requires different data objects from multiple locations. For example, an environmental modeling application need data from different domains, such as atmosphere, ocean, eco-system and solid earth. To repeat an experiment, a set of data objects are often needed as initial input. During the execution of the workflow, such application may also need more data objects, e.g., for tuning the simulation or comparison purposes. These data objects may be published by different producers and located in different locations, they also has different sizes. The most relevant variables to our approach are the data objects' sizes and the number of objects in a set, which we call the "Window Size".

5.1. Approach

In the approach we are proposing, we assume that the application has the knowledge about the sizes of the objects requested, this can be achieved using for example, metadata catalogs. The application would group a set of objects to request, we call the number of objects in this set the "Window Size". Then the application orders the objects names according to their size, either in ascending or descending order. Then Interest packets are sent one by one to the closest router.

5.2. Experiment Setup

In order to evaluate the proposed approach, we have conducted several experiments to investigate the effect of such ordering, in addition to what effect would the change of window size have on the performance of fetching the objects.
The elements used in the experiments are one consumer(where the ordering happens) and

one router, as shown in Figure 5.1 We used only one router in this experiment because gath-

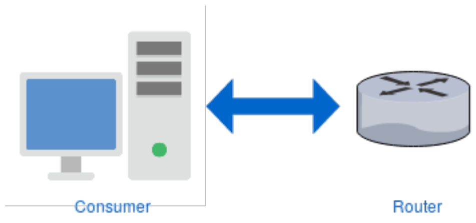


Figure 5.1: The components used in the experiment

ering results from one router (that is acting also as the producer, by generating objects when not found in the cache) can give us a good indication of the change of performance that is happening in the network. This also makes implementing the experiment simpler.
The consumer and the router/producer components are both python scripts which are a part of ndn-cxx library. We have edited those scripts to fit our approach by adding the ordering methods on the consumer side, and cache replacement methods on the router/producer side. The set of objects requested in the experiment contains 30 objects with sizes between 50KB-10GB. We have chosen this variation in sizes in order to simulate a workflow where objects requested have varying sizes. Additionally, that helps in demonstrating the effect of ordering objects, where some large objects may replace smaller ones when the cache is full and vice versa.
On the consumer side, in order to examine the effect of changing the window size on the network's performance, we tested window sizes between 5-50 objects per set in the experiments. For the sake of evaluating our approach, we tested the following ordering methods: random, ascending and descending.
On the router/producer side, we tested the following cache replacement strategies: FIFO, LRU, LFU and RR. The reason behind testing the different cache replacement strategies is that there is a strong connection between the order of objects sent (according to their sizes) with the way the previous cached objects are being replaced. The output of the experiments was the cache hit, which increases by one each time an object is found in the cache. The cache hit gives a strong indication of the performance, as when an object is found then there is no need to fetch if form further away, also no additional latency in fetching the object would occur.
Throughout the experiment, one value of each variable is tested. In each experiment, 1000 interest packets are sent by the consumer. In addition, each experiment is repeated 10 times and the results from the 10 experiments are averaged.

5.3. Results

In this section we present the results of the experiments relating NDN networks performance optimization using application level knowledge, specifically the sizes of the objects. In Figure 5.2 we present the plot showing the cache hit ratio for the different cache replacement strategies. The change of window size in this case does not make a difference because the

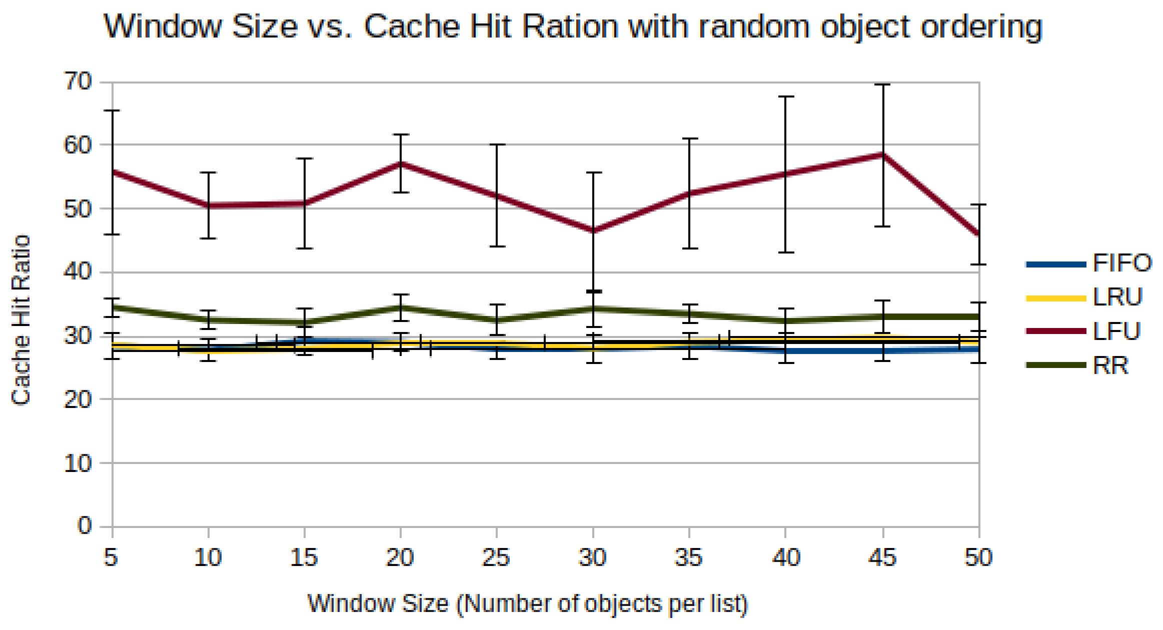


Figure 5.2: Window Size vs. Cache Hit Ratio with random object ordering

objects are sent randomly regarding to their sizes. We present this plot in order to show the different behaviours of the cache replacement strategies. We can see in this plot that LFU cache replacement strategy gives the highest cache hit ratios followed by RR, then LRU and FIFO, which give close cache hit ratio values. These results are taken from the experiments where the cache size of the router/producer is 15 GB.

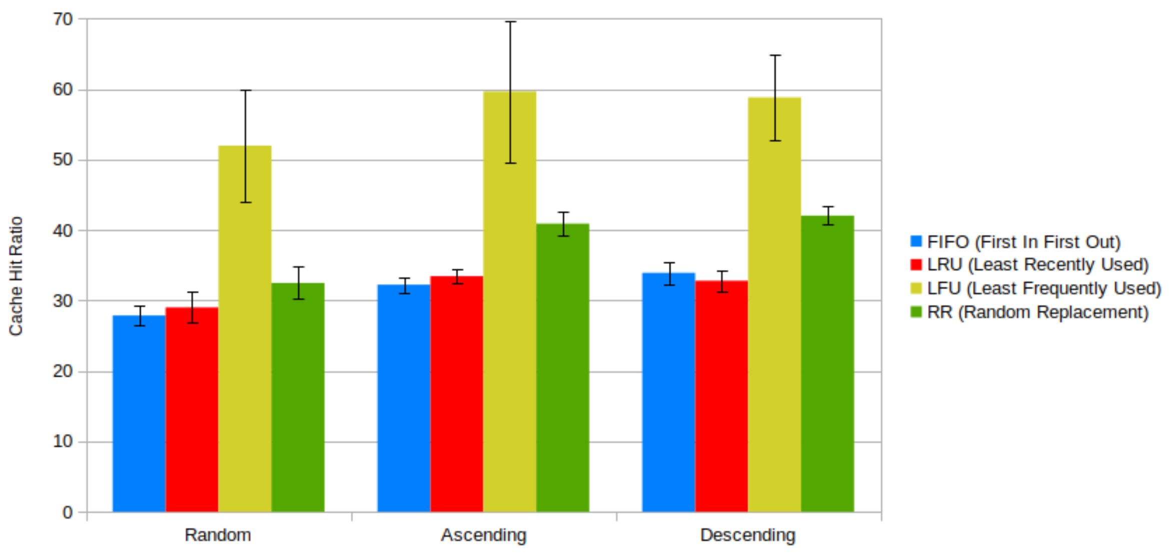


Figure 5.3: Barchart presenting the cache hit ratio of the different cache replacement strategies when applying the different object ordering methods

In Figure 5.3 we present a barchart showing the cache hit ratio values of the different cache replacement strategies with the three ordering methods. The results are taken from the

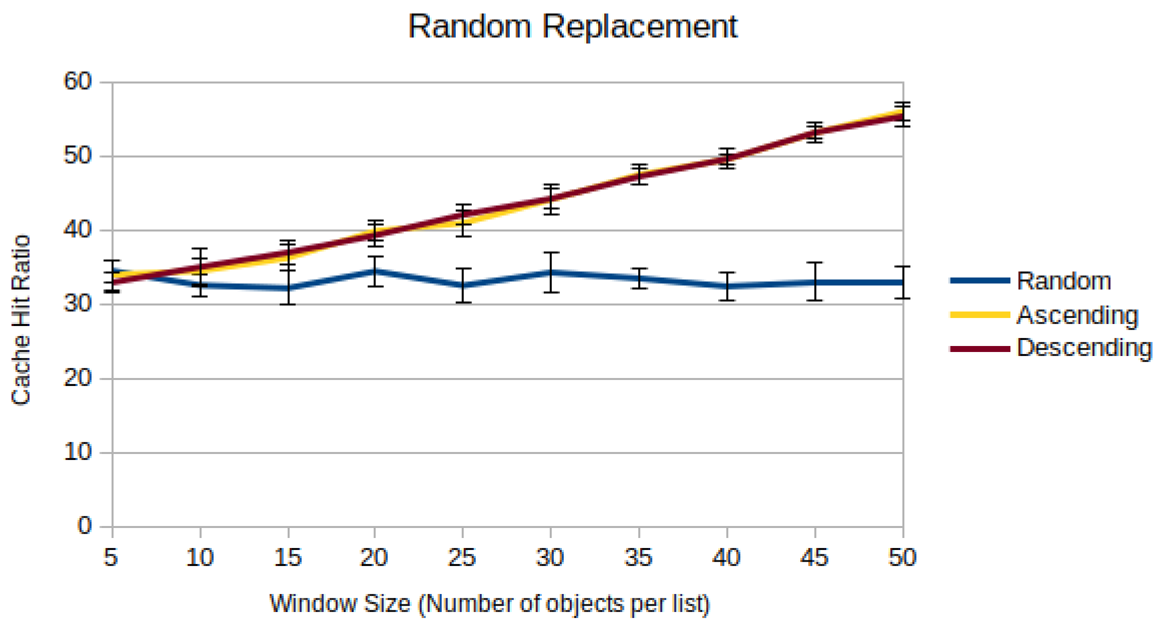


Figure 5.4: Window size vs. cache hit ratio for RR cache replacement strategy

experiments where the cache size is 15GB and the window size is 25 objects per set. We can see in this plot that the highest cache hit ratio values occur with LFU cache replacement method accompanied with ascending or descending ordering methods.

In Figure 5.4 we present the window size vs. the cache hit ratio for RR cache replacement strategy. The results of are taken from the experiments where the cache size is 15GB. This plot shows that both ascending and descending ordering methods give higher cache hit ratio values than the random method. Furthermore, the ascending and descending ordering methods give very close cache hit ratio values in this case.

In Figure 5.5 we present the window size vs. the cache hit ratio for LFU cache replacement strategy. The results are taken from the experiments where the cache size is 15GB. This plot shows that the ascending ordering method gives the highest cache hit ratio values followed by the descending ordering methods then the random method. In addition, the three methods give the highest cache hit ratio values with LFU cache replacement strategy in comparison with the other three examined strategies. The similar plots presenting window size vs. the cache hit ratio for LRU and FIFO cache replacement strategies are presented in the appendix 9 in addition to the tables showing the exact results with their standard deviation values.

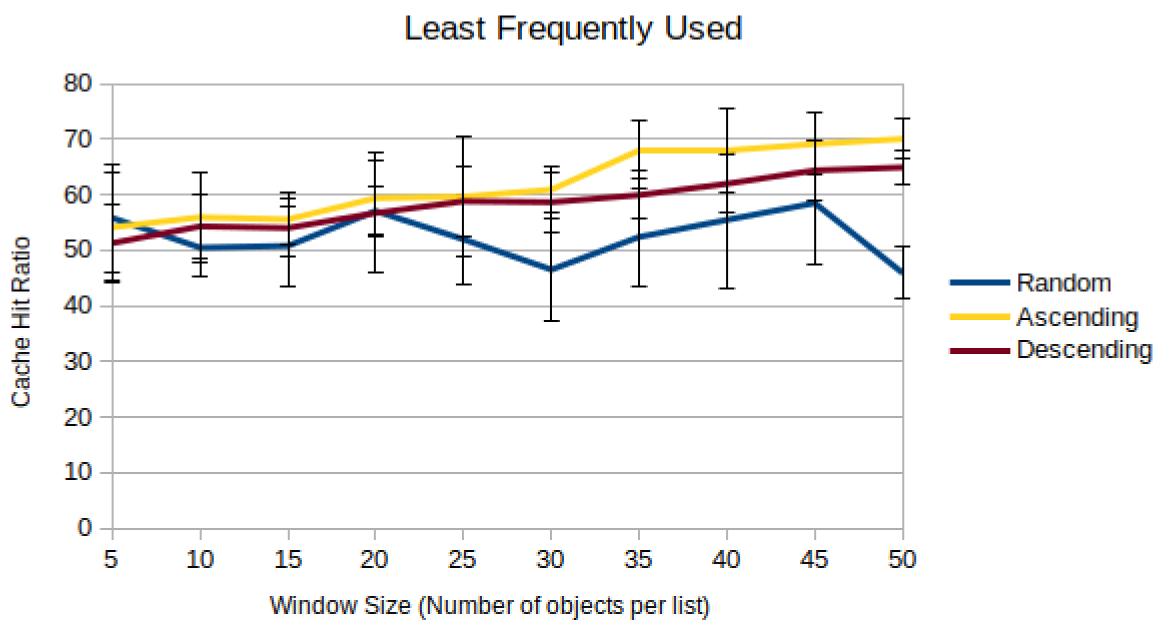


Figure 5.5: Window size vs. cache hit ratio for LFU cache replacement strategy

6. Discussion

In this chapter we will discuss the two approaches we have proposed in Chapters 4 and 5, in addition to their results.

In Chapter 4 we proposed an approach for fetching DOI identified objects via NDN networks. We see now that such a process is possible in principle. Nevertheless our approach does not deal with the difference in naming DOI identified objects within the NDN network between the nodes. If the script we presented is running on the machines connected to the NDN network, then each machine would have a specific prefix/es that it publishes objects under. That means that matching if an NDN name is derived from a DOI would be slightly more complicated than the case is in our script. On the other hand, the element following the prefix of a produces can be specified as "doi/" , for example, for DOI identified objects. In such case, this element can be restricted for DOI identified objects in the whole network which would make the aforementioned matching process feasible.

In Chapter 5 we proposed an approach for optimizing the NDN network performance using application level knowledge, specifically the objects sizes, number and ordering. We also conducted experiments for evaluating the approach, which results' were presented in the aforementioned chapter. From the results of the experiments, we found that ordering objects in ascending or descending order according to their sizes give better performance than sending objects randomly without ordering. In addition, we observed that the best performing ordering method is the ascending ordering, especially when accompanied with LFU cache replacement method. We also found that both ascending and descending ordering methods gave close cache hit results when accompanied with the three other examined strategies.
Looking at the results, we believe that the ascending ordering of objects according to their sizes gives high cache hit ratio values for the following reason: Assuming that we are requesting a number of objects which have different sizes randomly, meaning that we do not order them according to their sizes. In this case, if one of the small objects we are going to request is in the cache of the router, yet we request a large object first, then the small object might be overwritten by the large object when it arrives to the cache of the router. This leads to having to fetch also the small abject although it was available in the first place. Nevertheless if the objects are sent in ascending order according to their sizes, we can make use of the small objects available in the cache before fetching the larger ones.
Likewise, we believe that the descending ordering of objects according to their sizes gives

high cache hit ratio values for a similar reason, which is the following: Assuming that we are requesting a number of objects which have different sizes randomly, meaning that we do not order them according to their sizes. In this case, if one of the large objects we are going to request is in the cache of the router, yet we request a smaller object first, then if the large object was filling the cache, it would be overwritten by the small object when it arrives to the cache of the router. This leads us to having to fetch also the large abject although it was available in the first place. Nevertheless if the objects are sent in descending order according to their sizes, we can make use of the large objects available in the cache before fetching the smaller ones. The benefits of such an approach are many, for one thing, the round trip time it takes for fetching some object would be reduced. Besides, non necessary traffic would be avoided, adding to that, further routers would have to process less requests.

We argue that if the ordering strategies, in combination with the changing window size, are used as intelligence on the consumer side that can dynamically choose the combination of those solutions, the network's performance gain would be high.

The shortcoming of the proposed approach is that it's not always feasible to have the objects sizes on the application side. For that reason, we argue that implementing such functionality on the router side would be more efficient. On the grounds that the router knows what objects are available in its cache, it can choose to deliver the available objects before they get overwritten by later requested objects. Such implementation requires facilitating sending a list of interests in one interest packet, which allows the router to make priority decisions relating which objects to deliver first. The functionality of sending a set of interests that have the same prefix is proposed previously[11], yet, sending a set of interests with different prefixes (different producers) is not available yet for NDN networks.

7. Conclusion

In this project we have presented two approaches to facilitate the integration of NDN networks with big data infrastructures. The first approach deals with fetching DOI identified objects via NDN networks, while the second approach deals with NDN network performance optimization using application level knowledge. We found that it is possible to integrate DOI objects with NDN networks. Nevertheless, a few considerations should be taken into account which were not dealt with in our approach, such as the difference in naming DOI identified objects in the NDN network when they are published by different producers. In addition, our approach focuses on integrating only DOI identified objects withing NDN networks, while there are many other different naming systems that can also be integrated.

From the experiments we conducted for evaluating the approach proposed in Chapter5, we found that implementing ordering based on objects' sizes on the application level enhances the network performance. In addition, we found that LRU cache replacement strategy gives the highest performance with the proposed ascending ordering according to size method. For ascending and descending ordering methods, we found that they gave close cache hit ratio values for FIFO, LRU and RR cache replacement strategies. We believe that changing the ordering methods, in addition to the window size, dynamically as a part of the application intelligence, would leverage the integration of NDN networks in big data infrastructures.

Although we found that implementing such functionality on the application level would enhance the network performance, the shortcoming of not always being able to get the object sizes on the application side restrains this approach. Nevertheless, implementing such functionality on the router side would enhance the network performance, as we argue. That is because the router has the knowledge about what objects are available in its cache and can arrange the order of the sent objects in an optimal way.

8. Future Work

Integrating different naming systems, other than the DOI, in the NDN network would add a great benefit for big data infrastructures. It would also be interesting to facilitate sending a list of interest in one interest packet, which would allow the router then to prioritize sending available objects to the consumer before fetching objects that might overwrite what is already available in the cache.

In this project, we did not get the chance to test the performance of the two proposed approaches combined. We believe that evaluating their performance together would give a clearer view of the influence of these approaches on the integration of NDN with big data infrastructures.

9. Appendix

Python code implementation of the approach proposed in Chapter 4:

```
import re
import urllib3



name = input("Please␣enter␣an␣object␣name:")
columns = input("Please␣specify␣comma-delimited␣columns␣numbers␣if␣wanted␣(Example:
    ↪ ␣1,2,3):") #tadd a test to make sure of the input formate
filterParameterValue= input("Please␣specify␣comma-delimited␣parameter␣abbreviated␣
    ↪ names␣if␣wanted␣(Example:Station,TARA_100):␣")
def testname(name):
    global nametype
    if re.match(r"\b10\.(\d+\.*)+[\/](([^\s\.])+\.*)+\b", name):
        nametype = "doi"
        print("DOI")
    elif re.match(r"/.*/.*", name):
        nametype = "ndn"
        print("NDN")
    else:
        nametype = "Unrecognized"



def add_parameters2name(columns, filterParameterValue):
    global ndn_new
    if columns != "":
        ndn_new = ndn_new + "/attrib+ndn+" + columns # assuming that in our network
            ↪ this part of the name is used only for deriving NDN from DOI
    if filterParameterValue != "":
            ndn_new = ndn_new + "/attrib+ndn+" + filterParameterValue
    else:
        print(ndn_new)
```

```
testname(name)
if nametype == "doi":
    #print("good")
    ndn_new = "/ndn/doi/" + name
    add_parameters2name(columns,filterParameterValue)
    print("NDN␣name␣from␣DOI:" + ndn_new)
    print("Feed␣name␣to␣NDN␣network")
    test_found_doi = input("Is␣the␣object␣found?␣Yes␣or␣No:␣")
    if test_found_doi == "Yes":
        print("Object␣found!")
    elif test_found_doi == "No":
        print("Send␣request␣to␣DOI␣service")
        http = urllib3.PoolManager()
        send_url = "https://ws.pangaea.de/dds-fdp/rest/panquery?datasetDOI=doi.
            ↪ pangaea.de/" + name
        if columns != "":
            send_url = send_url + "&columns=" + columns
        if filterParameterValue != "":
                send_url = send_url + "&filterParameterValue=" +
                    ↪ filterParameterValue

        print(send_url)
        doi_req = http.request('GET', send_url)
        if doi_req.status == 200:
            #add_parameters2name(columns, filterParameterValue)
            print("NDN␣name␣derived␣from␣DOI␣is:␣" + ndn_new)
            print("Publish␣object␣in␣NDN␣network")
        else:
            print("Error␣no␣object␣with␣the␣given␣DOI␣was␣found!")
    else:
        print("Yes␣or␣No") #
elif nametype == "ndn":
    print("Send␣interest␣packet␣to␣NDN␣network!") #add attributes to the ndn name
        ↪ or not? we could find the whole file if not added and we could also get
        ↪ a part of it if added
    #maybe we should ask first for the part with attribute and if not found ask for
        ↪  the whole file ??
    test_found_ndn = input("Is␣the␣object␣found␣in␣NDN␣network?␣Yes␣or␣No:␣␣")
    if test_found_ndn == "Yes":
        print("Done")
    else:
        if re.match(r"/[n][d][n]/[d][o][i]/.*", name):
            print("NDN␣name␣is␣derived␣from␣DOI!")
            doi_name = name[9:]
            doi_name = re.sub(r"attrib\+ndn.*", "", doi_name)#find a way to do these
                ↪  two assignments in one line
            print(doi_name)
            print("Send␣request␣to␣DOI␣system!")
            http = urllib3.PoolManager()
            send_url = "https://ws.pangaea.de/dds-fdp/rest/panquery?datasetDOI=doi.
                ↪ pangaea.de/" + name
            if columns != "":
                send_url = "https://ws.pangaea.de/dds-fdp/rest/panquery?datasetDOI=
                    ↪ doi.pangaea.de/" + name + "&columns=" + columns
            if filterParameterValue != "":
                    send_url = send_url + "&filterParameterValue=" +
                        ↪ filterParameterValue

            doi_req = http.request('GET',send_url)
            print(doi_req.data)
            if doi_req.status == 200:
                print("Publish␣object␣in␣NDN␣network␣with␣the␣name" + name)
            else:
                print("Error␣no␣object␣with␣the␣given␣name␣was␣found␣neither␣in␣NDN␣
                    ↪ network␣or␣DOI␣system!")
        else:
            print("Object␣name␣is␣not␣found␣in␣NDN␣network␣and␣not␣derived␣from␣DOI␣
                ↪ name!")
else:
    print("Error:␣The␣name␣entered␣is␣neither␣a␣DOI␣nor␣an␣NDN␣name!")
```

Window Size	**FIFO**		**LRU**		**LFU**		**RR**	
	Ratio	SD	Ratio	SD	Ratio	SD	Ratio	SD
5	28.46	2	28.59	1.9	55.84	9.8	34.46	1.5
10	27.8	1.6	27.63	1.7	50.49	5.1	32.52	1.5
15	29.18	2.2	28.01	1.3	50.81	7.2	32.12	2.3
20	29.02	1.4	28.93	1.5	57.08	4.5	34.4	2.1
25	27.86	1.4	29.01	2.3	51.98	8	32.48	2.3
30	27.92	2.1	28.08	2	46.53	9.3	34.23	2.7
35	28.44	2	29.03	2.8	52.41	8.8	33.44	1.4
40	27.5	1.7	29.35	2	55.48	12.3	32.36	1.9
45	27.77	1.8	29.74	1.6	58.51	11.2	32.97	2.5
50	27.91	2	29.07	2	45.91	4.7	32.98	2.2

Table 9.1: Window Size vs. Cache Hit Ratio with random object ordering

FIFO						
Window Size	**Random**		**Ascending**		**Descending**	
	Ratio	SD	Ratio	SD	Ratio	SD
5	28.46	2	28.38	1.6	28.39	1.5
10	27.8	1.6	28.78	1.4	29	1.7
15	29.18	2.2	28.95	2	30.2889	2.1
20	29.02	1.4	28.93	1.5	30.59	2.3
25	27.86	1.4	32.24	1.2	33.92	1.6
30	27.92	2.1	36.29	1	37.18	1.7
35	28.44	2	41.28	1	40.91	1.6
40	27.5	1.7	44.19	0.9	44.51	0.7
45	27.77	1.8	47.63	0.7	48.06	0.9
50	27.91	2	50.92	0.7	50.83	0.9

Table 9.2: Window size vs. cache hit ratio for FIFO cache replacement strategy

LRU						
Window Size	**Random**		**Ascending**		**Descending**	
	Ratio	SD	Ratio	SD	Ratio	SD
5	28.59	1.9	28.58	1.6	28.62	2
10	27.63	1.7	28.26	2.2	27.85	2.9
15	28.01	1.3	27	1.9	26.36	2
20	28.93	1.5	29.01	1.8	27.95	1.1
25	29.01	2.3	33.45	1	32.79	1.5
30	28.08	2	36.71	1.1	37.09	1
35	29.03	2.8	40.4	1.2	41.13	1.2
40	29.35	2	44.28	0.6	44.68	1.1
45	29.74	1.6	48	0.7	47.94	0.7
50	29.07	2	51.16	0.7	51.33	0.9

Table 9.3: Window size vs. cache hit ratio for LRU cache replacement strategy

LFU						
Window Size	**Random**		**Ascending**		**Descending**	
	Ratio	SD	Ratio	SD	Ratio	SD
5	55.84	9.8	54.15	9.9	51.31	6.9
10	50.49	5.1	55.96	8	54.31	5.6
15	50.81	7.2	55.58	4.7	54.03	5.2
20	57.08	4.5	59.41	6.7	56.73	10.9
25	51.98	8	59.66	10.8	58.83	6.3
30	46.53	9.3	60.94	4.3	58.67	5.3
35	52.41	8.8	68.03	5.2	59.95	4.3
40	55.48	12.3	68.08	7.6	61.99	5.3
45	58.51	11.2	69.16	5.6	64.4	5.3
50	45.91	4.7	70.07	3.6	64.93	3.1

Table 9.4: Window size vs. cache hit ratio for LFU cache replacement strategy

RR						
Window Size	**Random**		**Ascending**		**Descending**	
	Ratio	SD	Ratio	SD	Ratio	SD
5	34.46	1.5	33.94	2	32.89	1.3
10	32.52	1.5	34.44	1.8	34.97	2.5
15	32.12	2.3	36.2	1.7	36.93	1.5
20	34.4	2.1	39.84	1.3	39.27	1.4
25	32.48	2.3	40.91	1.7	42.04	1.3
30	34.23	2.7	44.1	1.9	44.21	1.3
35	33.44	1.4	47.47	1.3	47.2	1.2
40	32.36	1.9	49.53	0.7	49.57	1.3
45	32.97	2.5	53.1	1.3	53.13	0.9
50	32.98	2.2	55.97	1.2	55.34	1.4

Table 9.5: Window size vs. cache hit ratio for RR cache replacement strategy

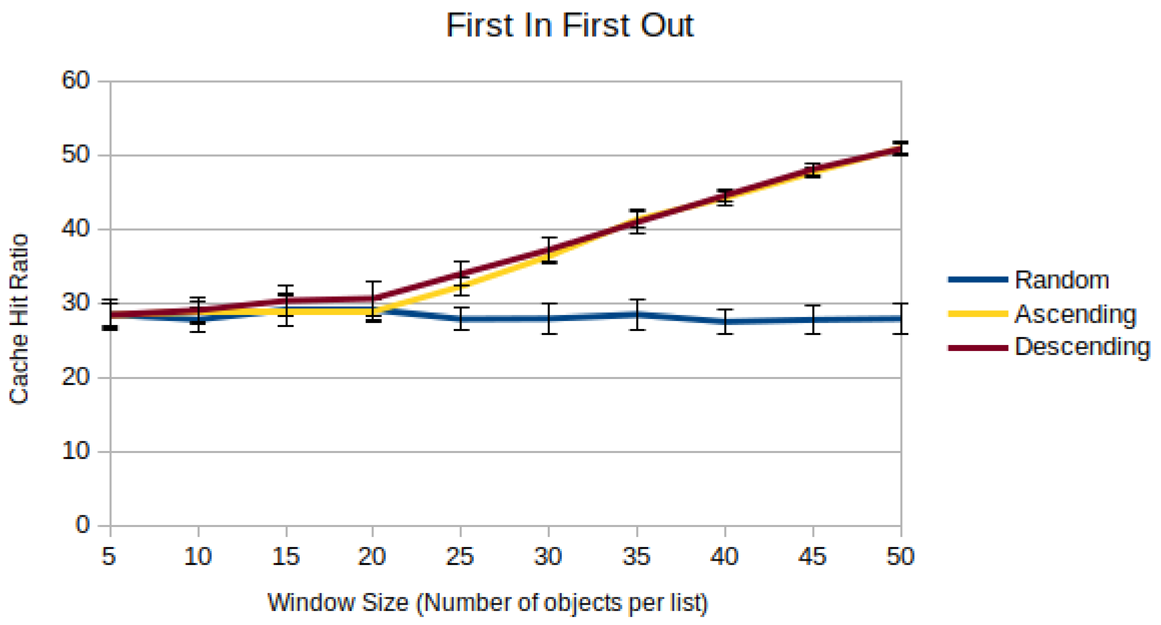


Figure 9.1: Window size vs. cache hit ratio for FIFO cache replacement strategy

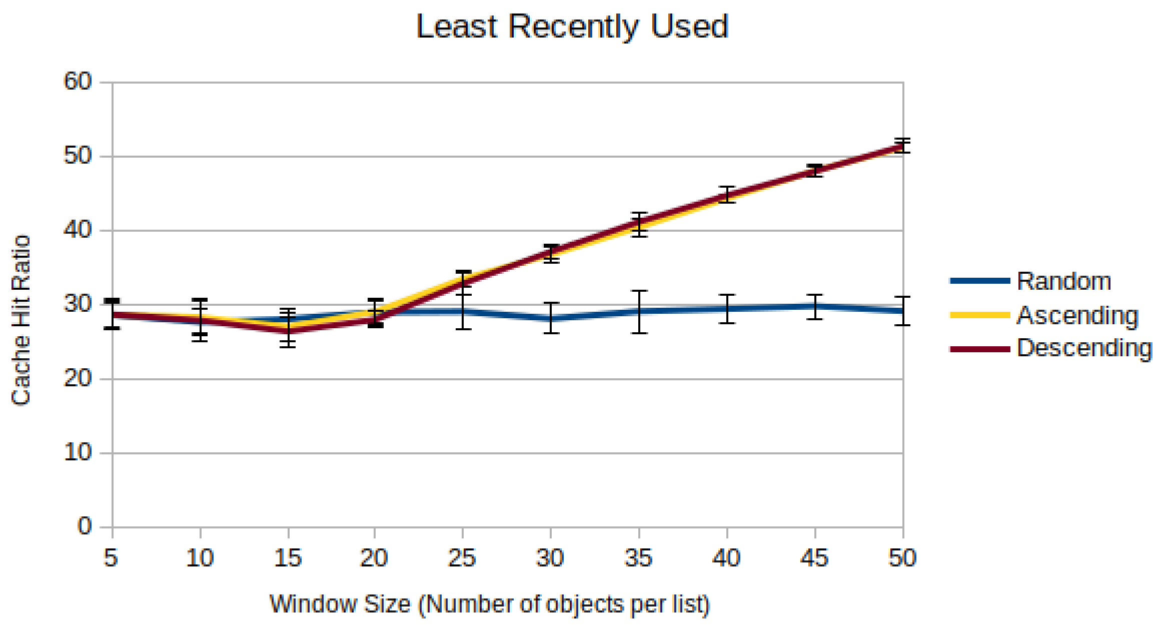


Figure 9.2: Window size vs. cache hit ratio for LRU cache replacement strategy

Bibliography

[1] Wikipedia. Persistent identifier — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Persistent_identifier&oldid=787492944`, 2017. Online; accessed 12-July-2017.

[2] S. Sun, L. Lannom and B. Boesch. Handle System Overview. RFC 3650, November 2003. URL `https://www.ietf.org/rfc/rfc3650.txt`.

[3] Wikipedia. Information-centric networking — wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Information-centric_networking&oldid=773804105`, 2017. Online; accessed 12-7-2017.

[4] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, Christos Papadopoulos, et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 2010.

[5] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[6] named data.net. Ndn project overview. `https://named-data.net/project/`, . Accessed: 2017-06-11.

[7] Min Chen, Hai Jin, Yonggang Wen, and Victor CM Leung. Enabling technologies for future data center networking: a primer. *Ieee Network*, 27(4):8–15, 2013.

[8] Panjai Tantatsanawong, Somkiat Dontongdang, U Prasertsak, et al. Improving big data on research and education networks using future internet approach: A case study of networks analysis. In *Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), 2015 12th International Conference on*, pages 1–5. IEEE, 2015.

[9] Oliver Schmitt, Tim A Majchrzak, and Sven Bingert. Experimental realization of a persistent identifier infrastructure stack for named data networking. In *Networking, Architecture and Storage (NAS), 2015 IEEE International Conference on*, pages 33–38. IEEE, 2015.

[10] Yuxin Jing. *Evaluating caching mechanisms in future Internet architectures*. PhD thesis, Massachusetts Institute of Technology, 2016.

[11] Xiaoke Jiang and Jun Bi. Interest set mechanism to improve the transport of named data networking. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 515–516. ACM, 2013.

[12] ISO. Information and documentation — digital object identifier system. `https://www.iso.org/obp/ui/#iso:std:iso:26324:ed-1:v1:en:fn:1`, 2012. [Online; accessed 12-7-2017].

[13] doi.org. Doi handbook-introduction. `https://www.doi.org/doi_handbook/1_Introduction.html`, 2015. Online; accessed 12-7-2017.

[14] Sam Sun, Larry Lannom, and Brian Boesch. Handle system overview. Technical report, 2003.

[15] doi.org. Doi handbook-numbering. `https://www.doi.org/doi_handbook/2_Numbering.html#2.2.2`, March 2017. Accessed: 2017-07-12.

[16] Laurence Lannom. Handle system overview. 2000.

[17] doi.org. Resolution. `https://www.doi.org/doi_handbook/3_Resolution.html`, March 2017. Accessed: 2017-06-13.

[18] named data.net. Named data networking: Executive summary. `https://named-data.net/project/execsummary/`, . Accessed: 2017-06-11.

[19] named data.net. Named data networking: Motivation details. `https://named-data.net/project/archoverview/`, . Accessed: 2017-07-10.

[20] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Multi-source data retrieval in iot via named data networking. In *Proceedings of the 1st international conference on Information-centric networking*, pages 67–76. ACM, 2014.

[21] named data.net. Signature. `https://named-data.net/doc/ndn-tlv/signature.html`, . Accessed: 2017-07-12.

[22] Alexander Afanasyev, Xiaoke Jiang, Yingdi Yu, Jiewen Tan, Yumin Xia, Allison Mankin, and Lixia Zhang. Ndns: A dns-like name service for ndn.

[23] Lan Wang, AKMM Hoque, Cheng Yi, Adam Alyyan, and Beichuan Zhang. Ospfn: An ospf based routing protocol for named data networking. *University of Memphis and University of Arizona, Tech. Rep*, 2012.

[24] Divya Saxena, Vaskar Raychoudhury, Neeraj Suri, Christian Becker, and Jiannong Cao. Named data networking: a survey. *Computer Science Review*, 19:15–55, 2016.