DISSERTATION

THE FUTURE OF NETWORKING IS THE FUTURE OF BIG DATA

Submitted by

Susmit Shannigrahi

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Summer 2019

ABSTRACT

THE FUTURE OF NETWORKING IS THE FUTURE OF BIG DATA

Scientific domains such as Climate Science, High Energy Particle Physics (HEP), Genomics, Biology, and many others are increasingly moving towards data-oriented workflows. Each of these communities generates, stores and uses massive datasets that reach into terabytes and petabytes, and projected soon to reach exabytes. These communities are also increasingly moving towards a global collaborative model where scientists routinely exchange a significant amount of data. The sheer volume of data and associated complexities associated with maintaining, transferring, and using them, continue to push the limits of the current technologies in multiple dimensions - storage, analysis, networking, and security.

This thesis tackles the networking aspect of big-data science. Networking is the glue that binds all the components of modern scientific workflows, and these communities are becoming increasingly dependent on high-speed, highly reliable networks. The network, as *the* common layer across big-science communities, provides an ideal place for implementing common services. Big-science applications also need to work closely with the network to ensure optimal usage of resources, intelligent routing of requests, and data. Finally, as more communities move towards data-intensive, connected workflows - adopting a service model where the network provides some of the common services reduces not only application complexity but also the necessity of duplicate implementations.

Named Data Networking (NDN) is a new network architecture whose service model aligns better with the needs of these data-oriented applications. NDN's name based paradigm makes it easier to provide intelligent features at the network layer rather than at the application layer. This thesis shows that NDN can push several standard features to the network. This work is the first attempt to apply NDN in the context of large scientific data; in the process, this thesis touches upon scientific data naming, name discovery, real-world deployment of NDN for scientific data, feasibility studies, and the designs of in-network protocols for big-data science.

ACKNOWLEDGEMENTS

This journey has been a long but delightful one. The exercise of listing everyone who has helped - mentors, collaborators, peers, friends, and family - will undoubtedly be incomplete. I am indebted to all the wonderful people whom I had the privileged to work with and learn from, and family and friends who have supported me unconditionally.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

This thesis is about employing Named Data Networking (NDN), a future Internet Architecture, that can better serve scientific communities with large amounts of data. Technologically, big-data dependent science or “big-science” is a critical but very challenging domain with an ever-increasing volume of data, long distance networks, global collaborations, contention for resources at a global scale continuously require cutting-edge technologies that can optimize resource usage, improve performance, reduce bottlenecks, and support scientific workflows. This thesis shows that Named Data Networking can support big-science communities better through simplified data publication and discovery, fast retrieval, intelligent resource usage, and the support for functionality such as remote computation. More importantly, this thesis shows how an NDN-based network can move the common functionality into the network and allows reusability of protocols and software across scientific domains.

Scientific communities are entering a new era of exploration and discovery in many fields, from climate science [57] to High Energy Particle physics (HEP) [26], from Astrophysics [59] [66] to Genomics [21], and from Seismology [182] to Biomedical research [117], that is complemented by data. An increasing number of science experiments are making observations, generating a hypothesis, creating experiments (simulations, observational experiments, or both), and collecting data. Based on the simulated or observed data, changes to the hypothesis and experiments are made, until the observations can be adequately explained. These datasets used for the experiments as well as data derived from these experiments then become reference datasets that are used in further research that continues to move the cycle of scientific discovery forward.

With the help of data-intensive science, scientists are now able to use the observed and simulated dataset to translate abstract ideas into conclusive findings and concrete solutions. Data-intensive science is opening up radically new opportunities as well. Take for example the Next Generation DNA Sequencing (NSG) . Until the very recent years, DNA sequencing was slow, expensive, and only a few institutes were capable of performing it [23]. With the advances in supercomputers, specialized sequencers and algorithms, the cost of sequencing has dropped considerably and continues to drop. For example, sequencing a complete human genome used to cost around $2.7 Billion in 2013, and currently, it costs under a thousand USD [23]. With commercial incentives, several companies offer full sequencing under one hundred USD. This massive drop in cost has opened the door for more advanced scientific discovery - for example, earlier scientists could only focus on testing their hypothesis on a smaller number of genomes. With more publicly available datasets [21], scientists can test their hypothesis against a larger number of genomes, potentially enabling them to identify rare mutations, precisely classify diseases, and more accurately treat the diseases [156].

Another such example is the climate community. In a recently published article [110] researchers from UC Davis show that extreme heatwaves in the Central California Valley directly correlates with heavy rainfall half-a-world away - over the Indian Ocean and Southeast Asia. For this study, the researchers utilized weather data collected from several weather stations in California as well as data from the Australian Bureau of Meteorology. Using heat wave data from June through September collected over 32 years, from 1979 to 2010, they show that heavy rainfall over the Indian ocean can predict an oncoming heatwave (within 4-14 days) in central California.

These ongoing paradigm shift is not only restricted to the Genomics or climate community but is common across many scientific domains. The Large Hadron Collider, the Climate modeling supercomputers, the large telescopes, low-cost genome sequencers, the high-stability, high-resolution transmission electron microscopes, are but a few examples of this shift where science is becoming increasingly intertwined with data-intensive technologies, high-performance algorithms and computing facilities. The volume of data that is being recorded, published, and analyzed is highest in the human history [57] [50] [99] [172] - leading to important scientific discoveries that enhance the understanding of our surroundings, present pictures of atoms and the galaxy, and cure diseases that were untreatable until very recently.

On the other side of the story, the shift of scientific communities towards data-intensive research has started to create huge amounts of data. For example, the largest data- and network-intensive programs such as the Earth System Grid Federation (ESGF) [57], the Large Hadron Collider (LHC) [26] program, the Large Synoptic Space Telescope (LSST) [59], photon-based Sciences, the Joint Genome Institute applications, and many other data-intensive emerging areas have each produced Petabytes of data and quickly approaching the Exabyte scale [141]. The global nature of these communities along with the astronomical rate of growth in data volume create unprecedented challenges: in global data distribution, processing, access and analysis, in the coordinated use of massive but still limited computing, storage and network resources, and in the coordinated operation and collaboration within global scientific enterprises each encompassing hundreds to thousands of scientists. While data-intensive science makes the scientists more efficient in their scientific endeavors, this newly found efficiency, in turn, generates more data. Therefore, the continued cycle of scientific breakthroughs in each of these fields depends crucially on the ability to efficiently manage and utilize massive datasets whose scale and complexity continues to grow exponentially with time.

1.1 Problems of Scientific Data Management

Due to the massive volume of data and the lack of standard organization practices, these scientific communities face tremendous data management challenges in archiving, searching, sharing, visualizing, and analyzing the datasets generated by large-scale measurements and supercomputer simulations. Each of these domains also requires complex workflows supported by large-scale computing, data handling, and network capacities; scientists in these domains waste significant resources on the logistics of handling and managing Big Data. While high-speed networks, data repositories, and sophisticated software have helped to reduce such complexity significantly, these communities need to maintain their unique software and hardware infrastructures, ensure data integrity, and improve data distribution mechanism. Moreover, as research progresses and data is distributed, these already massive datasets are further enlarged with additional copies, metadata, annotations, emails, and other auxiliary information. These datasets are often generated or stored at different sites with elusive ties back to the original dataset.

In addition to the size of the datasets, the number of Big Data users and their download demands has been increasing at an exponential rate [12] [50]. For keeping up with the demand of ever-increasing global participants, various scientific communities have created a plethora of specialized scientific data management solutions that painstakingly reimplement same functionality [12] [1] [144]. These solutions are almost always at the application layer, hard to use, and requires familiarity with various home-grown data formats, naming schemes and storage architectures. This lack of flexibility affects both data producers and consumers. Users often spend hours, days or weeks to find and download big datasets which often fails midway or are unacceptably slow. Even when users are only interested in a small subset of the data, they often need to download a massive amount of data only to discard a significant portion, usually due to lack of sufficient granularity in data access.

The scientific workflows vary greatly. However, from a scientist's perspective, the following describes a very high-level picture; data is created at a central location (e.g., LHC), collected from various distributed sources (e.g., Weather data) and moved to storage hosts. Strategically placed hosts on Internet stores and serves large amounts of data to scientists across the world. Data is often replicated to other hosts to ensure reliability and load distribution. In many communities, high-performance computer clusters or supercomputers provide computation capability. Storage at these computation locations are not infinite, so data must be moved from the storage location to computation facilities and eventually removed after computation finishes. The network plays a vital role in this ecosystem - connecting instruments, connecting storage and supercomputing facilities, connecting users to these facilities, routing requests to them, transporting data as well as results, and supporting a plethora of applications for various functionality, data access, computation, visualization, and collaboration.

1.2 Networking for Big Science

In the scientific communities, the network is often viewed as a black box, offering only packet transport and connectivity. The current Internet protocols were not designed to support Petabytes of Data, and certainly not Exabytes; the Internet designers did not anticipate this deluge of data and networking design artifacts such as end-to-end connections, congestion control algorithms, stateless networks often works at odd with large data ecosystems. Additionally, since the current network exposes very little and supports a limited set of operations by design, each of these software ecosystems must develop ancillary software and protocols for monitoring and interacting with the network, leading to the increased complexity of the software ecosystems. Technologies such as Content Distribution Networks, Software Defined Networking, Peer-to-Peer networking, and Network Virtualization are changing this perspective slowly, though the underlying limitations remain.

The current model of data distribution and access does not affect only the users but also the data producers. Producers often see a large number of failures and duplicate requests [161] which waste resources and causes even more failures. Since the available resources are not sufficient to rapidly accommodate all requests for data and results at any given location [166], these failed requests, and subsequent retries create a ripple effect in multiple dimensions - network, storage, and computing power. These problems necessitate the optimization of workflow in such a way that makes the most efficient use of the available resources while serving the highest priority requests with reasonably short turnaround time. However, only optimizing the resources is not enough, and steps must be taken to reduce wasting the resources and to reduce the turnaround times. An example of a current issue to be resolved is that of failed large transfers following completion of a long-running job; it not only takes resources away from other competing requests, but it also leads to delays and inefficiencies in the research work of the entire scientific collaboration.

Named Data Networking (NDN) , an instance of Information-Centric Networking (ICN), is a new Internet architecture [196] which focuses on the what (the actual content names), rather than the where (the host where the data resides) . By making content, and not the hosts, the central focus of the design, NDN can better address the needs of data-intensive applications. NDN-supported applications can better interact with the network, offload the standard functionality into the network, implement intelligent functionality in the network, and detaches specifics of the path from content retrieval. Several features of NDN can be beneficial to the scientific communities. For example, directly using names in the network for forwarding and content retrieval, in-network caching, and smart forwarding strategies can simplify data management for scientific domains.

This thesis builds on the promise of NDN - it shows the incongruities between data-intensive applications and the current network paradigm. Using NDN at the network layer, this thesis shows how NDN can support data-intensive science. This work uses Climate, High Energy Particle Physics(HEP), and Genomics workflows as examples, studies the problems that these big-data communities face, find the commonalities among these challenges, and shows how to address them better using NDN.

1.3 Contributions

The contribution of this work is in two complementary directions; it contributes to the big-data science by demonstrating how applications can be simplified, generic functionality can be pushed to network, and problems of large data transport addressed. On the other hand, it demonstrates a compelling use case of NDN, an emerging architecture. In the process, this thesis contributes to network research by creating novel in-network protocols, discussing trade-offs of naming schemes, deploying the protocols on an actual testbed, and contributing to the architectural design. This work designs, implements, and deploys NDN-SCI, a generic data management framework built on top of NDN that helps with scientific data publication, discovery, retrieval, and other specialized operations in various workflows. The in-network protocols not only demonstrates NDN's compatibility with big-data science but also shows how NDN can interact with other cutting-edge networking technologies such as Software Defined Networking (SDN), ESnet's OSCARS. Another contribution of this work is presenting the in-network framework that moves standard functionality across scientific domains into the network and showing that an NDN supported network can adequately support the needs for future big-data applications, and at the same time, simplify applications and increase software reusability.

1.4 Organization

The rest of this thesis is organized as follows: first, it discusses the problems of the current network in the context of current scientific workflows. It then introduces NDN in the context of scientific data management. Using a real climate access log, this thesis evaluate NDN's potential to improve large data management and shows the NDN is indeed capable of supporting big-data science. The next chapter discusses the deployment of a real testbed and elaborates the considerations of deploying a new Internet architecture. Chapter 5 presents naming guidelines for NDN based applications and demonstrates a potential migration path for current scientific applications into NDN. Only naming of data is not sufficient for creating a name-based ecosystem; Chapter 6 demonstrates a framework for discovering such names. Finally, Chapter 7 shows how "NDN-SCI", a Data Management framework in the network can support the needs of scientific applications in domains such as Climate Science, High Energy Particle Physics and Genomics. Finally, Chapter 8 summarizes the effort and discusses the lessons learned - lessons that the author hopes would be useful for both scientists and network operators involved in the logistics of big science data as well as the networking research community.

Chapter 2

Background

In the past few decades, scientific domains have seen a proliferation in sophisticated instruments [11], observational facilities [59] [66], and computing resources [11] [54] [111] that have tremendously improved capabilities for scientific communities ranging from climate science [12] to high energy particle physics (HEP) [11], from astrophysics [59] to genomics [144] [172], and from seismology [77] to biomedical and heathcare research [184]. Scientists can now corroborate their theories and mathematical models using simulated [11] [12] and collected data [59] [66], making them more efficient in their scientific endeavors, which, in turn, generate more data. Additionally, using raw data for experiments create more derivative data, metadata, notes, annotations, and indexes, which increases the total data volume. For example, the large telescope project [100] needs additional 15PB of storage for this additional information in addition to the 75PB raw data. Given the size of the data, the continued cycle of breakthroughs in scientific fields depends crucially on our ability to efficiently manage and utilize massive datasets whose scale and complexity continues to grow exponentially [176] [51] with time.

Networking is the underlying technology that enables these intelligent scientific applications [192] [125]. The Internet has proved to be invaluable in supporting new applications in scientific domains, fostering global collaborations, and providing access to data at scale [12] [1] [172]. However, in addition to the size of the datasets, the number of Big Data users and their download demands has been increasing at an exponential rate [176] [51]. For keeping up with the demand of ever-increasing global participants, various scientific communities have come up with their domain-specific software solutions and have created a plethora of specialized scientific data management solutions [3] [172] [77] that painstakingly reimplement same functionality. These solutions are almost always at the application layer, often require familiarity with various community-specific data formats [148] [43] [104], naming schemes [178] [163] [133], and storage architectures. This lack of flexibility and application layer complexity affects both data producers and consumers. Users often spend hours, days or weeks to find and download big datasets which often fails midway [166] or are unacceptably slow [10]. Even when users are only interested in a small subset of the data, they often need to download a massive amount of data only to discard a significant portion, usually due to lack of sufficient granularity [12] in data access.

Further, there is no provenance or verifiability associated with scientific data themselves. The security and trust are associated with the hosts that store the data. Data is trusted only when it comes from a verified and trusted source, such as a repository hosted at a well-known institute. However, lack of trust in data itself means data cannot be served from anywhere - even if data is stored near the user, the user must retrieve the data from a trusted source to ensure validity and provenance of data. However, this method can be sub-optimal since the trusted source can be far away, may fail, or become overloaded.

The current model of data distribution does not affect only the users but also the data producers. Producers often see a large number of failures and duplicate requests [161] which waste resources and can cause even more failures. Since the available resources are not sufficient to rapidly accommodate all requests for data and results at any given location [166], these failed requests, and subsequent retries create a ripple effect in multiple dimensions - network, storage, and computing power. These problems necessitate the optimization of workflow in such a way that makes the most efficient use of the available resources while serving the highest priority requests with reasonably short turnaround time. Moreover, only optimizing the resources is not enough, and steps must be taken to reduce wasting the resources and to reduce the turnaround times. An example of a current issue to be resolved is that of failed large transfers following completion of a long-running job [70] [171]; it not only takes resources away from other competing requests, but it also leads to delays and inefficiencies in the research work [132] of the entire scientific collaboration.

Another problem with current scientific data management paradigm is the definite lack of interoperability among scientific communities [129] [69]. Though the operations on data are similar, e.g., all communities need to publish, discover, and retrieve data, each scientific community has developed independent data management solutions. An elegant solution to this problem could be pushing the common functionality, such as content discovery, retrieval, and placement to the network layer so that all these communities could use them without developing them scratch [72]. However, it is challenging to implement such a model using the existing network model. As a result, each of these communities must (re) -implement everything from scratch, creating an ecosystem of one-off solutions that are often hard to maintain, tightly integrated around a single scientific workflow, and unusable by other scientific communities [72].

This chapter points out the requirements and the problems of scientific data management, many of which are the result of incongruity between application requirements and current workflows and network models [161] [163]. These mismatching semantics lead to complex logic and software modules at the application layer - often created to work around the network's limitations, making them hard to debug and maintain [161] [163] [132].

The significant contribution of this chapter is two-fold - it discusses why these contemporary solutions are inadequate for solving data management problems despite innovative engineering and large amounts of resources. It also shows why intelligent engineering that addresses the network's shortcomings (e.g., CDNs) are not acceptable solutions for most of these communities. At the same time, this chapter discusses how pushing the standard functionality, such as content discovery, retrieval, and content placement into the network layer can create an elegant solution for the big-science communities.

2.1 Requirements for Big Scientific Data

Scientific communities have different workflows and requirements, even with the same communities [76] [101]. The goal of this section is not to exhaustively enumerate all such requirements since such an exercise will most definitely be incomplete and futile. Rather, this section focuses on enumerating the requirements that are common across most scientific communities. These requirements were gathered while working with domain scientists from diverse domains such as Climate Science, HEP, and Genomics.

- Data Publication - In most of the systems, data is published from several nodes around the world [161]. There is a necessity to publish data into the system quickly, without conflict, and consistently. In some cases, such as in HEP, data is published from a centralized place. Currently, data publication is not consistent across communities - often data is hidden in FTP or HTTP sites that are neither indexed nor easy to find.
- Data discovery - Users need to find data easily. In some cases, catalogs are used to hold the names of the data that the users can query. However, in most cases, data discovery begins with the discovery of associated hostname, which in itself is difficult. Currently, users need to know where data is replicated as well - if

there are several replicas, users/applications need to know them all in case one becomes unavailable.

- Time constrained retrieval - Time constrained retrieval is another aspect of big-science. Since existing infrastructure is insufficient to adequately accommodate all requests at a given time [161] [166], users or applications must complete their data retrieval or transactions within a specific time allowing other applications to utilize the network.

- High-Performance bulk data transfers - Scientific applications need to transfer a large amount of data over the network. The transfers are susceptible to loss and congestion. Even a tiny amount of packet loss can kill the data transfer performance [10]. Additionally, having the ability to transfer data from multiple sources at the same time, switching path depending on path characteristics (such as loss) are some of the properties that can help bulk data transfers.

- Pushing computations to the data source - Transferring large amounts of data over the network can be time-consuming and prone to failure. Also, large scale scientific computations need the significant processing power of [192] data centers or supercomputers. The current way to pushing computations to data is not straightforward - applications need to track where data is and push computation to it. A name based network can route the computation to the data source without application involvements.

- Optimized resource usage - Since resources in scientific communities are not infinite, optimizing available resources in the network is essential. Right now, resources are hard to reuse, partially because of the end-to-end nature of the network and also because provenance is not attached to the data itself. However, reusing data already flowing through the network and strategically caching content in the network can cut down on bandwidth usage and free up network resources to the other users.

- Reusability of applications and protocols - Currently, each community designs, develops, deploys, and maintains their protocols and applications that repeat functionality. Reusable frameworks can provide opportunities for developing protocols that can be used by multiple communities cutting down on cost and effort. A number of these common functionalities can be implemented in the network. Functions such as high-speed retrieval, caching, bulk data transfer and others can be network supported and do not necessarily need to be reimplemented repeatedly.

2.2 Problems with scientific data management

The massive volume of science data is a problem for scientific communities [64]. However, data size not the only problem that requires attention [64]; this thesis shows that the data volume merely exposes underlying data management problems in the fundamental building blocks, such as the network, data management applications, and scientific workflows. Further, this thesis argues (and later demonstrates) that a number of these problems arise from the mismatch between the requirements of distributed scientific workflows and the underlying network architecture, either directly or often as an indirect side-effect.

This section presents a background on the problems of scientific big-data management, the factors that contribute to it, and how the network and workflows can be adapted to solve (or minimize) these problems. In summary, scientific data management problems have several axes; the aggregate sizes of data requests for these communities are enormous. Scientific communities often name and organize content in an ad-hoc manner since the ad-hoc organization does not currently inhibit information exchange. However, the ad-hoc organization certainly makes data management process decidedly inefficient and finding, downloading, and reusing datasets very hard. Finally, lack of support from the underlying network means all aspects of data management solutions must be implemented at the application layer (or at the middleware), leading to increased application complexity and reduced software reusability. Each of these problems deserves attention, and the following sections discuss them individually.

2.2.1 Data Volume

The amount of data currently produced in various communities is already enormous. Table 2.1 shows data volume in a few of these scientific disciplines and in *one* community in each of them. This massive volume of data poses a significant challenge to scientific communities. For example, we studied an access log from a climate data distribution node and found that it served 2PB data (approximately 250K 8TB hard drives) in three years [161]. The CMS experiment [166] at CERN [141] produces about 1PB data (125K 8TB hard drives) per month and expects data volume to go up from Petabytes to several Exabytes in the near future [141]. The Large Telescope [100] is projected to produce about 20 terabytes (TB) of raw data per night, and the total amount of data collected over the ten years of operation is projected to be around 60 petabytes (PB) (7.5 million 8TB hard drives) .

The raw data is not the only component that contributes to the cumulative data volume [100] [12]. The raw data must be accompanied by ancillary information such as where they were collected, the name of the project, date, the format of the file and the data, and many other information. Figure. 2.1 shows the metadata information from a single file in the Climate domain. Often each file will contain such metadata [133], leading to a significant increase in the total volume of data. In fact, for the large telescope project [100] this additional information needs another 15PB of storage for

Table 2.1: Data Size in Various Scientific Communities

Scientific Discipline	Community	Data Size	Avg. File Sizes
Atmospheric Sciences	CMIP5	2PB [12]	500MB
Atmospheric Sciences	CMIP6	18PB [29]	-
High Energy Particle Physics	CMS	36PB [22][1]	2GB
Genomics	Human Genomes	2-40EB [172]	100GB [172]
Astrophysics	LSST	75PB	20TB

the catalog database in addition to the 75PB raw data. Figure. 2.1 shows an example of metadata accompanying a climate data file. Once the raw data and the metadata is stored, the actual scientific analysis can begin. Depending on the workflow and the number of users, each workflow can produce orders of magnitude more secondary data than the original raw data [100] [65]. The amount of data produced is only going to explode, by most predictions [100] [141] [172], exponentially.

2.2.2 Inflexibility in Data Access

The immense volume of large scientific data not only puts an immense load on the existing infrastructures, but scientists must also invest their time and effort in creating data distribution infrastructure at the expense of scientific discoveries [12] [1]. Some of the problems cannot be avoided because the data volume is simply too large. However, scientists can avoid many large data transfers by pushing the computation to the data or retrieving a required subset of the larger datasets [163] [166]. Due to lack of flexible tools, workflows, and intelligent protocols at the network, users often retrieve entire datasets at a significant cost [65] [161]; they spend hours, days or weeks downloading big datasets, often canceling and retrying several times [161]. Moreover, users typically retrieve data through various dissimilar processes, often with a substantial manual component requesting it from a colleague, creating logins and navigating public web servers, learning several native data formats, making the process

```
netcdf pr_19020101_060000 {
dimensions:
    time = UNLIMITED ; // (124 currently)
    grid_cells = 40962 ;
variables:
    float pr(time, grid_cells)  ;
        pr:long_name = "Total precipitation rate- 6 hourly" ;
        pr:title = "Total precipitation rate- 6 hourly" ;
        pr:units = "kg/m2/s" ;
        pr:type = "no " ;
        pr:positions = "center" ;
        pr:missing_value = 1.e+36f ;
    double time(time)  ;
        time:quantity = "time" ;
        time:units = "days since 1-1-1" ;
        time:calendar = "noleap" ;

// global attributes:
        :calendar = "noleap" ;
        :institution = "Colorado State University" ;
        :history = "Fri May 18 13:22:42 2012:
        ncks -v pr gatm_y0000201.g2.nc pr_19020101_060000.g2.nc\n",
            "Mon May  7 13:42:01 2012: ncks -d time,612,735
            gatm_0001-08-01T00:00:00.g2.nc ../gatm_y0000201.g2.nc\n",
            "Created: 05/05/2012 at 20:30 -0700 GMT " ;
        :run = " Real world amip test case (02562)     ! " ;
        :grid = "geodesic" ;
        :version = "0.5" ;
        :extravaratts = "yes" ;
        :total_grid_size = 40962 ;
        :NCO = "4.0.2" ;
```

Figure 2.1: Metadata from a Climate Data File

cumbersome [12] [133]. A coherent framework working in sync with the network and scientific workflows can help tackle this ever-increasing volume of datasets.

2.2.3 Disorganization in Data Management

The problem of science data is not only in its massive size but also a distinct lack of data organization [65] [154]. Scientific data is seldom organized optimally for data access. Instead, they are typically organized by the time order (e.g., pr_19020101_060000.g2.nc), or by appending increasing numbers to a fixed string (pr_19020101_060000.g2.0.nc, pr_19020101_060000.g2.1.nc) [179]. The resulting datasets are not only cumbersome to manage, but the arbitrary organization makes them very hard to find, retrieve, and use [127].

Ad-hoc data management processes, various data naming schemes, data formats, and lack of structured metadata makes data management more problematic [161]. A scientist needs to know where data resides and how they are named. Finding the names is a hard problem in itself since most scientific data is not indexed either by search engines or existing scientific data management systems and more often than not, the easiest way to find data is to contact the scientist who generated or collected the data. Even when data location and names are known, moving data between repositories often requires advanced planning and operator intervention because of their large sizes [15]. All these manual components make the data management process cumbersome. The lack of built-in provenance in data means scientists are often unable to reuse data from their neighbors and can only get data from trusted repositories.

The naming of scientific datasets is also arbitrary. For example, data names are often tied to filesystem structures or HTTP URLs making them non-portable. Moreover, the names often do not follow any established naming conventions. Even when naming conventions exist, they are often not enforced [133]. Modifying these names are not easy since there are millions of scripts and other tools in existence that make assumptions about the structure and location of the data.

As data is shared and research progresses, these already unwieldy datasets are further enlarged with additional copies, metadata, annotations, emails and other auxiliary information that not only grow the size of a dataset but are often generated or stored at different sites, with sometimes elusive ties back to the original dataset [12] [59]. Retrieving, grouping and verifying the authenticity of such datasets are very hard. Additionally, as the previous section mentions, the raw data is only a small part of the whole dataset. The derived datasets are often more substantial than the raw data, stored at different locations, along with copies of the original data. Since reproducibility is essential in scientific communities [34] [128], these datasets are often stored with the copy of raw data, increasing the data management challenges. Currently, names are mutable and not strongly bound to the content; the only way to ensure reproducibility is to store both original as well as derived data. This approach not only increases the data size but also creates multiple copies of the same data at various institutes and organizations.

There is also a definite lack of interoperability among scientific communities [129] [69]. Though the operations on data are similar, e.g., all communities need to publish, discover, and retrieve data, each scientific community has developed independent data management solutions, as discussed earlier. Often, such solutions are hard to use, requiring human involvement or familiarity with various home-grown data formats, naming schemes and storage architectures. Besides, all these solutions are built at a considerable cost, both in terms of money and effort [105]. An elegant solution to this problem could be pushing the common functionality, such as content discovery, retrieval, and placement to the network layer so that all these communities could use them without developing them scratch [72]. However, it is impossible to implement such a model in the current IP model. As a result, each of these communities must (re) -implement everything from scratch, creating an ecosystem of one-off solutions that are often hard to maintain, tightly integrated around a single scientific workflow, and unusable by other scientific communities [72].

2.2.4 End-to-end networking paradigm

The architecture of the network itself causes many problems faced by today's data management applications. IP networking uses *hosts* as the core primitive; one must first identify the host responsible for the desired content or service. As a result, data management is complicated by needing to keep location state up to date in the face of changing network dynamics. Several overlay techniques have been proposed on top of the current network that can simplify this challenge. For example, bittorrent [37], tor [169], publish-subscribe systems [87] [86], and peer-to-peer systems [183] [94] can make the network transparent to the applications. However, all these techniques still must maintain location information somewhere and periodically update them.

Host-based networking complicates data discovery and makes data transfers brittle; a failed node requires the consumer applications to identify an alternative content source and restart the transfers. Since data is bound to host in the IP network, data becomes unavailable when the host goes down. Currently, this problem is addressed by replicating data among multiple hosts where each host has a full or partial copy of the content. However, scientific data is usually large and requires massive resources for replicating all data. Replication also increases the overheads for managing such data; each location, data names, and attributes must be recorded in a catalog and multiple instances of the catalog need to be synchronized and updated periodically. Additionally, replication alone does not automatically result in an efficient content distribution system, network and application protocols must get involved to create a successful failover mechanism.

Currently, there are two ways to download data over IP networks. When transfers are not very large or do not need to meet a deadline, data is requested using HTTP or

FTP over a shared network without any QoS guarantees [180]. When it is critical that requested data reach the requester by a deadline, flows are separated from other traffic using reserved resources [204] [25]. Reservations include router resources such as queue capacity and interfaces, as well as the capacity of network links [204]. However, a reservation does not eliminate the underlying architectural shortcomings of TCP/IP, such as aggressive congestion control, inability to optimally utilize network resources, among others.

Network monitoring is hard: The current IP based network layer does not elegantly support network condition monitoring [180]. As a result, the applications (or the users) must keep track of the network performance and if unacceptable, restart the transfers manually. Consequently, scientific communities often deploy intelligent but complex applications that keep track of the network performance and try to adapt in the face of changing conditions. This communication is often indirect since the applications have no way to communicate with the network directly.

TCP congestion control interferes with transfer speed: For a large bandwidth link, losing even one in hundreds of thousands of packets can dramatically reduce transfer speed [41]. To circumvent congestion and packet loss in public networks scientific communities have built dedicated science networks such as the LHC Optical Private Network (LHCOPN) [119] and ESnet [4] that provide dedicated paths for science data. However, while dedicated networks can remove some of the uncertainties of public networks, scientific traffic flowing over these networks may still encounter congestion and packet loss, as a result of a problem in a router or when other scientific flows are competing for bandwidth. Losing any packet during the transfer is very problematic for large data transfers since it usually results in degraded performance. The potential solution proposed by the networking community range from newer congestion control algorithms [48] [149] to multiple parallel TCP streams [84], and from UDP based solutions [124] to multipath TCP [140].

Ad-hoc usage of network resources: Scientific communities routinely transfer data ranging from multiple Terabytes to Petabytes [161]. As this thesis just discussed, using TCP/IP for such large data transfers can dramatically reduce transfer performance [41] leading to missed transfer deadlines, wasted resources, and possible retries from the clients. Due to the ever-changing nature of the large networks, scientific applications often use bandwidth reservation to ensure lossless data transfers and seamless performance. In IP networks, end-to-end channels with reserved bandwidth are usually created using protocols built on top of RSVP [197]. However, the current model of resource reservation can be inefficient [165]. First, reservations are often made by the users in an ad-hoc manner. A user trying to reserve bandwidth must know the data source and the destination, request a reservation, and transfer data within the reservation's validity period [165]. There are several inherent problems associated with this approach. The users need to make sure the chosen source and the destination are optimal and need to know the operational details of the network and its capacity. From the network's point-of-view, the whole affair can be highly inefficient; if a user creates a reservation but only uses a small portion of the available bandwidth, the rest of the reserved bandwidth is wasted. Contents from end-to-end flows are not reusable even if multiple reservations share the same underlying path and retrieve the same content.

IP network architecture does not support big-data applications well: In IP all data transfers are end-to-end. There is also no way for the applications to communicate with the network and the operations necessary for data dissemination is hard to implement. For example, in scientific communities data is often replicated among multiple servers [161] [12]. Theoretically, these replications should help with faster download by supporting parallel retrieval from multiple sources.

However, even when multiple data sources are available, it is hard to utilize them simultaneously at the network layer. Though the networking community has proposed workarounds such as multiple connections at the application layer or multipath TCP [142] at the transport layer, these approaches still require knowledge of the underlying network. Moreover, multipath TCP works only between a pair of source and destination - if the access link to the data producer is congested or if the data producer fails, multipath TCP does not help. Multiple connections at the application layer only work if the data is created to support parallel retrieval. For example, some protocols such as HTTP supports byte range requests. However, using HTTP for all applications might not be optimal, and all applications that choose not to rely on a specific higher layer protocol must implement parallel transfer protocols independently.

Network changes are hard to react to: Traditional IP networks do not keep any state in the network, and there is no built-in measurement state in the network. As a result, even when a path or the data producer degrades, an IP-based network continues to use the same path for data transfer unless there is some external intervention. Recently, there have been multiple efforts to use SDN [83] to measure the performance of paths and rapidly change the paths in case of a failure or degradation. However, this approach is still dependent on external entities to make decisions and communicate these decisions to the router. As a result, the process is still inherently slow - a faster and more straightforward approach would be to keep states in the network routers themselves; keeping states in the routers will enable them to react to the changing conditions rapidly and without external intervention, potentially eliminating the need to involve the applications in network decisions.

2.3 Current Scientific Data Management Systems

Sheer size and complexity of scientific data as well as its distributed nature make data management complex. Scientific datasets are typically generated at geographically distributed sites, curated, and made available to remote users. Significant resources are required to rapidly accommodate all requests for retrieval, post-processing, visualization, and analysis of the data. While large super-computing centers can accommodate such requirements, resources are scarce, and large amounts of data must be moved to various repositories to eliminate bottlenecks. High-Energy Physics datasets are typically generated at a single location and then distributed around the world via a tiered system. They also need to support the distribution of secondary data such as simulation output.

Since the IP networking protocols do not provide uniform frameworks to address these common problems, various communities have designed and developed customized data management software at a significant effort and cost to satisfy their needs. The climate community uses ESGF [57] for searching and accessing CMIP5 [179] data. Similarly, the HEP community has developed xrootd [68] for its data, and the Genomics community has developed iRODS [189]. These IP based applications do not provide an appropriate network service model to facilitate data discovery and retrieval operations smoothly. Moreover, much of the functionality of these software is similar and could potentially be served by a common infrastructure. The next sections describe these data management systems as well as contemporary solutions such as P2P networks and CDNs. The sections also discuss the limitations of these solutions and then introduces NDN in the context of large science requirements.

2.3.1 CDNs as a solution

Content Delivery Networks [186] tries to address some critical problem of content delivery, namely, making content widely available, provide resilience though content replication, and reduce user latency by moving content closer to the user. As a result, CDNs fit very well into the content delivery requirements such as *content generation*, *sharing, and access* [186]. CDNs usually replicate contents on a large number of surrogate servers that are placed strategically at various points on the Internet. These surrogate servers host content for content producers and improves the quality and speed of content delivery, reduce latency and increase efficiency by placing content near the users. Today's CDNs also support dynamic content and video distribution; Netflix, YouTube, Hulu, and others use CDNs for delivery of large amounts of video content.

At a very high level, typical CDNs have three main components [186], origin server(s), request redirection mechanisms, and a large number of surrogate servers strategically places around the Internet. The origin servers contain original content that a content producer wants to distribute. These origin servers are controlled by the content producer or an entity that manages these servers on the content producer's behalf. The surrogate servers are distributed geographically and replicate content from the origin server, either partially or fully. These servers also serve requests on behalf of the origin server, essentially acting as caches. The replication strategy for CDNs differ, some CDNs pull content on a cache miss, some CDNs provide full stack replication, and some CDNs replicate only the larger objects such as video or image files. Sometimes CDNs employ heuristics to prefetch content based on what the users *might* be interested in the near future. The surrogate servers play multiple vital roles; first, they reduce the load on the origin server and prevent slashdot effects. They also reduce the latency experienced by the clients. Finally, they reduce the traffic in the network, which results in savings both regarding money and network resources.

The request redirection is possibly the most important function of a CDN. Request redirection chooses the most *optimal* replica based on several parameters such as network congestion, distance to the user, the load on other replicas, latency to the user and others. Request redirection mechanisms come in many flavors and Table 2.2 summarizes them.

Table 2.2: CDN Request Redirection Mechanisms

DNS based request routing	Uses DNS to route requests to nearest replica
HTTP Redirection	Uses HTTP Redirect to route requests to nearest replica
URL Rewriting	Returns a new URL with best replica
DNS anycast	Use closest DNS server for looking up IP
CDN peering	Direct interconnect between CDNs and data providers
Global load balancing	Centralized system for routing request to best servers
Multi-CDN	Data provider chooses which CDN to use

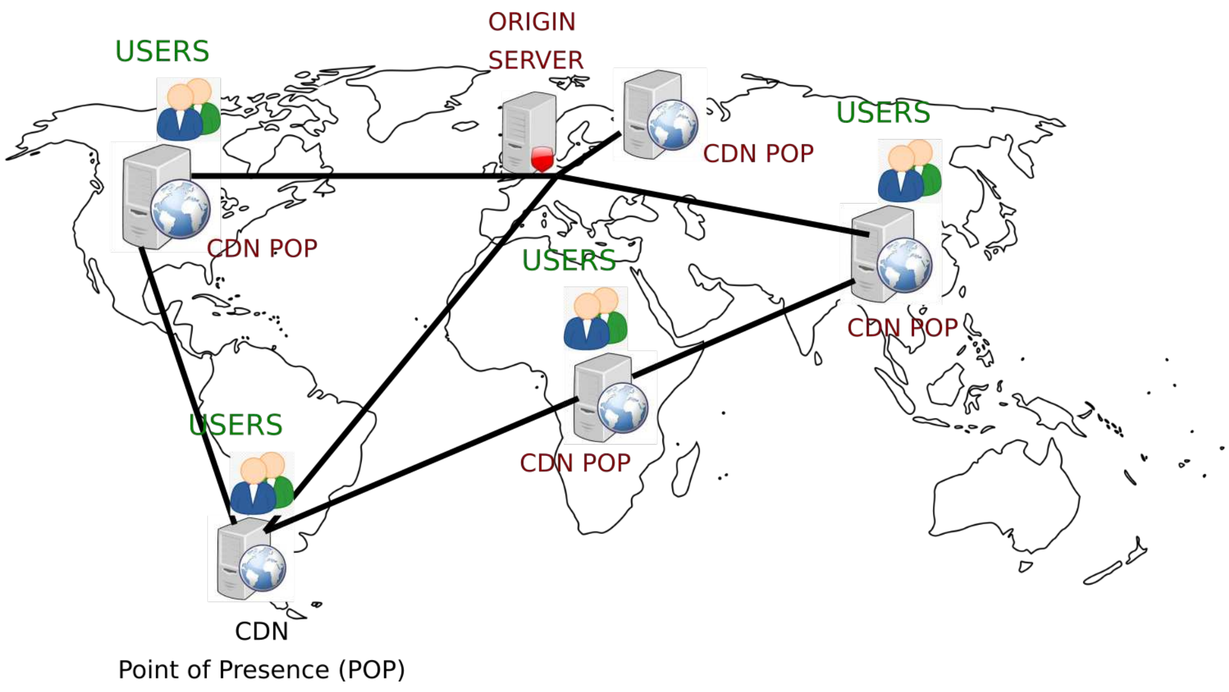


Figure 2.2: CDN Overview [186]

The DNS based request routing [168] is very common in CDNs. The hostname (e.g., cnn.com) points to a DNS server that returns one server IP address from a pool of server IPs. By returning different IP addresses for different requests, CDNs can balance the load on the servers and improve client experience. GLSB [96] [16] uses the same principle, but at a different layer; a GLSB master server keeps track of all the replica servers and their loads. When a browser requests for $cnn.com$, the request comes to the GSLB master server. Depending on the load and the geographic proximity, the master server returns an IP address for the request. The big difference between DNS and GSLB is the fact that DNS is unaware of network or server conditions and load whereas GSLB tracks the health of each server and can make a more informed decision in choosing the surrogate server. URL rewriting and redirections [187] are also used to redirect requests a suitable server- the request for cnn.com is either *rewritten* at the server side to a new URL or a new redirection URL is returned to the client which follows the new URL. The main difference between these two methods is the URL rewriting happens at the server-side, and transparent to the user and URL redirecting happens at the client-side and involves the client. Anycast can also work for redirecting the requests to the nearest server [46] - in DNS anycast, the DNS servers are assigned the same IP address. Based on the routing topology, the network redirects the requests to the nearest DNS server. Note that anycast can also be used to redirect requests to services other than DNS, such as HTTP servers. There are other ways to redirect requests too - for example, a client that needs to send enormous amounts of data over the network (e.g., Netflix) may peer with other CDN providers [39]. In such a scenario, the client deploys servers inside multiple CDNs and control how content is served with the help of "in-house" heuristics and one of the other redirection mechanisms.

The previous section discussed at a high level how CDNs work. Note that all the solutions are built on top of IP semantics and almost always involves clever engineering to work around IP's end-to-end semantics and the limitations. For example, CDNs must carefully consider user access patterns and available server and network resources for provisioning resources. However, the data producer can not always predict how popular their content will be (the so-called "Slashdot effect") and it might not always be economical to over-provision resources in expectation of such a scenario. Another observable shortcoming comes from DNS based redirections - if a CDN uses DNS based redirection and one of the servers goes down, the IP address must be removed from the pool of servers [168]. However, the DNS protocol does not know if a server is up or has gone down - as a result, more sophisticated applications (such as GSBL) must be built that tracks server health and work with DNS to update the available server pool when a server becomes unavailable. DNS updates are not instantaneous, making it difficult to react to ever-changing network conditions efficiently. To summarize, though CDNs provide novel ways to work around the IP network's limitations, they are still limited by the current host-based networking paradigm.

Additionally, while CDNs can address some of the data management challenges, current data management solutions bypass commercial systems in favor of custom solutions, partly due to costs associated with CDNs [161] [6]. In addition to the high hosting and access costs, scientific datasets are extraordinarily long-lived and expected to be useful for several decades [1]. This type of long-term agreements are virtually impossible in commercial CDNs, and the risk of losing data is high if a host CDN goes out of business. Proprietary CDN technology stacks and lack of compatibility between CDNs also makes it hard to switch providers.

Though commercial CDNs have gained much popularity in recent years, they have not gained wide adoption in scientific communities for data distribution. Analyzing a climate access log from ESGF, this work found that for a single workflow, the total transfer size is 2.1 Petabytes over five years or approximately 4TB/month. If a CDN provider such as Microsoft Azure served those requests, it would cost the community over $314,000 [6] for the basic service. Note that ESGF has more than twenty nodes around the world and cumulatively serves much more than 4TB/month. Since CDN costs also grow exponentially with the volume of data [2], serving exabytes of data will make CDNs prohibitively expensive.

Distributing Exabytes of data using CDNs will not only be prohibitively expensive, but CDNs have little or no economic initiatives to carry scientific traffic for free. The long lifetime of these datasets is also a problem. Unlike typical CDN contents such as traditional web pages, videos or files, scientific data are extremely long-lived and therefore, expected to be useful and delivered for several decades. This type of long-term agreements are costly, and the risk of losing data is high if the host CDN goes out of business. Proprietary CDN technology stacks and lack of incompatibility between CDNs make it difficult to switch providers. Such a move will require careful planning, modification of existing tools and protocols. Once the data is locked into a proprietary ecosystem, it is inconvenient at the least to re-develop all the tools that were built around the ecosystem, retrain the personnel, and move to another provider.

Since using commercial CDNs for data distribution is neither very attractive nor practical for the scientific communities, they must develop and maintain home-grown software stack and data distribution infrastructure. Widely varying workflows requires different communities to develop their highly specialized solutions at great effort and cost, primarily at the application layer. Application layer solutions usually target a single workflow which makes them harder to reuse for other scientific workflows. The following sections discuss three such applications, one each for High Energy Particle Physics, Climate, and Genomics.

2.3.2 Peer-to-Peer Networks as a solution

Peer-to-peer (P2P) networks are overlay systems built on top of lower-layer transport and network mechanisms, predominantly TCP/IP. The overlay networks do not have any centralized control, and peers form self-organizing overlays that can offer features such as smart routing, peer selection for storage, advanced routing, fault tolerance and more [116]. P2P networks are different from client-server models in the sense that there is no strict role for any of the participants. Nodes can function as both client and server; some nodes may also have special functions such as tracking content replicas. There are primarily two classes of P2P network overlay - structured and unstructured [116]. Structured P2Ps are tightly controlled and peers agree on how an where to place content - examples are Chord [173], Tapestry [202], Pastry [153], and others. Though structured P2Ps are efficient in locating items, they incur higher overheads in maintaining the overlay [116].

Unstructured overlays organize peers in random order and use various methods such as flooding or random walks to find content [116]. While unstructured overlay means less overhead in maintaining the overlay, the cost of looking up content is higher. Examples of this type of overlays are BitTorrent [91], Freenet [58], Gnutella [151], and others.

BitTorrent is probably the closest in functionality to NDN. The files are split into named blocks, and anyone who wants to download the content can request one or more blocks from a peer. However, the data-centric view exists only at the application layer - underneath, peers must discover other peers' IP address, set up TCP connections to them, and continuously estimate the quality of the connection in terms of download speed and content correctness. Peers also need to incentivize other peers to keep sharing content (a tit-for-tat mechanism) [91].

Though there are projects that share scientific data over BitTorrent, such as Academic Torrents [13] and Open Science Torrents [20], there are several problems with sharing scientific data over BitTorrent. First, there is no way to tell if the content being retrieved is genuine until the download finishes. For large scientific data, this is problematic since this approach wastes resources. Second, obscure content is hard to find; as chapter 3 will show later, this poses a big problem for scientific data where content popularity distribution has a long tail. Third, the performance of a BitTorrent network is directly related to the number of interested peers. So while popular content is downloaded faster, unpopular content will not be. Finally, if a tracker goes offline, the content becomes unavailable. The lack of security and lack of predictable performance makes BitTorrent and similar protocols hard to use for scientific data.

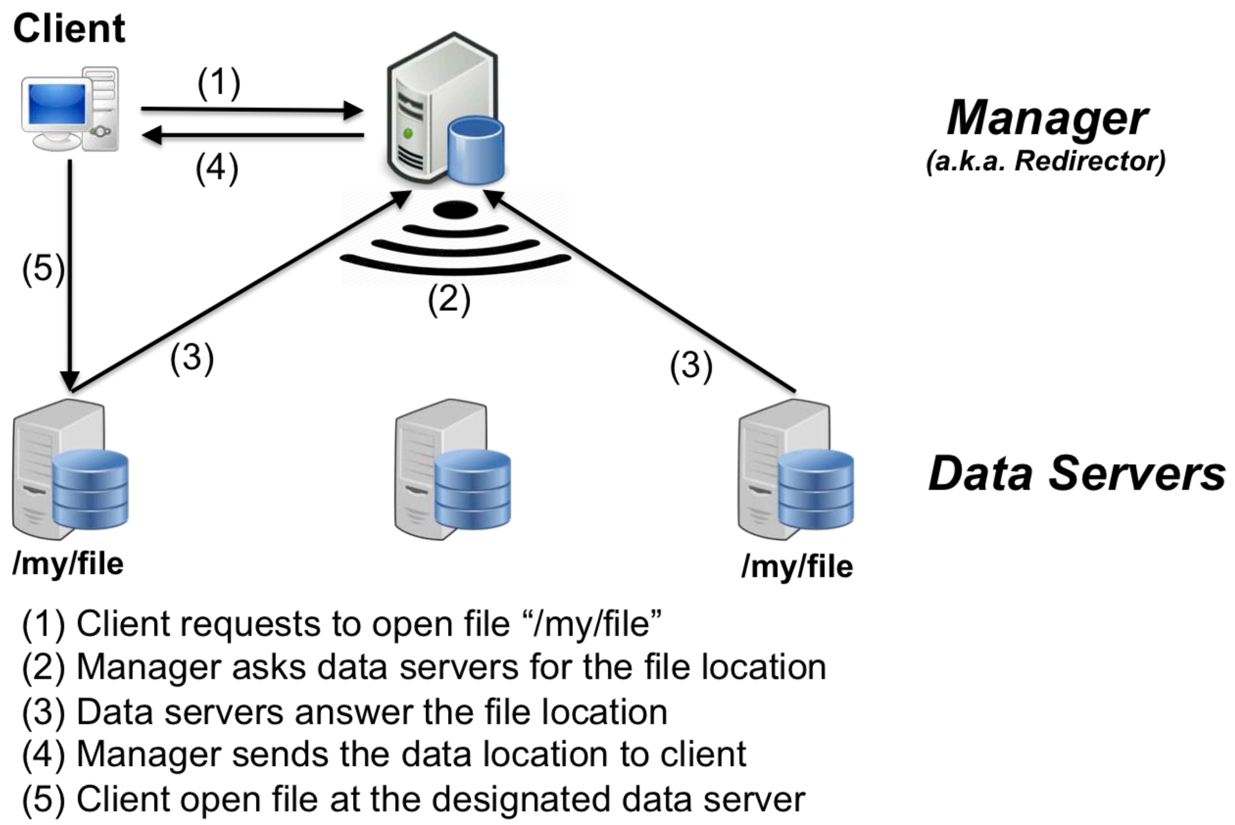

Client
(1)
(4)
Manager
(a.k.a. Redirector)
(5)
(2)
(3)
(3)
Data Servers
/my/file
/my/file
(1) Client requests to open file "/my/file"
(2) Manager asks data servers for the file location
(3) Data servers answer the file location
(4) Manager sends the data location to client
(5) Client open file at the designated data server

Figure 2.3: Xrootd Architecture [1]

However, NDN can facilitate BitTorrent and similar protocols at the network layer and is an active research area [122].

2.3.3 Xrootd

Xrootd ("eXtended" rootd) [68] [1] is a system for scalable cluster data access created by the High Energy Particle Physics community that provides scalable storage, discovery and retrieval capabilities. The Xrootd framework offers an intelligent solution on the server side to distribute the data distribution load among multiple machines that host either partial or fully replicated datasets. At the same time, the framework provides the client side with transparent data access APIs where the users need not know where the data resides.

In xrootd, the data resides on multiple servers organized in a system that dynamically matches clients with servers that have the desired data. Xrootd's architecture is shown in Figure. 2.3. The system consists of a manager, several data servers and the clients. All data servers register themselves with the manager. When a client wishes to open a file the client sends a request to the manager with the desired filename. Upon receiving the request the manager multicasts the request to all the data servers andOnly those servers that have the desired file respond. The manager decides which data server should serve the request and informs the client accordingly. The client then contacts the server directly.

xrootd provides various functionality that facilitates data distribution without client/user involvement. The clients provide user authentication, resource/data location discovery, access and transfer of data both in streaming and block transfer form [68]. Additionally, the system handles failure by finding another working server when a server fails, tracks failed data servers and brings them back into the federation when they become available again, retries the requests automatically until no server is found or a specified number of attempts have been made. Finally, the system also optimizes network resource usage and achieves high throughput by exploiting TCP/IP characteristics, such as multiplexed persistent TCP connections. In the Xrootd system, a server can manage resources by asking clients to delay contacting the server. Furthermore, clients can be redirected to another server at any time, an approach that improves fault tolerance. When a server becomes unavailable, the client launches another search. Xrootd employs several mechanisms to optimize performance and minimize resource usage such as multiple independent streams on a single socket.

The server side in xrootd has four layers [68] [1]; (a) a Network and thread management layer, (b) a Protocol layer, (c) a File system layer, and (a) a Storage layer. Figure. 2.4 shows the server layer architecture. The layered approach allows for flexibility in each layer and a way to substitute modules without affecting other layers.

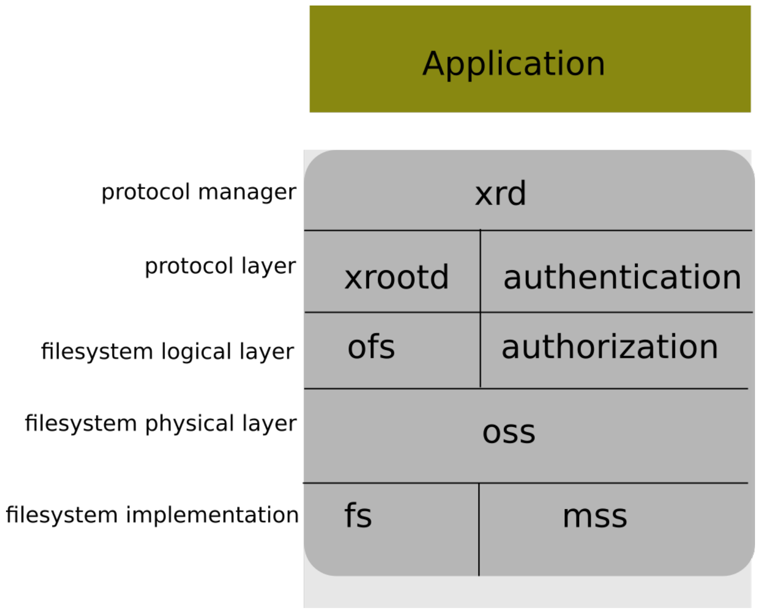


Figure 2.4: Xrootd Server Layers [1]

The network and the thread management layer isolates all other layers from the details of the underlying network. It also maintains a thread pool and assigns work to them when appropriate. The protocol layer is the layer that supports different data access protocols. xrootd is the default protocol in this layer. The xrootd protocol is a TCP based data access protocol with multiple optimizations. For example, it supports multiple independent streams on a single socket which minimizes resource usage.

Additionally, clients are redirected to more suitable servers at any time, allowing for dynamic server selection and load distribution. Besides, the server side may ask the clients to delay server contact. By pushing the decision back to the network, the server is able to avoid being overloaded and allows the network to find another server. A client may also ask the server to be prepared for future transfers, enabling the server to do intelligent prefetching.

The file system later provides a higher layer API for various underlying file systems. Additionally, this layer also merges multiple client requests allowing more efficient file access and resource usage. In addition to eliminating redundant requests, the file system layer also keeps track of active and idle files and closes them as necessary.

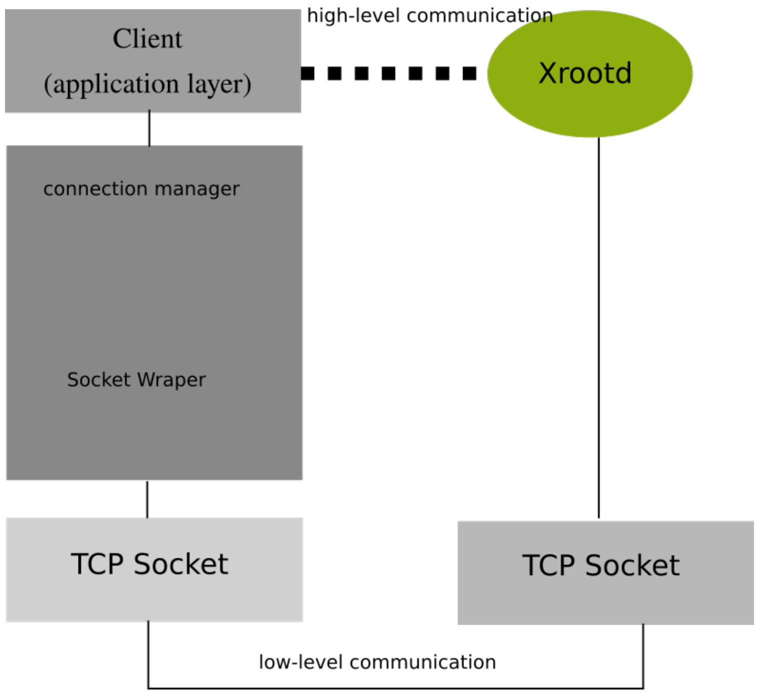


Figure 2.5: Xrootd Client Layers [1]

Finally, the layer provides file-based access control based on the information provided by the protocol layer.

Finally, the storage layer provides a logical file system. The logical file system provides transparent interfaces to the underlying storage. Also, it also provides specific optimization based on the underlying storage. A logical file system ensures a single consistent view of storage, add or remove storage without affecting functionality, and ensures multiple storage systems work together seamlessly.

The client side works with the server side to ensure fault tolerant and reliable behavior of the system. When the server tells the client to delay a request (or when a server crashes) the client re-launches the search for the file. The client has built-in policies for redirection, failure recovery, and read caching. Additionally, the client performs connection multiplexing, data retrieval, and conversion to various data formats.

Xrootd has several limitations since it is built on TCP/IP networks. The following section summarizes the shortcomings.

- Based on TCP/IP networks: xrootd provides many functionalities that are better implemented in the network. Request rerouting, transparent failover, high-speed retrieval are functions that are currently built at the application layer that requires complex engineering. Further, while xrootd tries to optimize network resources by multiplexing multiple connections into a single TCP stream, the mechanism is limited by TCP's inability to perform well over long distance links for large scale data transfers [180].

- No data provenance: Xrootd's security framework allows any authentication protocol and channel-based security but these mechanisms do not currently provide data provenance.

- Use of application-level redirectors: For requesting content, clients must contact a manager/redirector which may be unavailable, or overloaded, creating a choke point or a central point of failure. Xrootd's redirection and fallback mechanisms introduce delays, and the algorithms and parameters for failover must be implemented and tuned case by case, which becomes difficult in the HEP case where the data is distributed at sites in all world regions, some with poor or highly variable network connections [1].

- No transparent failover: Failover in xrootd is not transparent to clients since they are active participants in remedying the failure. When the server or the manager returns a failure code, the client must re-initiate the whole content retrieval process beginning with the content search.

- Complexity: Xrootd protocols are implemented to work around the point-to-point client-server model over TCP/IP networks. This paradigm introduces complexity in the application layer. Functions such as caching of extracted data object collections, and dynamic relocation of datasets are possible, but not implemented automatically due to the complexity [68].

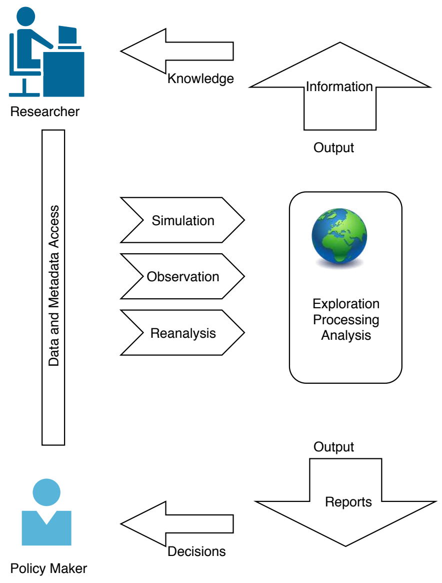


Figure 2.6: ESGF Overview [179]

While xrootd provides an excellent framework for HEP data distribution, it is still limited by the limitations of the TCP/IP network. Besides, it must work around the limitations of the application layer, making it complex and domain specific. xrootd is not the only data management framework that suffers from this problem, as this thesis shows below, data management software from Climate and Genomics communities suffer from similar problems.

2.3.4 ESGF

The Earth System Grid Federation (ESGF) [57] is a data management system used by the climate community to distribute CMIP5 data [179]. ESGF adopts a federated software architecture consisting of multiple geographically distributed nodes that coordinate through a peer-to-peer (P2P) protocol. The development of the system is led by the U.S. Department of Energy's (DOE) Office of Biological and Environmental Research (BER) . Additionally, international institutes and partners in Europe and the US help to develop, deploy, and maintain the software framework as well as the data. The system not only hosts and transfers petabytes of data but also hosts data for a very long term, at least a few decades. Figure. 2.6 reproduces ESGF's architectural overview from the paper that introduces ESGF [179].

ESGF adopts a federated software architecture that has multiple geographically distributed nodes connected over a peer-to-peer (P2P) protocol. Various institutes, known as sites, host one or more of these nodes. There are four types of nodes in ESGF that performs various tasks. **Data nodes** for secure data publication and access, **Index nodes** for indexing and metadata search, **Identity Provider Node** for user authentication and secure delivery of user attributes, and finally, **compute nodes** for data analysis and visualization. ESGF uses off-the-shelf as well as homegrown protocols for providing these functionalities. For example, ESGF uses OpenID for user authentication, a custom catalog for search, and provides wget scripts to the users for data download. Users search for data through the ESGF web portals or desktop client that connects to a local node. This node distributes the user queries to other nodes in the federation, assembles the results, and finally returns them to clients as scripts containing wget requests.

The generator of the data is responsible for naming the data based on community agreed schemes. For ESGF and CMIP5, the most common data naming scheme is described in the data reference syntax manual or the DRS. Once the data is generated and named appropriately, Data and metadata are ingested into a data node via an ESGF Publisher software. This software parses the data file, populates a SQL database on the data node, and the database eventually populates a global database, called the THREDDS catalog. Users access the system through the ESGF web portals or desktop client that connects to a local node which distributes the user query to other nodes in the federation. The users search for the data using the web interface, the local node assembles the results and return them to clients as scripts containing hyperlinks, which clients can then invoke to download data through various tools such as wget, GridFTP [5] or OpenDAP [8].

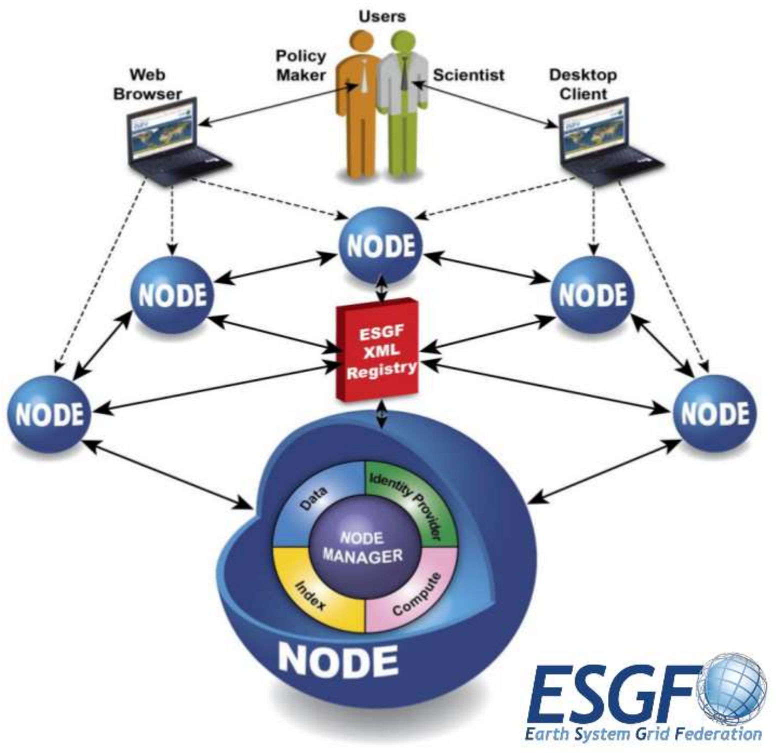


Figure 2.7: ESGF Architecture [179]

ESGF has several limitations as well. Querying different ESGF nodes with the same set of parameters often generates different results [133]. Data distributed through ESGF lacks built-in provenance. Climate community acknowledges the need to support subsetting operations [12] which is not currently supported. Many retrieval tools associated with ESGF cannot provide advanced capabilities such as parallel retrieval or transparent failover. ESGF uses various technologies such as OpenID/PKI-based authentication methods and custom middleware for user authorization and myproxy for PKI infrastructure. However, configuring and maintaining such software is burdensome for both users and administrators [12]. Below, this thesis summarizes these problems.

- No data provenance: Consumers can download CMIP5 data from secure nodes. However, the files themselves are not signed by the data producers and only have checksums associated with them. While checksums are useful for verifying file integrity, they do not provide publisher provenance. Relying on checksums and host-centric security means an attacker can change any datasets and the associated checksums without alerting the users. Lack of publisher provenance also makes in-network caching impossible.

- Batch mode downloads: When users want to download a set of files, the ESGF portal generates a bash script containing several wget commands. A user can then run the script on any Unix system, and the wget commands in the script execute serially. There are several limitations to this approach. Serially requesting data is inefficient since it does not utilize available bandwidth; neither can it use multiple sources for file downloads. These scripts are not intelligent enough to recover from a failure or stop retrieval when failures happen.

- No partial download: The scripts generated by ESGF does not use the resume option provided by Wget, and an unsuccessful transfer has to start over from the beginning. The granularity of a successful download is the whole file - even a missing byte requires restarting the download from the beginning. Users often need to run download scripts multiple times for retrieving all content successfully.

- No parallel download: HTTP does not provide an easy way to parallelize file downloads. Though intelligent clients can use multiple threads to retrieve parts of a file, all requests still go to the same server. In case the server or a link is overloaded, parallel transfers cannot speed up downloads. Since the requests go to the same server, transfers cannot use an alternate server if something fails but must restart the transfer from another server.

- Intelligent clients require complex configuration: ESGF provides Globus [5], a sophisticated client for high-speed transfers. However, globus calls for an elaborate setup, not portable like bash scripts, and requires complex authentication mechanism. Users often prefer simpler but less robust wget scripts over complex setups.

- Authentication failures: ESGF uses OpenID for user authentication. Though CMIP5 data is open, clients need to authenticate for accounting and auditing purposes. Downloaded retrieval scripts request user credentials before running actual wget commands. The community recognizes the authentication module as a ”major pain point” [12]. Indeed, we noticed many failures from the same user within a very short time. If a user improperly configures his/her credentials, all subsequent download requests fail. Though the log specifically does not mention what causes these failures, anecdotal evidence points towards a large number of authentication failure.

- ESGF does not exploit the temporal locality of requests: The IP networking model does not provide request aggregation or caching at the network layer. However, the access log shows a significant amount of temporal locality among duplicate requests. Currently, all these requests must travel to the server, consuming considerable network and server resources. We show in the later sections that even some aggregation will result in considerable bandwidth savings.

- No caching: The fact that requests are temporally close suggests caching might be useful in reducing server and network load. It might also speed up data delivery to the client. Currently, there is no default caching mechanism in the network or the application stack, and therefore, each client must configure and maintain their caches. Convincing each consumer to configure their individual caches is a daunting task. We show that the ability to place caches in the network makes data transfers faster, reduces network and server load, and requires minimal user involvement.

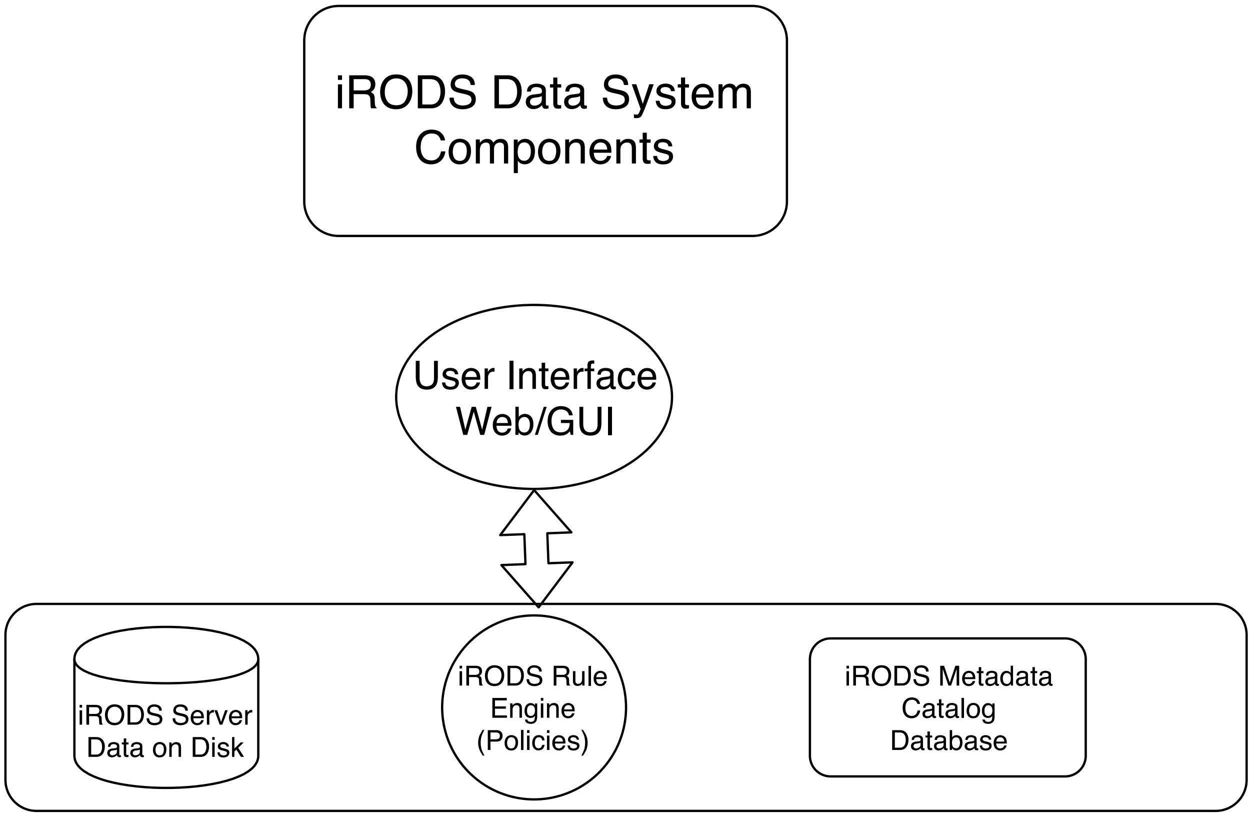

iRODS Data System Components
User Interface Web/GUI
iRODS Server Data on Disk
iRODS Rule Engine (Policies)
iRODS Metadata Catalog Database

Figure 2.8: iRODS Overview [144]

This section shows ESGF makes climate data easily accessible. However, it also inherits significant problems from the IP network and application stacks that it uses as the building blocks. As this thesis shows later, instead of working around the network limitations, a network supported next-generation data management can be more simplified, flexible, and have better performance.

2.3.5 iRods

Similar to xrootd and ESGF, iRODS [144] is a data management solution for the Genomics community. iRods is open-source and solves similar data management problems as xrootd and ESGF but implements them differently. Figure. 2.8 shows the functional overview of iRODS. Each iRODS deployment is a "zone", which are essentially logical separations among different entities. A zone consists of several data servers and a catalog server. The data/storage servers store actual data as well as metadata. For example, an institutional deployment could be a single zone or can have multiple departmental zones. iRODS provides four core functionality for data management as described below.

- Data Virtualization: iRODS organized data objects in ‘collections” [144]. Data objects and collections resemble files and subdirectories, respectively. However, they are logically different from files and subdirectories. However, there are a few subtle differences; collections do not refer to physical storage path, making the system more flexible. A data object may refer to multiple replicas. This way, if a replica becomes unavailable, the data is still available from another replica. These collections and the data objects are then stored in “Storage resources” in iRODS. Each resource has a logical name mapped to an URL like representation, made of the hostname and the storage device. This scheme provides a layer of indirection where the underlying mapping can be changed without changing the logical names.

- Data Discovery: Like most scientific communities, the Genomics communities associate a certain amount of metadata with the actual raw data. These meta-data then can be parsed, indexed, and organized for data discovery. A user of iRODS can associate any metadata with the actual data, the collections, users, or resources. These metadata are then transformed into a catalog and stored into a relational database that can be queried and used for data discovery.

- Workflow Automation: iRODS servers run a rule engine that allows the users to define their workflow. It allows scientists to create specific workflows that are tightly managed but at the same time, automated. It also allows flexible rules where data objects can be manipulated and organized based on the rules. Automating most of the workflow means scientists can no longer need to create custom scripts for automating their jobs, which can often be tedious and error-prone. For example, one rule might be to check the integrity of files before running a job.

- Secure Collaboration: Scientists often need to keep data private due to privacy requirements, non-disclosure agreements, or for practicality reasons. However, scientific collaborations need to share data between multiple groups. Without a framework, keeping track of data and permissions become very difficult. iRODS provides public data access through temporary tickets, a Unix like file system permissions for users and groups, and a federated access model for sharing data across multiple iRODS zones. This flexible, multi-tiered access control enables easy sharing and manageable data security [144].

While iRODS provide a simplified system for data management, similar to xrootd and ESGF, it implements all these intelligent functions at the application layer. Functions such as replication, data transfer, failover are all implemented at the application layer, and not at the network. Hence, the problems faced by iRODS is similar to the problems faced by xrootd and ESGF sections. Some of the additional problems faced by iRODS are:

- Fast Data transfer: iRODS provides an option for fast parallel transfers by breaking down the bigger files into smaller chunks. These chunks are then transferred in parallel. However, this approach still transfers between two end hosts and can not exploit replicated datasets. While multi-source retrieval can be supported at the application layer by individually naming each chunk, the underlying naming schema will need to track multiple hostnames and where data is located, making it complex.

- Remote computation: iRODS supports remote execution of a chain of commands. However, these remote executions are bound to a particular host that is running the service. The clients must know the host addresses of where the service is running and send the requests to the specific server. A name based service may be better suited since the users no longer need to know the host address; just knowing the name of the service would suffice.

To summarize, contemporary data management technologies are always built at the application layer, which leads to increased complexity. While much work has been done to alleviate the big-data problem in scientific domains, the solutions often need to work **around** the current network's primitives, and not **with** it, making them complex, community-specific, and hard to maintain and reuse. Additionally, each of these communities painstakingly re-implements every functionality, ranging from data discovery to data retrieval, from data replication to failure recovery, and from access control to security. While some communities implement functionalities that are useful across domains (e.g., fast parallel retrieval in iRODS), no cross-domain reusability exists. Since many of these functions are shared across scientific domains, pushing them into the network stack not only simplifies the applications but also provides a common platform that can be used across scientific communities, reducing development and maintenance costs.

The function of the IP network is to only forward packets. As a result, implementing such functionality in the network is not possible. On the other hand, Future Internet Architectures such as NDN provides robust semantics that can allow these scientific communities to push these functions into the network stack. The following section provides a brief overview of NDN. Later, in Chapter 7 this thesis describes a domain-agonistic data management solution built on top of NDN.

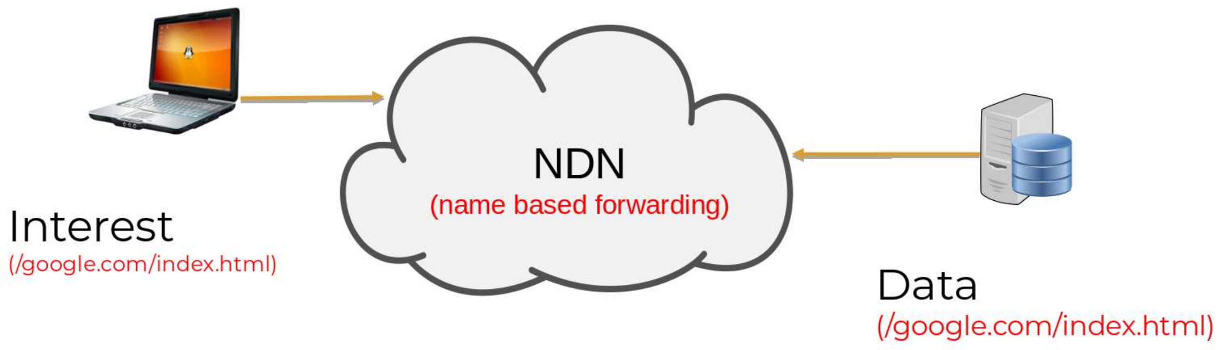


Figure 2.9: NDN Request/Response Overview

2.4 Named Data Networking

This section discusses Named Data Networking (NDN) [196] in the context of scientific data management. Named-Data Networking (NDN) is one of NSF's Future Internet Architecture (FIA) [73] projects that investigate Information-Centric Networking (ICN) . The NSF FIA program supports the investigation of new Internet architectures that are possible replacements for the current host-centric model. NDN is an instance of ICN. In NDN data is accessed by name rather than through the host where it resides. Naming the data allows the network to participate in operations that were not feasible before. Specifically, the network can take part in discovering and local caching of the data, merging similar requests, intelligent retrieval and more. A consumer asks for the content by name, and the network forwards the request towards the publisher. In-network routers also cache returning content along the return path. Requests for data may be retrieved from the original publisher, a repository, a router cache or a neighbor; all content is signed making it easy to for the client to verify that it received an untainted copy. By using extensive caching in the network, NDN allows more efficient distribution (the more popular the content, the more available it becomes) . Additionally, NDN provides mechanisms for in-network failover, fast data retrieval, customized retrieval strategies, and others. Through these features

NDN can significantly benefit scientific communities; the following section provides an overview of the NDN architecture in the context of large science data distribution.

NDN addresses the problems of big-science at the network layer. For example, NDN uses caching and Interest aggregation to efficiently use available bandwidth by reducing request duplication; name-based forwarding adapts to network changes immediately; creative forwarding strategies use multiple paths, and NDN's hop-by-hop congestion control mechanism does not need to slow down consumers if another path is available [157].

2.4.1 Architectural Overview

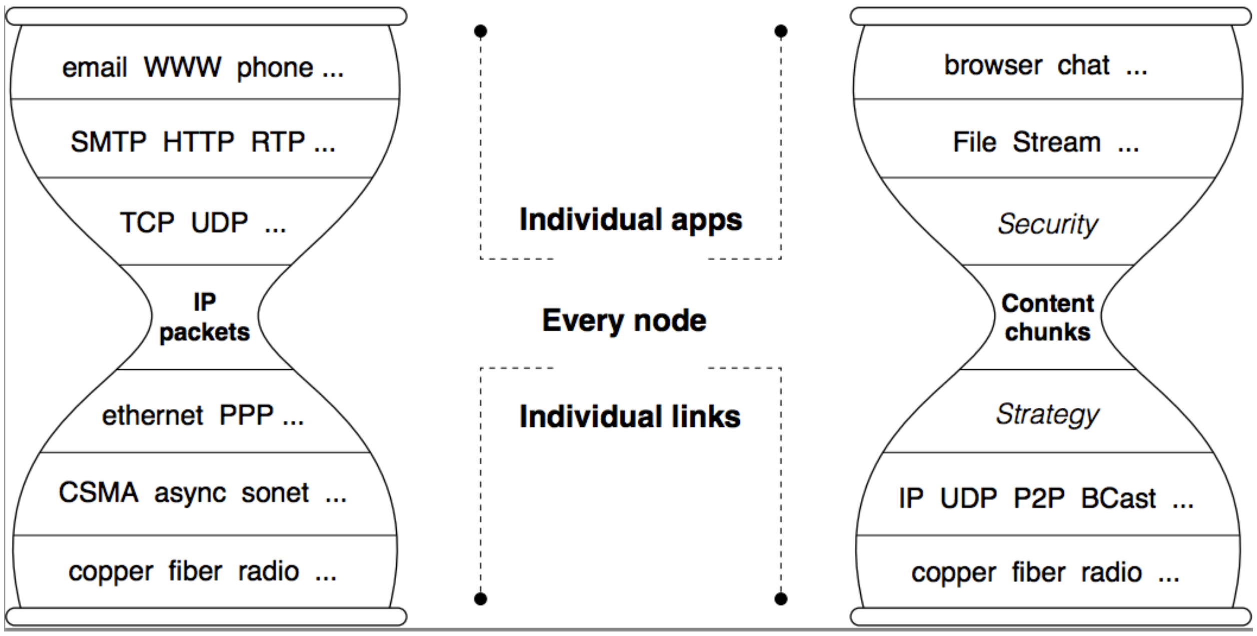


Figure 2.10: NDN in Network Stack
[198]

Similar to today's IP architecture, the thin waist is the centerpiece of the NDN architecture. Like IP, NDN is a "universal overlay" [196] - NDN can run over any transport technology, including IP and the reverse is also true. IP's hourglass architecture makes the original Internet design elegant and successful where a particular layer (IP) implements a minimal set of functionality for forwarding packets. Due to the universal and simplistic nature of this layer, the upper (e.g., application) and lower (e.g., physical) layer technologies were able to evolve without changing the universal packet forwarding mechanism. As Figure. 2.10 shows, NDN retains the same hourglass model. However, today's applications are typically written for retrieving content [196] [74] instead of connecting remote computers. The applications themselves or application specific middleware need to map this content-centric requirement to the current Internet's host-based model. NDN attempts to align these two models by using data names in the thin-waist instead of IP addresses. As Figure. 2.10 shows, using data names instead of IP addresses for delivery offers a new set of minimal functionality based on content names. By changing the semantics of network service from delivering packets to a destination to retrieving content, NDN can essentially eliminate the need for complex middleware and the need for the applications to perform common network functions.

For communication, NDN uses two types of packets, *Interest* and *Data*. The content consumer drives communication in NDN. To retrieve data a consumer sends out an Interest packet, which carries a name that identifies the desired data. One such name might be $/nytimes/frontpage$. A router maintains a name based forwarding table (FIB) (see Figure.) . The router remembers the interface from which the request comes in, and then forwards the Interest packet by looking up the name in its FIB. FIBs are populated using a named based routing protocol, such as NLSR [95]. When the Interest reaches a node or router with the requested data, it returns the content which carries both the name and the content of the data, signed with the producer's signature. This Data packet follows in reverse the path taken by the Interest. Note that Interest or Data packets do not carry any host information or IP addresses, they are simply forwarded based on names (for Interest packets) or state in the routers (for Data packets) .

NDN routers remember both Interests and Data for a while. How long these packets are remembered depends on the router configuration as well as the content validity period set by the data producer. Storing packets in the network allow deduplication of requests and reply. The router uses a Pending Interest Table (PIT) for keeping track of all the Interests waiting for returning data. If a duplicate Interest comes in, the router simply logs the interface in the PIT but does not forward the request upstream. This mechanism essentially aggregates all duplicate upstream Interests. When a downstream Data packet arrives, the router forwards it to all the interfaces listed in the PIT entry, removes the PIT entry, and caches the Data in the Content store, which can be either in router's buffer memory or on the disk. Because an NDN Data packet is signed, the router can store it locally in a cache to satisfy future requests. The content in the content store is replaced based on the local cache replacement policy. Data is returned precisely following the same path, and one Interest brings back exactly one Data packet, archiving hop-by-hop flow balance. Caching and signing of Data packets decouples content from the content producer. NDN can support various functions without additional complexity - it can serve popular content from a router near the edge, create effective multicast groups, and adequately support mobility and delay-tolerant networks using a store-and-forward mechanism.

2.4.2 Hierarchical naming

The NDN design assumes hierarchically structured names, e.g., a video produced by Netflix may have the name /netflix/videos/movie1.mpg, where "/" indicates a separator between name components. The whole video probably will not fit in a single packet, so the segments (or chunks) or the videos will have the name /netflix/videos/movie1.mpg/1..n. Data that is routed and retrieved globally must have a globally unique name. This is easily achieved by creating a hierarchy of naming components, just like DNS. In the netflix example, all movies under netflix will poten-

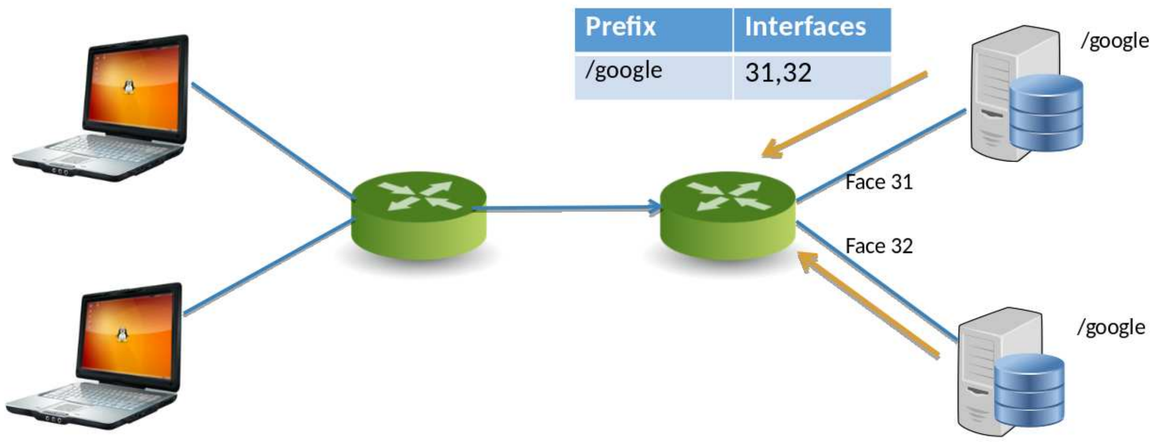


Figure 2.11: NDN Forwarding
[196]

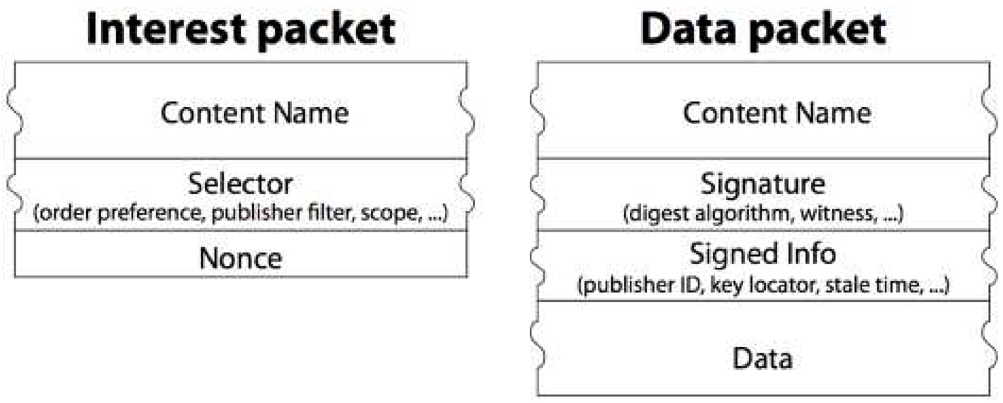

Interest packet
Content Name
Selector
(order preference, publisher filter, scope, ...)
Nonce
Data packet
Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

Figure 2.12: NDN Packets - [196]

tially reside under /netflix; /netflix is the name prefix that will be announced into the network. This hierarchical structure of names is useful both for applications and the network. For applications, it provides an opportunity to create structured, organized names. On the other hand, the network does not need to know all the possible content names, only a prefix, e.g., /netflix is sufficient for forwarding.

NDN places few restrictions on naming, only requiring that, (a) the name structure is hierarchical, (b) naming rules are globally agreed among content users, and (c) name prefixes are allocated to publishers (similar to how the current DNS assigns domain names) . The NDN architecture conceptually divides names into two variable parts,

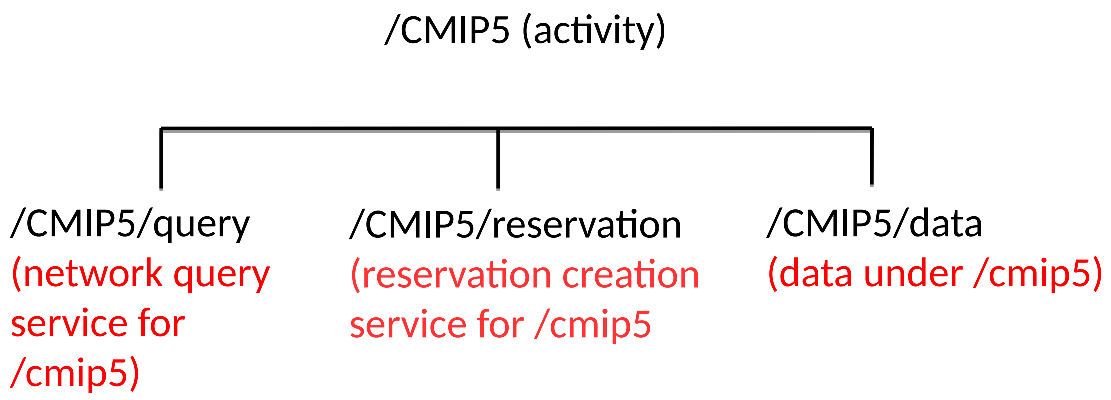


Figure 2.13: An example NDN name

the routing prefix (lbl.gov in Figure. 7.14), stored in router FIBs and used to route Interests, and the application-specific portion of the name. Once a routed name reaches the producer, it can extract the application specific part, such as a query, and return the appropriate data.

Named data enables NDN to automatically support various functions including content distribution, multicast, in-network caching, and producer and client mobility. Since the core component of an NDN based workflow is name-based, it provides a strong incentive to name datasets appropriately, which, in turn, should improve data organization.

2.4.3 Data-Centric Security

In NDN, security is built into the content. Each piece of data is signed by the data producer and is carried with the content. Data signatures are mandatory; on receiving the data, applications can decide if they trust the publisher or not. The signature, coupled with data publisher information, enables determination of data provenance. NDN's data-centric security is helpful in establishing data provenance. Current SSL/TLS based security models place trust in the data provider. However, there is no assurance of data validity if the data producer is compromised. Since the SSL/TSL based model requires trusted hosts, in-network data cannot be cached and reused. NDN's data-centric security enables in-network caching; it is no longer critical where the data comes from since the client can verify the authenticity of the data.

NDN's data-centric security can provide essential security through encryption. Additionally, all NDN objects, including routing announcements, content, control messages can be secured using public key cryptography. Since NDN is data-centric, the widespread DDoS attack in the current Internet is hard to achieve by targeting a particular host. Even if a particular host is attacked using NDN specific attacks (e.g., Interest flooding), consumers can still retrieve data from alternate sources, if such sources are available.

2.4.4 Data Provanance

NDN solves the problem of data provenance, too. In IP, SSL/TLS secures the data transmission channel between the source and the client. Secure data transmission combined with embedded file checksums verifies file integrity and assures authenticity. However, if both the file and checksum are altered at the source, there is no way to detect the modifications.

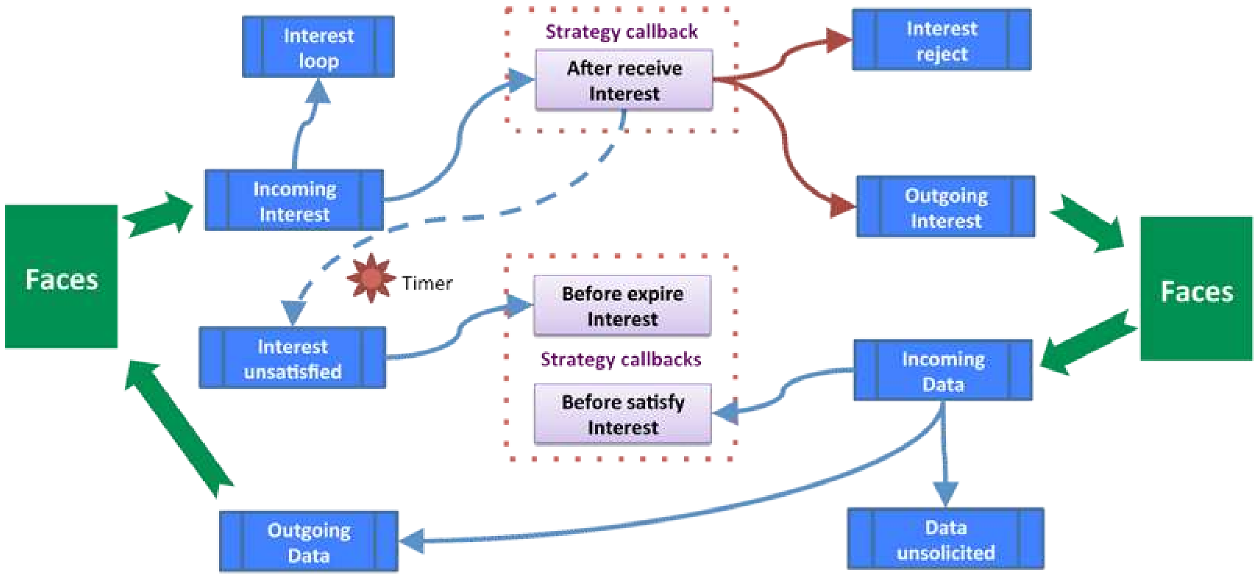


Figure 2.14: NDN Forwarding Strategy [19]

Moreover, with the current workflows, there is no way to prove publisher provenance. While scientific data often includes publisher details, they not signed and therefore has no cryptographic binding to the producer. This might be a problem in scientific communities; data and results from scientific communities have wide applications ranging such as economic forecast and policy-making, and as a result, data provenance and reproducibility of results are very important.

As mentioned earlier, in NDN each data packet has publicly verifiable built-in signatures. NDN makes it mandatory for the data producers to sign data digitally. Unsigned data is rejected either in the network or at the receiving client. This signature is used to verify data provenance at the client and also decouples the content from its original publisher. The receiver can get content from anyone, such as a repository, a router cache, or a neighbor, as well as the original publisher and verify that the data is authentic.

2.4.5 Intelligent Data Plane and Forwarding Strategies

NDN routes and forwards packets based on content names [19], which eliminates various problems that addresses pose in the IP architecture such as address space exhaustion, NAT traversal, mobility, and address management. In NDN, routers perform component-wise longest prefix match of the Interest name the FIB. Routing in NDN is similar to IP routing. Instead of announcing IP prefixes, an NDN router announces name prefixes that it is willing to serve (e.g., /netflix) . The announcement is propagated through the network and eventually populates the FIB of every router. Routers match Incoming Interests against the FIB using longest prefix match. For example, /netflix/videos/movie1.mpg might match /netflix or /netflix/video. Though an unbounded namespace raises the question of how to maintain control over the routing table sizes and whether looking up variable-length, hierarchical names can be done at line rate, previous works have shown that it is indeed possible to forward packets at 20Gbps or more [170].

One of the fundamental features that enables intelligent functionality in the network is multipath forwarding. IP routing uses a single best path to prevent loops. In NDN, the name combined with a random nonce effectively identifies duplicate Interests and stops them from propagating. As a result, NDN can record and forward using multiple paths for the same Interest. This multipath state in the network enables in-network load balancing, in-network failure recovery, and intelligent forwarding of requests.

In NDN, a particular module is in charge of making these intelligent decisions; these are called NDN strategies. NDN strategies are pluggable modules that allow an intelligent processing pipeline for requests and responses. Figure. 2.14 shows how strategies support intelligent data retrieval by creating and acting upon in-network states. Strategies are per-prefix and determine how to process Interest and Data packets based on various parameters such as pre-defined rules, values in a measurement table, or current network condition. As a result, forwarding strategies decide which path(s) to use, based on per-prefix, user-defined policies, a unique feature of NDN that can significantly benefit scientific applications. For instance, if an NDN router has two paths to the same prefix, a strategy can choose the least congested one; or a strategy may deploy BitTorrent-style forwarding for parallel retrieval. Besides, when the only way to guarantee timely completion of large data transfers is to create a reserved bandwidth path [41], NDN strategies can facilitate the creation of such a path [163].

2.4.6 In-network Caching

Automatic in-network caching is enabled by naming data since a router can cache data packets in its content store to satisfy future requests. Upon receiving a new Interest, the router checks if the content is in its local Content Store (CS), and if it is, returns the cached content. The CS is an in-memory buffer that keeps packets temporarily for future requests. However, there is no architectural restriction in making it large. This thesis assumes that in addition to the in-memory CS, there will be other larger, disk-based caches for facilitating content delivery. The content store is similar to router buffers in today's Internet. However, unlike today's Internet, NDN routers can reuse the cached data packets since they have persistent names and the producer's signature. For static content, NDN can serve temporally close requests from the cache. For dynamic content, caching benefits applications using multicast or when packets need to be retransmitted after packet loss.

While caching of content raises some privacy concerns. However, in NDN, the requester is anonymous (though not to the first hop router physically connecting the requester to the Internet) . Additionally, content can always be encrypted for ensuring privacy. Additionally, while NDN names are human-readable, the names do not need to have any meaning themselves (though meaningful names are certainly convenient) . A movie named /netflix/movie1 can also be named as /netflix/b2ee4 without affecting the network functionality.

In addition to the CS, NDN supports persistent, disk-based repositories (repos) [53] [52] [159]. These storage devices can support CDN-like functionality without additional application-layer engineering. Caching can also facilitate intelligent protocols such as in-network strategic caching where a router caches popular content for a longer duration for future requests, freeing up valuable upstream bandwidth. While caching is convenient for applications and enables a rich set of functionality, it is not *the* most critical aspect of NDN. NDN provides other vital features discussed above that are equally important in aligning content-centric applications to the network semantics.

To summarize, this section discussed the problems of large scientific data management and showed in addition to the intelligent applications we need intelligent network layer protocols that can provide the foundation to a generic data management framework. Named Data Networking (NDN) is a future Internet architecture that adopts a drastically different communication model than IP. NDN routes requests based on content names and not by the address of end hosts. NDN has a wide range of potential benefits such as in-network content caching with request deduplication that reduces congestion and improves delivery speed, simplifies applications, and provide data-centric security built into the network. By aligning the network with the application requirements, NDN not only makes the applications simpler but also enable a richer set of functionality that improves content distribution, reduces resource usage, and improve application reusability. The next chapter demonstrates the benefits of NDN for scientific data flows using a real-world scientific data access log; it shows NDN not only optimize resource usage and provide a consistent framework for data management, but it also improves data distribution by automatically providing CDN-like features at the network layer.

Chapter 3

NDN Prototyping for Scientific Data Management

We are entering a new era of exploration and discovery in many fields, from climate science to high energy particle physics (HEP) and astrophysics to genomics, seismology, and biomedical research, each with its complex workflow requiring massive computing, data handling, and network capacities [75] [130] [54] [201]. The continued cycle of breakthroughs in each of these fields depends crucially on the ability to extract the wealth of knowledge, whether subtle patterns, small perturbations or rare events, buried in massive datasets whose scale and complexity continue to grow exponentially with time [30] [78] [75].

As described previously in Chapter 2, despite technological advances, the largest data- and network-intensive programs including the Earth System Grid (ESGF) [57], the Large Hadron Collider (LHC) [26] program, the Large Synoptic Space Telescope (LSST) [59] and the Square Kilometer Array (SKA) astrophysics surveys [66], the Joint Genome Institute applications [172], and many other data-intensive emerging areas of growth, face unprecedented challenges: in global data distribution, processing, access and analysis [99] [130] [76] [101], in the coordinated use of massive but still limited computing, storage and network resources, and in the coordinated operation and collaboration within global scientific enterprises each encompassing hundreds to thousands of scientists [50] [180] [120].

This thesis has hypothesized that NDN, with its network-centric model of data management and transport, can support the needs of big-data applications better. This chapter looks at a contemporary scientific application, the Earth System Grid Federation (ESGF) [12], that presently distributes large amounts of climate data to a global scientific audience. Using ESGF as an example, this chapter presents a study of the NDN network model for scientific data distribution. This chapter shows that NDN can significantly reduce the load on the data producer, provide novel functionality in the network, improve reusability of data, speed up retrieval, and simplify applications through the use of name-based data retrieval, intelligence in the network, and in-network caching. The following section in this chapter describes why the ESGE log is representative of diverse scientific workflows, the details of the log, and the data access patterns, and how NDN can support these data access patterns.

The ESFG log is the most detailed log that this thesis was able to find; the topology generated from the log contained 12,000 nodes and 21,000 links. The log also spanned over three years with 33 million requests. While the author found data logs from both High-energy particle physics and Genomics workflow, they are either smaller in duration or did not have a complete topology and access patterns. However, the study of these limited logs did show that the data access patterns in those communities are indeed similar to the ESGF data access pattern. Due to the completeness of the ESGF log and its representativeness, this thesis replays the actual data requests in the log on a simulated NDN network and studies how NDN can improve data distribution for scientific communities with big-data.

3.1 CMIP5 and the ESGF

The Coupled Model Intercomparison Project (CMIP) [71] is a collaborative framework for studying the output of coupled atmosphere-ocean general circulation models. Phase 5 of this project is referred to as CMIP5. The CMIP5 project facilitates an assessment of the strengths and weaknesses of current climate models and facilitates the development of future models. For example, if the models indicate a broad range of values either regionally or globally, then scientists may be able to determine the cause(s) of this uncertainty using CMIP5 models. The volume of CMIP5 data, approximately 3.5PB, already presents significant data management challenges [174] and the upcoming CMIP6 [71] program is expected to have data sizes that reach into the exabytes, substantially increasing these challenges [135]. The size of this dataset and its distributed nature represents many of the contemporary scientific workflows and makes it ideal for the study. While data access details are subtly different in other scientific workflows [164] [50], for example, climate scientists access individual files while the physicists access datasets (a set of files), basic operations such as publication, discovery, retrieval, staging [161] [166] [185] remains same.

The Earth System Grid Federation (ESGF) is a distributed federation of nodes that publishes and distributes CMIP datasets. Section 2 includes a detailed discussion of ESGF's architecture. Briefly described, ESGF has several nodes around the world that holds parts of the CMIP5 dataset. These nodes are organized as a peer-to-peer overlay. Intelligent applications are used for data discovery, retrieval, and other operations. This section looks at a server log exported by an ESGF node at the Lawrence Livermore National Laboratory (LLNL), which is part of the ESGF federation. This works also looked at another log from an ESGF node at the *German Climate Computing Centre* (DKRZ), the access patterns and the findings from the other log were identical and therefore, not repeated in this chapter.

3.1.1 Data Access Log

This section utilizes an HTTP server log from the LLNL ESGF server that spans from November 2013 to June 2016. Each entry in the log represents a file download

Table 3.1: ESGF logging details - grey rows are used in the simulations

Field	Description
Request ID	Unique ID of the request
User ID	User's OpenID
User Email	User's Email address (Optional)
File URL	Location of the File
File ID	Unique ID of the file requested
Remote Address	IP address of the requested
User Agent	Requester's browser agent (Optional)
Service Type	Type of the Client (Optional)
Time of the request	When the file was requested
Success/Failure flag	Boolean flag denoting outcome
Duration of transfer	How long it took for serving the request
User ID Hash	Hash of User ID (optional)
Data Size	Size of the data on disk (Bytes)
Transfer Size	Bytes transferred

request and contains the filename that was requested. It also contains ancillary information such as the requester's IP address, User ID, request timestamp as the number of seconds since the epoch, the name of the requested file, a success/failure code, and file transfer size. Table 3.1 shows all the fields present in each entry. The request ID is the unique ID of the request used for identifying requests, and the User ID is the user's OpenID for identifying unique users. The user email field is optional and was not used for this work since the userID was sufficient for identifying users uniquely. The File URL field represented the unique HTTP URL for accessing the file. The FileID is a unique ID for each file that was not used since this is a subset of the previous FileURL field. The remote address was the remote requester's IP address. The next two fields, user agent and service type, was optional and mostly not populated, so they were not used for this study. The next three fields, the time of the request, success/failure flag, and the duration of transfer were extensively used, and they provided the timestamps of requests, whether the requests were successfully started, and if they were successful, the duration of the transfer. The userID hash is simply hashes of the UserID and as a result, need not be used. Data Size represented the size of data on the disk (the actual file size), and the transferred size showed how many bytes were transferred. Data Size and Transfer size together pointed out if a transfer was completed fully or partially; if the data size was equal to the transfer size, then the transfer completed fully, and the transfer was partial if the transfer size was less than the data size. A partial transfer can happen when a user kills the transfer midway or due to a network, client, or data server failure.

As the subsequent sections will demonstrate, the analysis of this log provides us with user request patterns, the average size of the files that were requested, the amount of time needed to satisfy each request, and more, as the next sections elaborate. These details allow us to infer the performance of the current system, identify potential bottlenecks, and evaluate NDN's fit with a global scientific workflow.

3.1.2 Request Locations

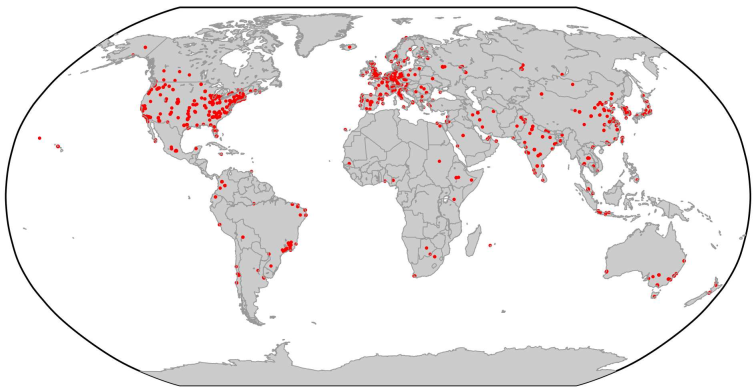


Figure 3.1: Geolocations of requests

The log contains about 18.5 million entries. Each entry represents an HTTP GET request for a single file. The analysis of this log yielded a set of unique IP addresses from these 18.5 million requests, henceforth referred to as the "clients". The analysis found 5692 unique users (derived from the UserID field in Table 3.1 and 9266 unique IP addresses (derived from the Remote Address in Table 3.1) in the log, so clearly users do not always download data to the same machine. The clients were from 78 countries and belonged to 911 individual ASes. The highest number of users in an AS is 683 and 80% ASes had ten or fewer users. Figure. 3.1 shows the locations of these globally distributed clients created using the using Maxmind City Database [123]. The requests came from all over the world with significant portions from Europe (17 percent), North America (25 percent), and Asia (44 percent) .

3.1.3 Request Statistics

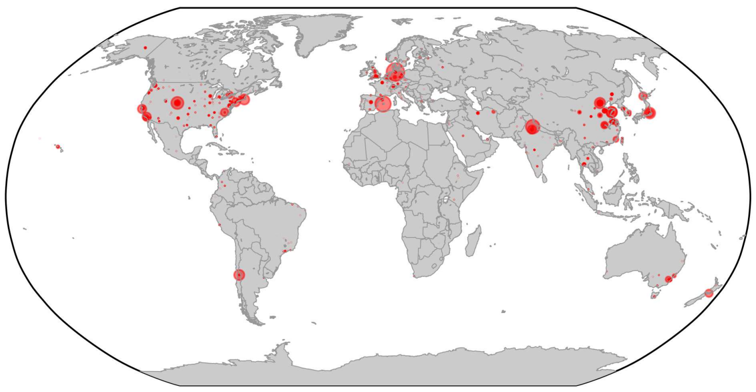


Figure 3.2: Geolocation of failed requests

To better understand the users' demand on ESGF and how efficiently the system handles the request, this work investigates the success and failure rate of the requests. The requests that failed to transfer any data (or transferred zero bytes) are tagged as "failures". The remaining requests fall into two categories: **partial transfers**, where transfer size is less than the requested file size, and **completed transfers**, where transfer size is equal to the requested file size. Figure. 3.3 demonstrates that the ESGF server was overloaded throughout the entire period of study. The bandwidth demand (the blue line in 3.3) on the server was orders of magnitude more than the server was able to serve (the red line), pointing to a significant bottleneck at the server.

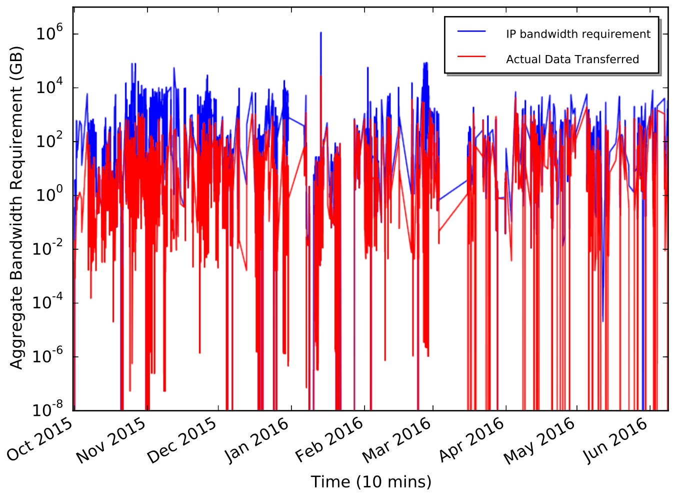


Figure 3.3: Bandwidth Requirement vs Transfer Size

Out of the 18.5 million requests, only 5.7 million are partial or completed (around 33 percent) ; the remaining failed without transferring any data. Since the log only provides a generic error code (-1) upon failure, the precise reason for such a significant number of failures is hard to find. Anecdotal evidence and consultation with system administrators of the LLNL node point toward authentication failures, server overload or user error. The failure heat map as shown in Figure. 3.2 that failures occurred all over the world and not restricted to a particular geographical region. For example, a large number of failed requests were from North America and Europe, regions with historically better connectivity; failures were also noticed in regions with historically worse connectivity. Figure. 3.4 shows that most of the users experienced failure, regardless of their location.

Since the primary goal of this section is studying actual data distribution, and the log does not provide a reasonable explanation for the failures, the failed requests were excluded from the study, but the entries with partial and completed downloads remained. Even extended discussions with the administrators of the node and the scientists were also unable to pinpoint the exact causes of these failures. Removing

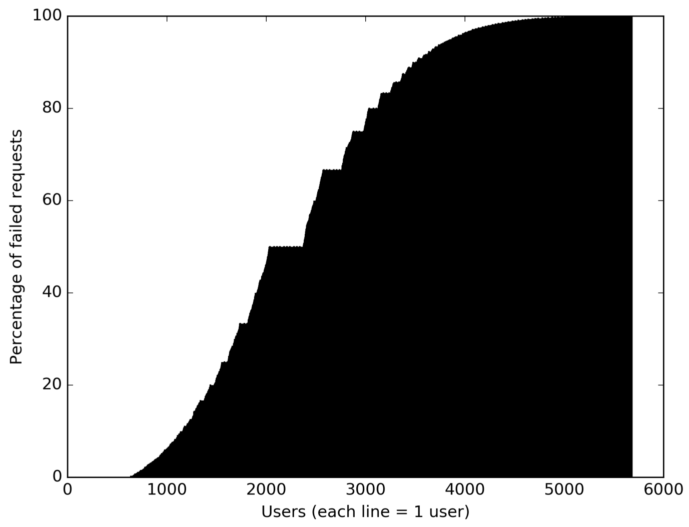


Figure 3.4: Failed Requests by User

failed requests left approximately 5.7 million (henceforth referred to as the *usable request set*) entries in the three-year log. While the failure rate is high, this is the most complete log this thesis was able to find. Additionally, this work has observed failure in other scientific applications in High energy particle physics and Genomics. As this section shows later, the servers are always overloaded which may lead to failures. A complete redesign of the applications is out-of-the-scope of this work. However, the findings from this work point out deficiencies and how they might be addressed, paving the path for more efficient future applications.

3.1.4 Duplicate Requests

Over the three years, the usable request set had approximately 1.8 million unique files; so on average, two out of three requests were duplicate. Figure. 3.5 shows the percentage of duplicate requests by week; in some of the weeks, the number of duplicate requests was between eighty to ninety percent.

One potential explanation for such a high percentage of duplicate requests might be interrupted, or partial requests since partial requests might trigger repeated re-

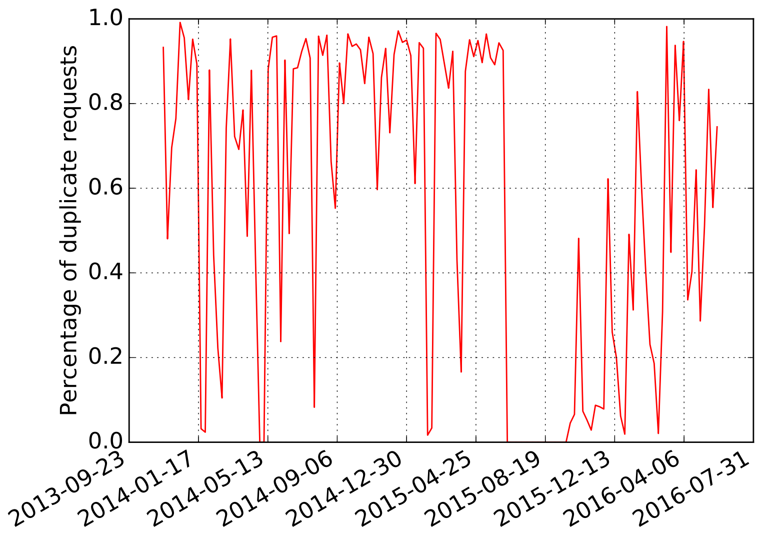


Figure 3.5: Duplicate Requests by Week

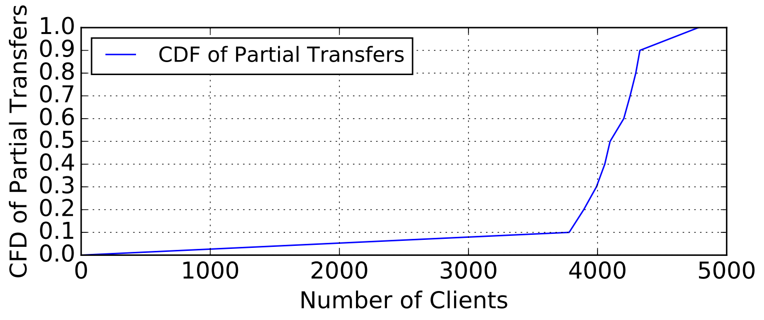


Figure 3.6: Classification of clients based on number of partial transfers

quests when a user retries requests that stopped midway. Figure. 3.6 shows partial transfers across clients, sorted by the percentage of partial transfers. The figure visually classifies clients into three categories: (a) approximately 3500 clients who successfully retrieved data about 90% of the time, (b) 500 clients with widely varying success rates ranging between 10% and 90%, and (c) approximately 500 clients who did not complete transfers about 90% of the time. Plotting the number of duplicate requests from these three groups in Figure. 3.8 confirms the earlier observation: both successful, as well as clients with partial transfers, were responsible for duplicate requests, though the number of duplicate requests was more for the latter group. Additionally, as figure 3.7 shows, the duplicate requests came from clients that experienced a high failure rate as well as from those who did not.

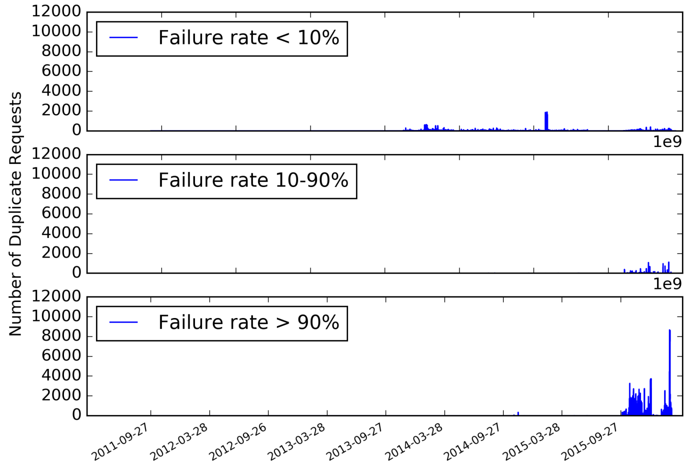


Figure 3.7: Duplicate Requests by Failure Rate

Another possible explanation for duplicate requests is a user requesting different parts of the same file in parallel. An investigation into the repeated requests from the same user showed that the combined size of the temporally close requests for the same file never adds up to the requested file size; they were either much larger

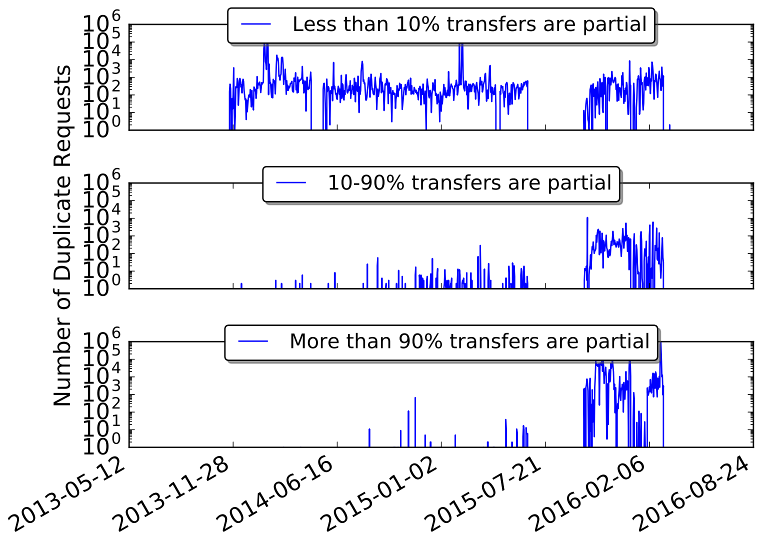


Figure 3.8: Daily duplicate requests from three client groups. Clients are grouped by percentage of partial transfers

or smaller. Other plausible explanations for duplicate requests are, file popularity, space/memory constraints at the user, possible trial runs before actual retrieval or failure at the client after retrieval. None of these conjectures can be confirmed without access to user logs and possibly real users. Therefore, this study treats duplicate requests as legitimate and do not try to guess users' intentions. However, there have been several studies [111] [67] that reaffirm locality of access in the scientific workflows. Additionally, more recent works by the author of this thesis on High-energy particle physics and Genomics workflows show a similar trend of the locality of access. Therefore, though the exact reason for the high number of duplicate requests unknown, this access pattern is well documented [166] [67] [111].

3.1.5 Request Size Distribution

Figure. 3.9 shows the file size distribution observed in the log. While the cumulative transfer size for scientific data is Petabytes or more, individual files are in the range of 30MB to 3GB, and 95% of the files were 1.3GB or less. 95% of the clients typically requested less than 10TB of data over the period covered by the log. These trends

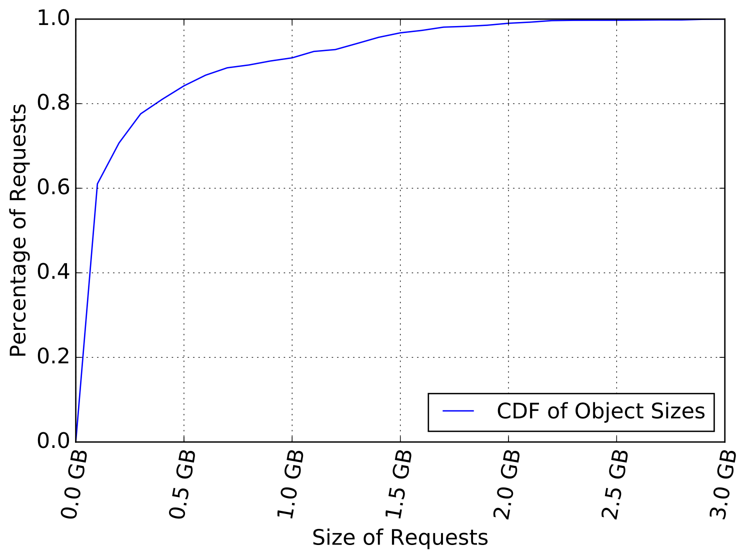


Figure 3.9: File size distribution

are consistent across scientific domains - subsequent works done by the author show that the average file size for High-Energy Particle Physics is around 2GB [166], and around 500MB for Genomics. These observations have implications on caching, and Section 7.25 shows that caching popular datasets in scientific domains do not necessarily require large in-network caches.

3.1.6 Duplicate Request Inter-arrival Times

Figure. 3.10 shows that duplicate requests are closely spaced in time. Specifically, more than 60% of the requests were repeated within a minute and 95% median inter-arrival times are less than 400 seconds. There were hardly any duplicate requests with an inter-arrival time more than 500 seconds. The short inter-arrival time makes these datasets ideal candidates for caching. Moreover, this observation shows that setting the content freshness time, the time that defines how long cached data is valid for fulfilling subsequent requests, to around 500 seconds is optimal for this workflow. Note that this is not the caching duration but provides an estimate the amount of cache needed in the network. Since duplicate requests are closely spaced, the total

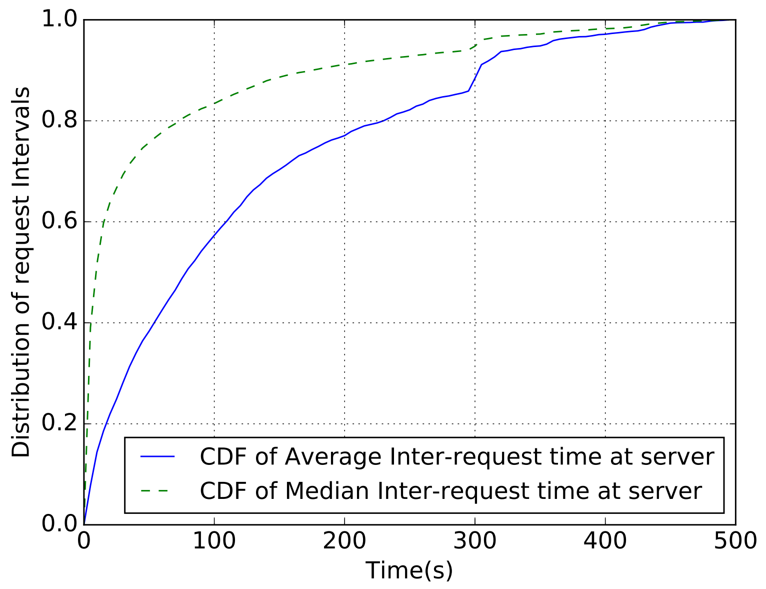


Figure 3.10: Inter-arrival time between duplicate requests

amount of cache required for effectively aggregating requests can be smaller, around 500GB for this specific workflow, certainly feasible today.

3.1.7 Request Frequency Distribution

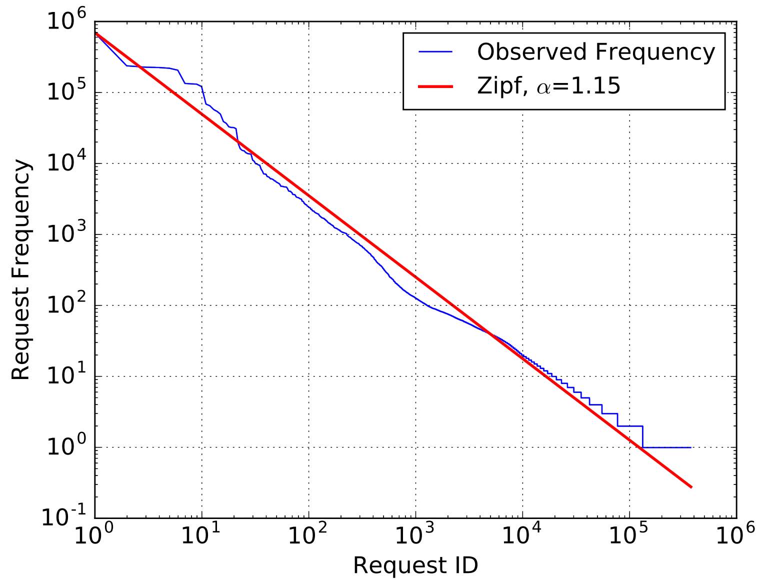


Figure 3.11: Popularity distribution of datasets

The request pattern for individual files follows a very long-tailed distribution with 98% of the files requested less than 20 times. However, Figure. 3.11 shows that some files were also prevalent, the highest number of requests for a single file was 700,000. The red line in Figure. 3.11 shows a Zipf distribution with $\alpha = 1.15$. The observed data request pattern follows an expected distribution [40] where some of the datasets are extremely popular followed by datasets with diminishing popularity. This observation can be useful for carrying out similar experiments or in constructing NDN traffic generators. The request distribution combined with inter-arrival time distribution can also produce a realistic traffic flow for future studies.

3.2 Simulation Setup

The previous section describes the analysis of a three-year log of ESGF data accesses as observed at a single server at LLNL. This section uses those analysis results to create and drive a simulation to determine the benefits of NDN in this major domain application. This work uses the user requests in the log and the derived topologies to drive NDN based simulations using NDNSim [121]. The simulation scripts are available on Github [9].

These simulations use approximated network topologies derived from the log containing all the clients. The network topologies were generated using a machine in the same subnet as the server at LLNL and the tool *traceroute* [118]. Reverse traceroutes from the server to the clients discovered all the hops that were then combined to create the final topologies. Figure. 3.12 shows a week's topology. Note that this work used different topology for different weeks; though the data producer remains unchanged, some of the clients and intermediate routers change for each week.

Clearly, this approach maps the *reverse path* of the requests which might not be the exact routes these requests took [102]. However, without access to *all* clients' ma-

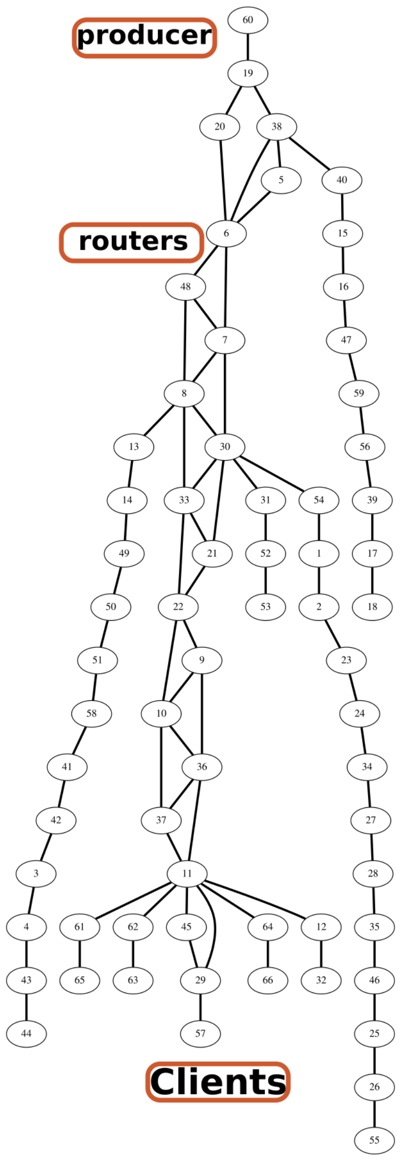


Figure 3.12: Sample Anonymized Topology for Simulations

chines, this is the best topology that could be generated. While a reverse traceroute-generated topology may not accurately represent the original topology used by clients, the actual topology is not crucial for this study. The generated topology is adequate to draw valid conclusions since this study only studies how caching and request patterns affect content distribution. Some network paths are likely to be different as well as the path lengths might be different. This work also assumes that the paths have not changed significantly over the last few years. Again, an accurate approximation of the internal network proves to be less critical in caching since most benefits come from edge caching. As long as workflows have duplicate requests and the network paths they follow intersect, NDN based in-network caching, interest aggregation, and intelligent strategies can simplify and optimize them.

The topology generated from the reverse traceroute results has approximately 12,000 nodes and 21,000 links. This topology was imported into NDNSim [121] and the NDN stack was installed on each node. NDNSim uses real data packets, and the total memory usage for a simulation grows linearly with the number of nodes (Total Memory Usage = Content Store Size X Number of Nodes) . As can be expected, the first attempt to run the scenario on a machine with 128GB RAM proved to be challenging as NDNSim quickly exhausted the available memory.

At this point, it was evident that a compromise was needed. The two choices were either to downsample the entire log using Poisson sampling or use a smaller portion of it to drive the simulations. Previous work has demonstrated that Poisson sampling of web access logs can sufficiently reduce the size of the data while preserving the original characteristics [136] [137]. However, downsampling the log alters request inter-arrival times and may result in inaccurate cache volume and caching duration estimates. Thus, a smaller portion of the log had to be used. Seven weeks out of the three-year log were randomly selected - the best NDNSim could do; These weeks were chosen randomly and were not cherry-picked based on some property, for example, most traffic. Figure. 3.13 shows request patterns, and actual data volume for the selected weeks, and the percentage of duplicate requests per week over the entire log. While simulating only seven weeks is a limitation in the study, figure 3.13 demonstrates that the qualitative observations remain valid since the chosen weeks had very typical request patterns and transfer volumes.

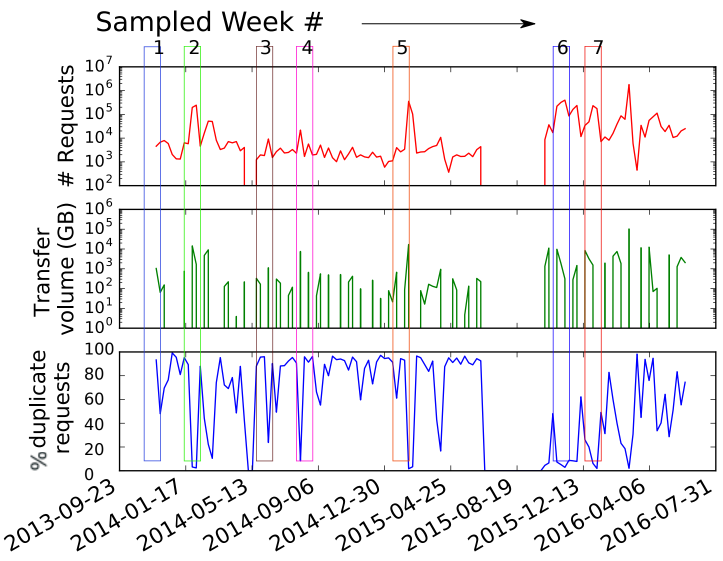


Figure 3.13: [Requests by week]Number of Requests, transfer volume, and percentage of duplicate requests for the whole log at weekly intervals. Colored boxes show sampled weeks. Actual weeks are W1:2013-11-19, W2:2014-01-29, W3:2014-5-21, W4:2014-07-17, W5:2015-01-29, W6:2015-10-01, and W7:2015-12-15

After selecting the target weeks, the next step was to create the corresponding topologies (one for each week) from the log, which are trees with the LLNL node at the root. However, the available memory was still unable to accommodate the simulation, and the simulations needed a lower number of events. Figure. 3.11 shows that a few clients made most of the requests, followed by a long tail of clients that made very few requests. This long tail added a significant number of nodes in the topology that stressed NDNSim. By keeping the clients responsible for 95% of the traffic and pruning the long tail, the number of nodes in the topology came down to a point where NDNSim could produce results.

What exactly did the simulations lose by removing the long tail? The tail (5%) consists of mostly unique requests. Looking at them more closely, these requests contain a small number of duplicates and thus had little effect on Interest aggregation. Given their relatively small number, these requests would not have a significant effect on caching either.

The simulations scheduled the requests in relative time based on the first request's timestamp ($ts_{first\_request}$) . A request r_1 with a timestamp 1378400495 is scheduled at time 1378400495 - $ts_{first\_request}$. This simple manipulation of timestamps keeps the simulation running time low but does not alter the original request pattern. The simulations start with cold caches and do not account for the cache warm-up time. However, this only underestimates the usefulness of caching and Interest aggregation since using a full cache would serve some of the "warm-up" requests from the cache. Figure. 3.11 shows the content popularity trend. Statistically pre-populating the caches/PITs would have resulted in more cache hits and Interest aggregation for the popular content.

In the simulation, the server returns a portion of the data as a signed binary content object upon receiving an Interest; this is referred to as a *chunk*. The content producer defines the size of this returned NDN object; this particular study sets the chunk size to 100MB. The producer also records the Interest names and the time when it received them.

The NDN pending interest table (PIT table) size is unlimited for these simulations. This table allows NDN to aggregate duplicate requests that are temporally close. The cache size varied depending on experiments and are described with each experiment.

Since a well-accepted congestion control mechanism is still lacking in NDN [147], these simulations used clients with a fixed Interest window size of 64, since the work

Table 3.2: Details of topologies by week. Legends are: FT:Full Topology, N:Nodes, L:Links, C:Clients after pruning, RQ:Requests, URQ:Number of unique requests, USR:Unique Users

	N	L	C	RQ	URQ	USR
FT	11570	16618	-	5724796	370623	2942
W1	142	152	20	4174	3935	19
W2	66	78	10	241452	4558	8
W3	119	124	15	1722	1658	15
W4	168	175	20	2632	1366	19
W5	1320	145	20	3118	3036	18
W6	76	79	10	105607	3097	11
W7	85	87	10	14282	570	8

found to be most performant. The log drives interest sending, and clients use Interests created using the NDN names followed by monotonically increasing segment numbers. These simulations use the file names from the log as-is; the NDN name for file x is $/cmip5/x$ for this study. A client may request data for file x using Interests starting with segment number 0 ($/cmip5/x/segment_0$) through segment number n ($/cmip5/x/segment_n$), with each segment bringing in a fixed amount of data. Original file size and NDN data chunk size set by the producer determine the total number of segments. Clients record Interest and Data names, request and reply timestamps, the number of hops, and Nacks or timeouts if any. On receiving the Interest, the producer returns a portion of the data as a signed binary content object. The producer also logs the Interest names and the times when they were received. Once the simulations complete, the resulting logs provides enough data for evaluating NDN's performance. Specifically, the simulations provided the following:

1. Request time and completion time of the requests that enable us to calculate transfer times
2. Requests that were forwarded to the server; this allows us to calculate the number of requests that reached the server

3. Requests that were aggregated in the network and Requests that were served from the cache, this shows how NDN can improve data distribution
4. Hop counts of each fulfilled requests that show how far requests traveled before being fulfilled either from a cache or the data server

3.3 Evaluation

This section evaluates NDN capabilities such as Interest aggregation, caching, and forwarding strategies in order to quantify how the performance of a system such as ESGF might benefit from them. This section shows that an NDN based network not only reduces the load on the server but also eliminate unnecessary network traffic, and speeds up data retrieval.

3.3.1 Interest Aggregation

NDN aggregates pending requests using the Pending Interest Table (PIT) . Such aggregation works if requests arrive before data comes back. Under normal network conditions, the round-trip time is small, which limits the usefulness of Interest aggregation in some applications [63].

The log tells a slightly different story; the log had several occurrences where the server saw repeated requests for the same data within a very short period. Since these requests are temporally very close, the first experiment investigated if the default PIT aggregation in NDN offers any benefits. To ensure in-network caching does not interfere with the study, the content store size was to zero on all nodes, including the producer and the clients, and content freshness time to 10 seconds. The simulation produced a graph showing the results of default Interest aggregation; Figure. 3.14 shows the number of Interests that reached the server. Three weeks saw no reduction in server hits, one week saw a minimal reduction, and the three remaining weeks saw a significant amount of reduction.

The weeks that saw no reduction had a relatively small number of requests, mostly unique. A small number of requests means that (a) the server is not very busy and can serve content very fast, and (b) the average time between requests is on average longer, which explains why there was no aggregation benefit. Weeks that saw a substantial reduction had a high number of duplicate requests, caused by many partial transfers and popular content requests. For example, in one week two clients requested 7,800 unique files for a total of almost 45,000 times.

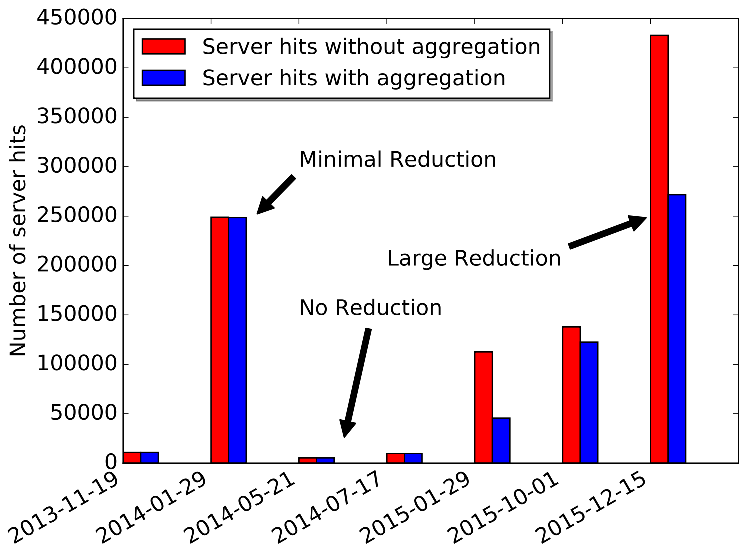


Figure 3.14: Effect of Interest aggregation: Requests that reach the data producer. Cache Size = 0

To further evaluate Interest aggregation benefits, the experiment calculated the reduction in the number of hops Interests traversed as a result of aggregation. This is accomplished by first calculating the total number of Interest hops across all clients without aggregation and then comparing it with the number of hops with aggregation. As Figure. 3.15 shows, some weeks saw a significant reduction in hop count in the weeks where aggregation was helpful.

A few nuances in this experiment should be mentioned here. To ensure timely completion of the scenarios, large chunk sizes (100MB) were used. While larger transmission delays improved aggregation by a small amount, using a large chunk size is both reasonable and beneficial for scientific data and also reduces NDN's signing overhead and improves throughput [166]. The retrieval time, hop counts, and the number of server hits also decrease linearly when the chunk size is increased. Finally, the experiments did not consider competing traffic; unless the PIT table is full, background traffic does not affect Interest aggregation.

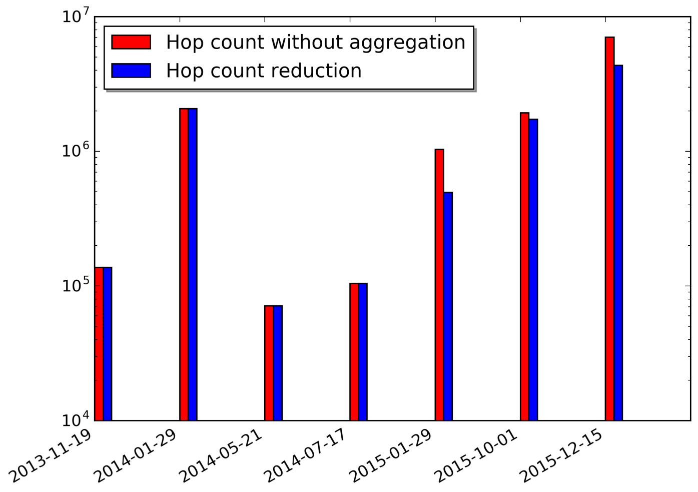


Figure 3.15: Number of hops saved using request aggregation. Cache Size = 0

The simulations show that there is little or no benefit from aggregation alone (without caching) when the number of duplicate requests is small.

3.3.2 Caching

After investigating the benefits of Interest aggregation, this section evaluates the combined effect of Interest aggregation and caching. In the experiments, the cache size was in the range 0 to 10,000 slots. Note that NDN uses the available number of cache slots as the unit of caching. For example, if a node has 100 slots available, it can accommodate 100 data packets in the cache. This experiment uses a leave-copy-everywhere caching policy, i.e., every node on the data path caches the data packets for an object. All nodes in the topology had the same amount of cache, and the content freshness time was set to infinity, so the validity of content does not expire. This means any content in the cache is valid, but content can still be evicted as the caches become full.

This experiment did not model background traffic in the simulations for several reasons. First, there are no accurate models for scientific traffic, so any artificially created background traffic would not be accurate. Second, NDN lacks mature congestion control models [147], and therefore these experiments exclude them. While the community has proposed some congestion control algorithms [147], [157], it is still an active research area, and none has been widely adopted. Finally, while caches would undoubtedly interact with other applications, in science networks and large applications such as climate or HEP, reserving cache space for specific applications is plausible, just like scientific communities reserves bandwidth [82]. These simulations used caches up to 1TB, which can be easily provisioned today, even per-application for a few large applications. However, as noted earlier in this section, much smaller caches, in the order of gigabytes, will be useful in reducing network traffic and huge in-network caches may in-fact be an overkill.

Figure. 3.16 shows the reduction in server hits when caching and Interest aggregation both are used **together**. The figure shows that a small cache is effective in reducing server hits and, as expected, the number of server hits drops as the size of the cache increases. However, the figure also shows that hit reduction is not proportional to the cache size. While a small cache is effective in reducing network traffic, there was a clear trend of diminishing return as cache sizes increase. The amount of cache needed to reduce traffic will depend on the traffic volume and request pattern.

However, Figure. 3.16 shows promising results: even a 1GB cache can provide a significant reduction in server and network traffic. The amount of reduction also depends on the traffic pattern. For example, in the week of 2014-01-29, caching reduced the number of server hits from 248,899 to 13,013. On the other hand, the week of 2013-11-19 saw a minimal reduction and the same amount of caching reduced the number of hits from 10,917 to 10,855. Caching not only reduces server load but benefits clients by decreasing the number of hops required to retrieve content. Figure. 3.16 shows the hop reduction due to caching.

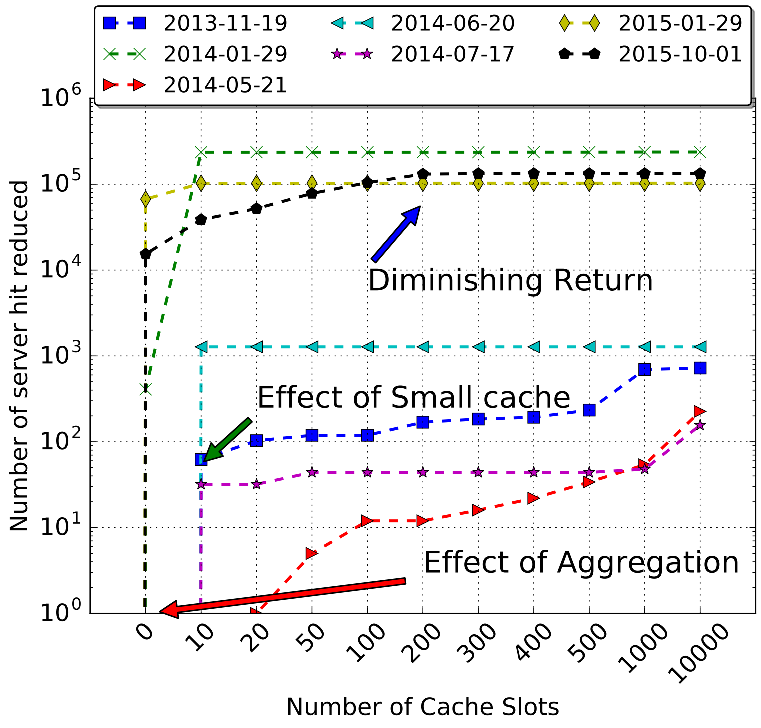


Figure 3.16: Request reduction at the server

3.3.3 Where to Cache?

Having established that caching and aggregation are useful, the study now investigates how cache placement influences data dissemination. The study first divided the nodes into two categories, edge nodes, which act as clients and network nodes,

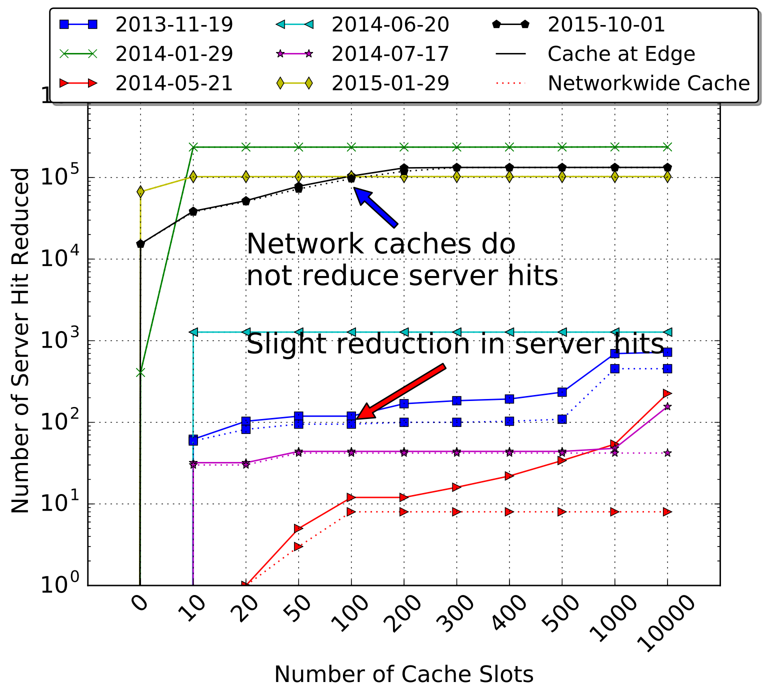


Figure 3.17: Cache at the edge vs cache everywhere

which act as NDN routers. The study installed caches on the edge nodes but not on the network nodes and repeated the simulation described in Section 3.3.2. Figure. 3.17 compares edge caching and the cache-everywhere policy described earlier.

The experiments found that the cache-everywhere policy performs slightly better than the cache-at-the-edge policy. This is not surprising; the cache-everywhere policy can serve multiple clients while cache at the edge serves only clients at each edge. However, the experiments also found that the cost of this improvement is very high. Figure. 3.18 compares the cumulative amount of cache used in these two scenarios. The cumulative cache volume of the cache-everywhere policy is consistently 7-8 times higher than cache-at-the-edge policy. Thus, for this application, caching at the edge provides a good compromise. This observation has immediate applicability - scientific nodes can deploy edge caches and improve their dataflow. However, NDN provides added opportunities to create dynamic caches at network hotspots which might be near the edge or at the core (at the border of an ISP or a country) and more self-reliant than an application layer cache.

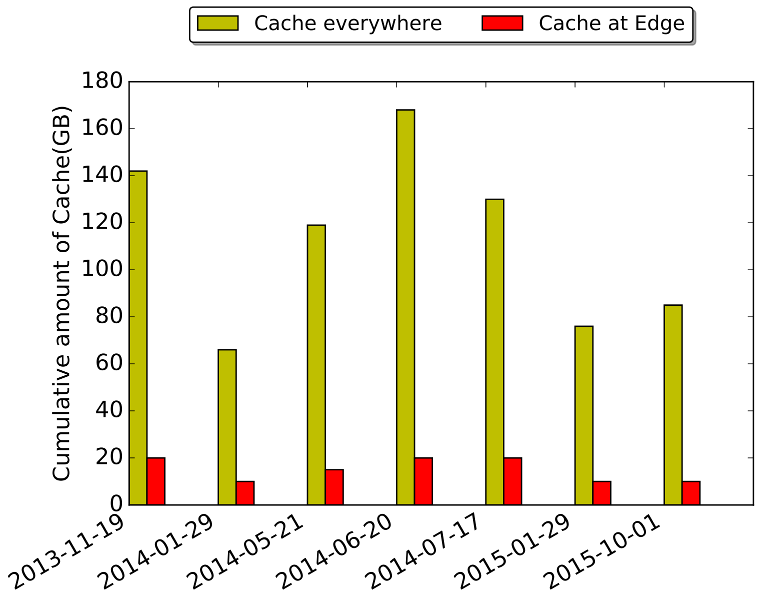


Figure 3.18: Cost of caching at the network vs at the edge

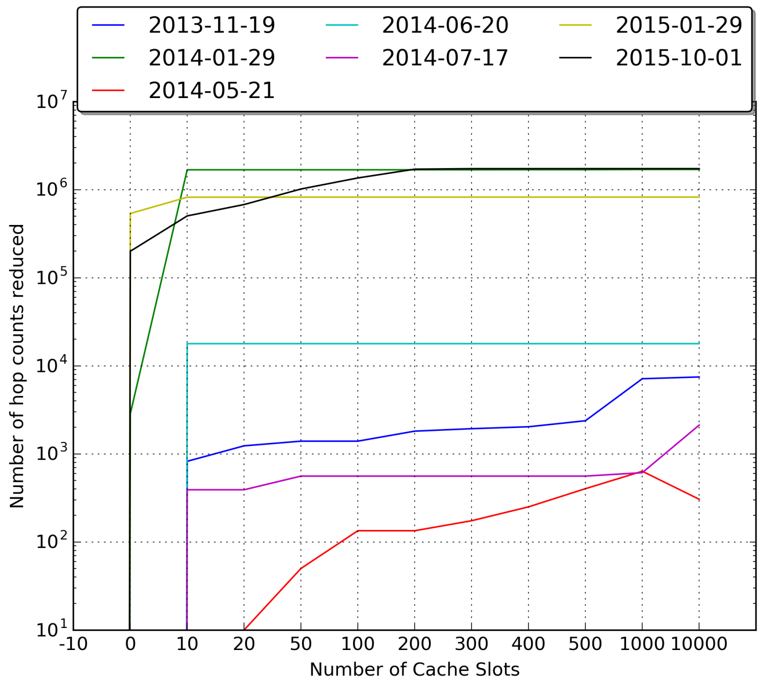


Figure 3.19: Hop reduction by Cache

3.3.4 How Long to Cache?

Duration of caching impacts data distribution and the amount of cache needed in the network. To find out how long data should be in the cache, this study calculated the inter-request time between duplicate requests. Inter-arrival times are calculated as follows - let's assume a consumer requested three pieces of content, $/a$ at time t_1 and t_2, $/b$ at time t_2 and t_3, and $/c$ at time t_1 and t_3. Inter-arrival time for $/a$ is $t_2 - t_1$, $/b$ is $t_3 - t_2$ and $/c$ is $t_3 - t_1$. By repeating this calculation for all unique files at all clients, this study is able to find out the client side inter-request times. Inter-arrival time can predict how long the content object is useful in in-network caches; keeping the content less than the optimal time will increase network traffic and keeping them longer will not benefit the clients but will consume cache space. Figure. 3.10 showed the inter-arrival time to be around 400 seconds for 95% of the duplicate requests. Data rate along with caching duration can predict the approximate cache size for a particular use case; in this case, caching all content on a 10Gbps link for 400 seconds would require a 500GB cache, certainly feasible today.

To summarize, this section makes several important observations:

- For this workflow, caching at the edge provides benefits that are very close to network-wide caching. This study shows that even a 1GB cache at the edge can significantly reduce network traffic. Network-wide caches work better than edge caching in some cases but at the expense of a massive increase in total cache volume across the network.
- Small caches at the edge seem sufficient to improve data distribution, and there is a limit beyond which the system sees diminishing returns. For this workflow, the size of sufficient cache volume is around 500GB.

- Finally, content from this workflow does not need to be cached for a long time. Data request patterns for this workflow is localized, and content only needs to be cached for a short period, around 400 seconds.

3.4 A CDN-like Strategy for ESGF

CDNs provide many services to clients in addition to data delivery [177] [98]. For example, they provide data replication and direction of client requests to the nearest replica [155]. NDN can provide similar services using forwarding strategies. This section presents a hypothetical scenario where data is replicated over multiple servers around the world and made available over an NDN network.

As mentioned earlier, ESGF provides $wget$ scripts [188] for data download that tie users to a specific server. The existing system does not automatically distribute requests to the nearest replica. For example, many requests in the log came from India and China though there are geographically closer ESGF replica servers [188]. Assuming the requested datasets are also available from these replicas, NDN can transparently choose these nearer servers. Selecting a far-away data producer affects these clients, and others, negatively since TCP based protocols do not work very well in high-bandwidth, high-delay networks [10]. Slow download speed also means that users may cancel their downloads and try again, taking up valuable bandwidth and server resources [81].

NDN is capable of addressing this problem at the strategy layer [164]. For ESGF or a similarly distributed system with multiple replicas, a simple strategy that chooses a server near the clients should improve data distribution. This work investigates how NDN can achieve this as follows: the study first amends the topology to add five more data producers; two in China, one in France, one in the US, and one in Germany. Locations of these replica sites are matched to the actual ESGF data servers. The study then updated the topologies using the method described in Section 3.1.1. The study then deployed an NDN strategy that finds the lowest latency path from a client to its nearest data producer as follows: on receiving the first Interest, the strategy multicasts the first Interest over all matching Faces. After receiving replies, the strategy ranks the faces according to latency, for subsequent requests. The operation is relatively lightweight since one multicast Interest is required at each node for an entire namespace. For example, after calculating the lowest latency route for $/cmip5$, Interests named $/cmip5/a$, and $/cmip5/b$ are both forwarded over the same route. In real networks, the nodes will repeat this periodically and update route rankings – this study simply calculated the routes once at the beginning. This study then simulated traffic flow for the week of 2015-10-01 with only one server, followed by the same simulation with multiple servers/replicas.

Figure. 7.8 shows the new request distribution among the servers. In the original log, server 6 was the only producer and served 100% requests. With the new strategy, server 6 received only 0.03% of the requests. The remaining requests were redirected to other servers closer to the clients. For this week, most of the requests were from China. The simple strategy redirected 96% of the requests to the two servers in China (server 1 and 2), freeing up resources at the LLNL node.

Choosing the server with the lowest latency also reduces the delay at the clients. Figure. 7.10 shows the mean delay at each client. Most clients saw a reduction in mean delay, but the level of reduction is also interesting; the mean delay for Client 3 was reduced from around 200ms to around 25 ms, an order of magnitude improvement. Other consumers also saw a significant reduction in latency.

This experiment demonstrated how a simple NDN strategy could provide sophisticated, CDN-like services to scientific data distribution systems. The ability to automatically select a server with low latency can significantly improve large data dis-

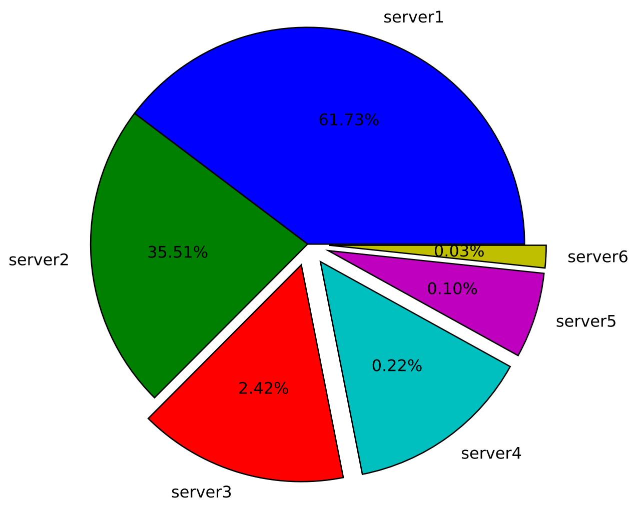


Figure 3.20: Percentage of requests served by each server. Server 6 is the original data producer.

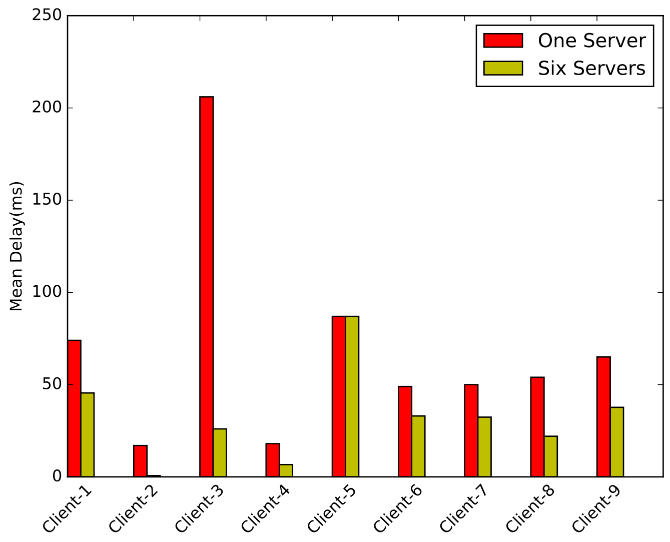


Figure 3.21: Mean delay of retrieving a data packet at clients

tribution by increasing throughput, improving robustness and lowering distribution costs.

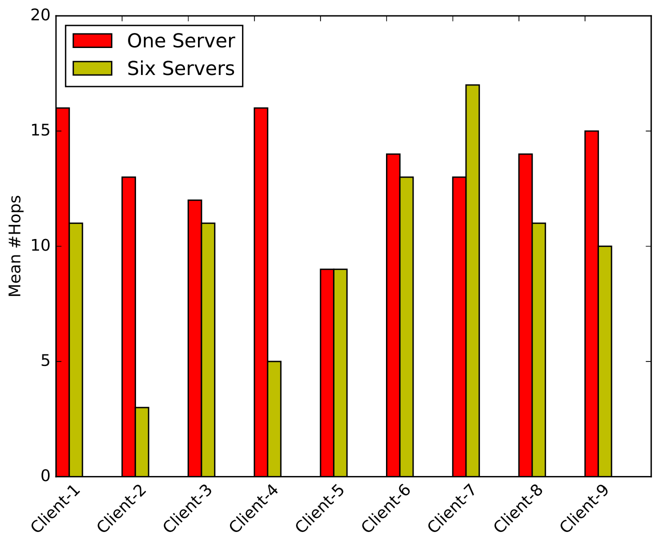


Figure 3.22: Mean number of hops at clients

3.4.1 Summary

To summarize, analyzing a 3-year data access log from ESGF, a federated system for distributing climate data, this study demonstrates the benefits of NDN for scientific workflows. These analyses bring out several observations that help to define various parameters for improving data distribution for scientific workflows:

1. The analyses show high-level of duplicity exists among user requests, and therefore aggregating temporally close requests can potentially reduce the load on the server and the network.

2. This study also characterized the data access patterns. This study shows that while climate data is very large, the average file size is small, around 1.3GB. This study also demonstrates that requests are highly localized and can benefit from

NDN's in-network request aggregation. The request distribution contents a few very popular files, followed by increasingly less popular content.

3. While network caching is useful for popular datasets, caches do not have to be large: small caches at the edge may significantly reduce network traffic because, despite the overall data aggregate being huge, individual files are small. This observation suggests that large caches at every network node might be unnecessary for scientific workflows. Instead, caching at the edge provided the best trade-off, and even a 1GB cache could significantly improve data distribution. Large data did not translate to long caching times either; only a few minutes was sufficient. The useful lifetime of data in in-network caches was also low, around 400 seconds.

4. Figure. 3.10 shows that the average inter-arrival time of duplicate requests is around 300-400 seconds. This observation means requests shows a high degree of spatial and temporal locality, a strong argument for network caching. It further means that long-lived caches in the network might be unnecessary, allowing us to reduce in-network cache sizes.

5. Finally, using a simple, latency-based forwarding strategy, this study shows how NDN could provide nearest-replica retrieval with very low overhead. This simple strategy could reduce the load on the data servers while reducing latency at the clients.

All these observations are good news for ICN and align well with the properties of ICN. The following sections use NDN simulations to investigate and quantify ICN benefits for this particular workflow. The results from this work will help both the NDN and climate communities. The climate community may benefit by incorporating NDN or its concepts into their current distribution tools. On the other hand, the NDN community can benefit from a real-life example to help guide NDN research and development. The study may also help create NDN-based tools, such as an NDN science traffic generator and help designing congestion control algorithms. The next chapters build on this study and show the necessary steps for integrating NDN with actual scientific workflows.

Chapter 4

Testbed Deployment

In order to develop NDN based applications and network protocols, the NDN community created a generic global testbed [18] where the community could investigate and develop various aspects of NDN, such as security, application support, routing, and forwarding. However, the generic testbed lacked resources and isolation needed to investigate many important research and implementation aspects of the network architecture at scale, such as caching, routing, scaling, performance that is required for big-data science applications [166]. Additionally, the generic testbed among global institutes lacked control and the flexibility of instrumentation of software stacks, required coordination among the participants for installing and maintaining software stacks. Additionally, the experimental nature of big-data experiments required installing modified versions of NFD [19] and the ndn library, ndn-cxx. Installing experimental versions of these core software on a shared testbed inconveniences other participants and affect the quality of results collected from various experiments.

For continuing network research for big-data without hindering other participants and allowing dedicated resources necessary for such research, NSF funded a small but dedicated high-performance testbed [166] that could handle the scale of big-data science. Figure. 4.1 shows the topology of the testbed. The proposal allocated resources for a campus NDN deployment between the Computer Science Department and the Atmospheric Sciences department at Colorado State University. The goal was to look at scientific data management problems in collaboration with the climate scientists, identify the critical problems facing the scientific communities, and

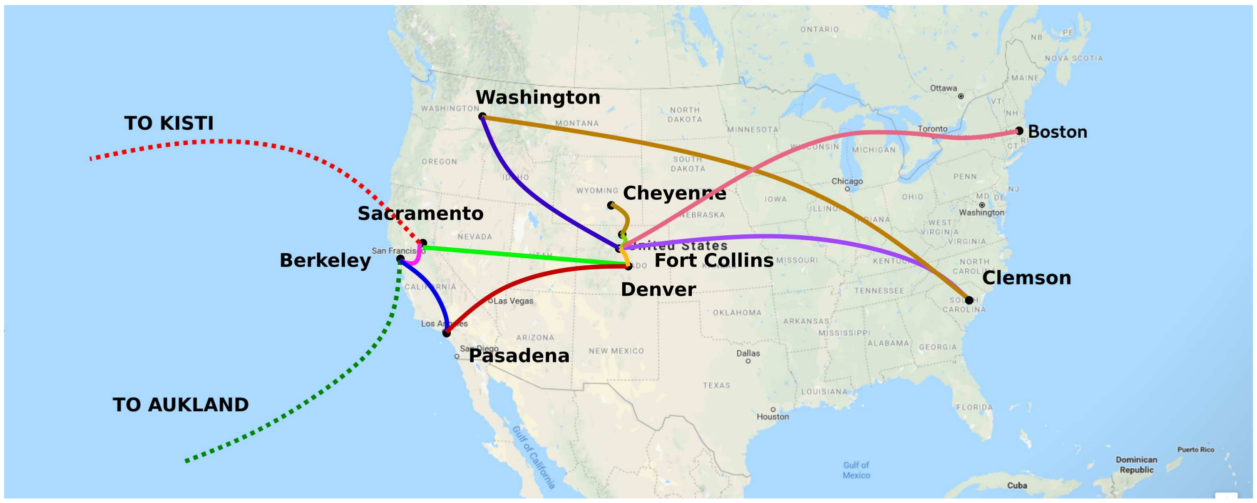


Figure 4.1: NDN Science Testbed

investigate an NDN based framework that can adequately address these problems. The climate community was ideal as the initial partner since they face enormous data management issues; datasets in the terabyte to petabyte range, management of distributed auxiliary information such as annotations, metadata, and other well documented problems emanating not only from the data size, but also the exponentially increasing user population and lack of a proper management framework. The reader should note that the testbed alone was not enough for evaluating all protocols developed during this work - this work routinely and extensively used NDNSim [121] when hundreds of nodes were needed for experiments. However, the testbed proved to be a valuable instrument for translating simulations to real-world deployments, albeit at a small scale.

The primary objective of this exercise was to enable research into both of a new networking architecture as well as its interaction with a challenging application in Big Data. This research provided with useful tools and protocols for the big-data community as well as created experience with a new, content-oriented network paradigm. Beyond the initial goal of the project, the testbed has proved to invaluable in developing, debugging, and deploying state-of-the-art NDN protocols open to the community. As the following sections will show, the testbed is still a valuable resource and has expanded to other communities such as High Energy Particle Physics, Genomics, and others. Additionally, other institutes, such as Clemson, KISTI, and Washington State University have joined the testbed by creating nodes at their respective institutes.

As this thesis has discussed in Chapter 5, NDN is able to help with the scientific data management issues by naming content appropriately. Users can locate and retrieve the required data by merely asking for it by name, thus integrating naming and storage – the network will not only locate the nearest copy but will authenticate and ensure the data integrity has not been compromised. The testbed provided an opportunity to test these protocols, looked into the challenges of deploying them, and how to address these challenges in the context of large scientific data.

The testbed cumulatively hosts over 70TB of climate, HEP, and genomics data that this work used for research, experimentation, and development of NDN-SCI [72], a distributed data management framework. The following section discusses the hardware and software used to create this testbed and enumerates the lessons learned from this exercise.

4.1 Equipment

The initial deployment of the testbed started with six dedicated nodes. These nodes were high-end machines with 20 cores each, 128GB RAM, 48TB disk space, and multiple 10Gbps network interfaces, both copper and optical. Since most of the machines were being outside the Colorado State University, the machines also had remote management ports enabled. Table 4.1 summarizes the configurations of these machines. The newer nodes that joined the testbed had upgraded hardware. For example, the Caltech and the Northeastern node have several SSD drives that are used as front-end caches, and the network ports on these nodes are also 40Gbps or more. These diverse configurations helped perform a variety of experiments. For example, the initial nodes were used to develop Ethernet transport for NDN that does not require an IP overlay and the nodes with SSDs are still being used to develop hierarchical caching schemes.

Table 4.1: NDN Testbed Hardware Specifications

CPU	2 x Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz
Core Count/CPU	10
Thread Count/CPU	20
Memory Capacity	1536 GB
Installed Memory	128 GB
PCI Slots	6
1 Gigabit Ethernet Ports	4
10 Gigabit Ethernet Ports	2 x Intel Corporation Ethernet X520
Hardware Raid Controller	DELL PERC H710
Remote Access Controller	Dell IDRAC
Hard Disk Drives	Dell 4TB
Hardware Raid Version	60

4.2 Connectivity

To address the transport needs of the applications while stressing the limits of the new architecture, this work deployed a 10Gbps network in the Colorado State University campus between the Computer Science and the Atmospheric Science departments. The network supporting big-data was separated from other campus traffic to minimize disruptions. Additionally, new optical fibers were deployed between the data centers to ensure seamless connectivity. Figure. 4.1 shows the locations of these nodes.

Outside Colorado State, the testbed currently utilizes existing infrastructure of several ISPs. The connectivity between CSU and Wyoming is provided by the Bi-State Optical Network or BiSON [92], a private high-speed optical network that connects the University of Wyoming to the Front Range GigaPop (FRGP) [93] in Denver, Colorado. The FRGP (www.frgp.net) is a network peering location in Denver and is operated by a consortium of research and higher education institutions that includes the University of Wyoming, Colorado State University, University of Colorado - Boulder, University of Colorado - Denver, Denver University, Colorado School of Mines, and the National Center for Atmospheric Research [126]. FRGP provides a peering point to the Commodity Internet as well as specialized research networks such as Internet2 [175] and Department of Energy's Energy Sciences Network (ESnet) [4]. All of these links have atleast 10Gbps available bandwidth that is isolated from commercial as well as other scientific traffic.

At Denver, the testbed joins ESnet [4]. A node at Denver, along with the nodes at Sacramento and Berkeley are connected using 10Gbps links and separated from commercial traffic using separate VLANs. This separation of traffic provides predictable round-trip times as well as eliminates the possibility of network congestion. The node at Caltech is connected over the Corporation for Education Network Initiatives in California (CENIC) [131], at 40Gbps. The node at KISTI is connected over a trans-pacific 10Gbps link provided by the Pacific Wave [17]. The node in Otago, New Zealand is connected over a 10Gbps link over the Kiwi Advanced Research and Education Network(REANNZ) [62].

Over ESnet, the testbed possesses the ability to instantiate additional on-demand Layer 2 Paths using the On-Demand Secure Circuits and Advance Reservation System (OSCARS) [82]. OSCARS allows network users to reserve time and resources on the high-speed networks, essentially allocating dedicated resources for large data transfers. OSCARS eliminates the need for manually configuring individual network devices for creating a dedicated path with reserved resources. Additionally, OSCARS provides the ability to provision backup resources in case of a network failure or unforeseen outages [82]. OSCARS has been useful in experiments where interaction between various layers of networking was required, for example, when the forwarding daemon did not find enough resources to fulfill a request and needed to instantiate a new path [165]. This thesis discusses these protocols in more details in the next Chapter.

4.3 Software Stack

Each of these testbed nodes run either CentOS7 or the latest version of Fedora as the operating system with all the latest patches applied. Besides, the machines are behind firewalls to prevent unauthorized access. Access and disk space are granted to the collaborators upon request. Additionally, monitoring, maintenance, and provisioning software are also installed on the testbed, but these are not specific to this testbed and therefore, are not included in this discussion.

The network stacks on these machines are tuned based on ESnet's recommendation for large data transfers [10]. Table 4.2 shows the testbed specific modifications. Since the most common network mode for NDN is an overlay over TCP or UDP transport, these modifications were necessary for achieving higher throughput. Additionally, this work contributed code towards creating Ethernet transports for NDN, so that NDN could function without an IP based underlay [19]. However, the Ethernet transport, along with congestion control and loss recovery algorithms in NDN is still work-in-progress. As a result, the testbed and the experiments this work performed extensively utilized NDN over TCP (and UDP) .

The testbed uses the latest version of NDN Forwarding Daemon(NFD) [19] and the NDN library (ndn-cxx) . However, similar to the default TCP/UDP stacks, this work made some modifications to the NDN stack as well. The first modification was to modify NFD and the ndn-cxx library to publish and receive large data objects. TCP

Table 4.2: NDN Testbed Network Stack Parameters. Most of these values were taken from the ESnet's guide for fast data transfers over Wide Area Networks [4]. The values were then manually examined and tuned to achieve highest possible throughput on the testbed. With these values, data transfers between two nodes were able to achieve 9.8Gbps over the 10Gbps links using TCP. NDN throughput was lower as described below.

Parameter	Variable	Size	Rational
Max Socket receive buffer	net.core.rmem_max	128MB	Default Socket Buffer size is too small [4]
Max Socket send buffer	net.core.wmem_max	128MB	Default Socket Buffer size is too small
Linux autotuning TCP read buffer limit	net.ipv4.tcp_rmem	4096 87380 33554432	Scale the TCP sending window for long paths
Linux autotuning TCP write buffer limit	net.ipv4.tcp_rmem	4096 87380 33554432	Scale the TCP receiving window for long paths
Default congestion Control algorithm	tcp_congestion_control	htcp	Experimentally found to work better than CUBIC [4]
Check if Jumbo Frames are enableds	net.ipv4.tcp_mtu_probing	1	Avoid MTU black holes [108]
Default Queuing Mechanism	net.core.default_qdisc	Fair Queuing	avoid overrunning a slow receiver [97]
MTU Size	-	9K	Performs better for both TCP & UDP [28]
UDP CPU afinity	-	Set to a core not used for Interrupt handling	UDP at 10G is CPU limited [4]

parameters along with large packets provide much better throughput than the default parameters provided by NFD and the Linux network stack.

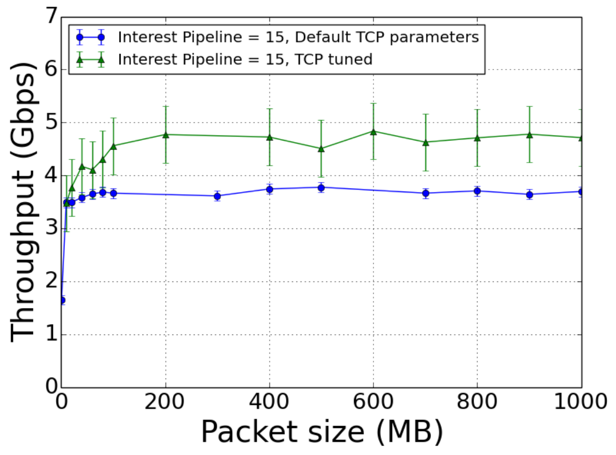


Figure 4.2: NFD and TCP Optimization for better throughput

Specifically, the following optimization improved application throughput on the testbed:

1. Large NDN packet sizes: The default packet size of NFD is 8800 bytes [19], too small to enable the high throughput required for scientific data. Experimentation on the testbed found that NFD throughput increased up to a packet size of 300MB. In most of the subsequent experiments for large science data, large packet sizes were used.

2. Pre-signing packets: The packet signing cost of NDN [166] is expensive, and in order to keep this from slowing down the data transfer, application packets were pre-signed. Since most data needs to be signed during publication, this is a realistic compromise. Additionally, for most scientific applications, data signing is a onetime operation, and once data is published, there is no recurring cost. However, this observation does not apply to dynamically generated datasets, and those operations would have to incur the delay of packet signing.

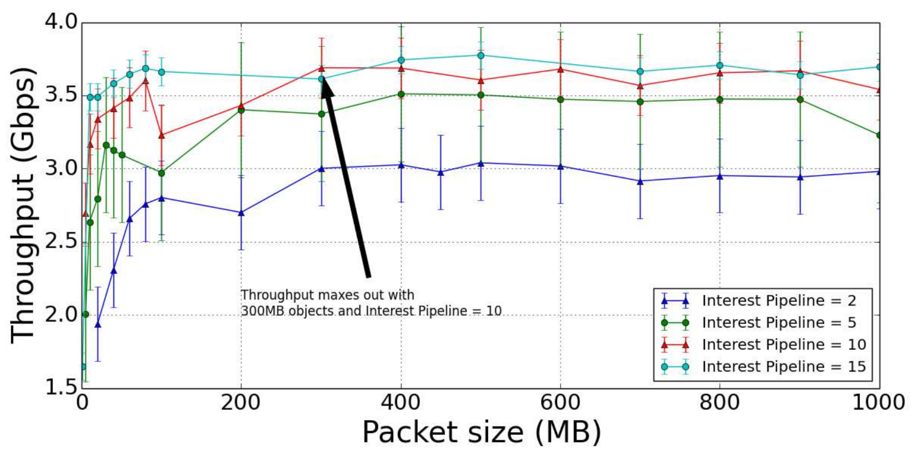


Figure 4.3: Pipelined Interests for better throughput

3. Need for a congestion control protocol: Originally, NFD's default pipeline size was 4 Interests. The experiments over the testbed showed that this value is too small for large data, achieving only 400Mbps throughput [166]. The use of larger pipeline size, along with other changes (described below) improved NDN's throughput dramatically, up to 4Gbps. Several experiments [166] observed that either a larger pipeline or a new NDN layer congestion control is needed to improve application throughput, leading to new research and implementations of congestion control algorithms for NDN networks [157]. With the above optimization, NFD was able to achieve around 4Gbps throughput with an NDN packet size of 300MB and a pipeline of 10 [166].

4.4 Lessons Learned

In establishing the testbed, a number of valuable lessons were learned. This section enumerates them below.

- Network tuning makes a huge difference in performance: The default network parameters supplied by the Operating systems and Networking stack are not adequate for high-speed transfer of big-data. Each parameter that affects the network must be set very carefully - a task that has been made easy by the tuning guide from ESnet [4]. However, many of these parameters need to be further tuned based on the topology, delay of the paths, and application requirements.

- Identifying NDN's bottlenecks: While this is expected of an evolving network stack, running big data experiments over the testbed pointed out several bottlenecks in the NDN codebase which included creating multiple copies of the same packet [166], huge signing overheads [19], and single threaded processing of packets [19]. All of these bottlenecks have been reported to the NFD developers and some of them has since been addressed.
- Development of Ethernet Transport for NFD: The integration with OSCARS [82] required developing an Ethernet transport for NDN. The testbed was invaluable in developing and testing these protocols since long-distance Layer 2 circuits are hard to find in regular intra-institute deployments. The NDN community and the author of this thesis have successfully demonstrated that NDN can successfully run without needing an IP underlay, a massive step in demonstrating NDN's feasibility as a future network architecture [164].
- NDN over UDP is not suitable for large data transfers: Several attempts were made to deploy NDN over UDP with the goal to bypass TCP's congestion control algorithms. However, it was soon evident that some congestion control algorithms are needed at the application or the NDN layer. Without any congestion control algorithms, severe packet loss was observed, which led to new research in NDN congestion control algorithms [157].
- Configuring long-distance networks is work-intensive: Configuring a high-speed testbed with the help of multiple stakeholders is a tedious process. First, the operators needed to agree on address allocation, correctly configure the VLANs, debug any connectivity problems, and even run new optical fiber ca-

bles. Once the basic configurations were done, troubleshooting any intermediate routers required coordination with various network operators. Additionally, even when options for dynamically creating paths were available, they often failed without sufficient feedback. Creating this global testbed underlined how configuring a network a very manual, labor-intensive, and inflexible process. Cutting-edge network provisioning and configuring methods, possibly with the use of Software Define Networking along with the next generation packet forwarding/routing mechanism, such as those provided by NDN, can work together to make the network more flexible.

In the next chapter, this thesis discusses the NDN-based protocols that were developed and deployed over this testbed. These protocols demonstrate how NDN can benefit big-science data in a real deployment scenario. More importantly, the testbed provides the opportunity to demonstrate shows how next-generation architectures such as NDN and SDN can work collaboratively to create a complete in-network framework for supporting large data distribution in various scientific communities.

Chapter 5

Naming Scientific Datasets

Scientific communities have a long history of collecting massive amounts of diverse data over tens of years and from numerous teams [178] [33] [119] [172]. It is not uncommon for a research team to run extensive simulations on supercomputers or generate data volumes that reach into the petabytes [178] [47]. These communities have a long history of sharing data among scientists around the globe, which means they are constantly challenged by discovery and distribution problems [12] [119]. Indeed, scientists often still share data by shipping physical disks around despite the availability of 10G and 100G links [55] [106]. While there are several reasons why scientific communities cannot fully utilize such high-speed links, part of the problem is ad-hoc data management and inconsistency in naming [166].

The previous chapter shows that content retrieval over NDN is beneficial to scientific workflows, and the ability to request content directly using content names has some significant advantages [161]. For example, NDN-supported requests to repositories can be substantially simplified and automated, with requests issued transparently to repositories that have the data without the need for application infrastructure to locate them (and users to establish login credentials, log in to various portals, search, and issue targeted requests) [72]. Data can be fetched from the nearest location, often a boon to collaborative efforts [163]; and finally, data can be published by the simple act of naming it [72]. Content can be retrieved from any available location: the publisher, a repository, a router cache, or even a friendly neighbor. Additionally, the network is able to support intelligent functionality in the network such as retrieval from the nearest data source, retrieval using multiple sources simultaneously, and others [161].

This chapter illustrates how to convert existing content names in large and demanding application domains, specifically, climate modeling, high energy particle physics, and genomics, to NDN-compatible formats. In the process, the chapter also discusses the current naming conventions in these communities, the trade-offs between various naming choices, and the author's experience of converting existing names to NDN compatible names.

In addition to demonstrating naming schemes for these significant categories of scientific datasets, this chapter also describes how to write translators to convert dataset names from existing, ad-hoc namespaces into NDN compliant names. The exercise in translating existing names is not only crucial for supporting legacy applications but also for creating naming guidelines for future applications, and this translation is not always straightforward. The lessons learned from this exercise are not only applicable to scientific communities but also broader use cases.

The contribution of this section is to demonstrate how the naming problems for a name-based network can be approached. In addition to creating a generic framework for translating existing names into NDN compliant names, this section discusses what should be the considerations for designing such names. Additionally, this section shows how naming affects basic operations of a name based network such as routing, discovery, and retrieval. Finally, this section provides general naming guidelines for future scientific applications.

5.1 Existing Naming in Scientific Communities

Currently, scientific content names can be divided into two categories. In the communities that have not established a naming convention, content naming is ad-hoc. In communities with established naming conventions, files are named according to these conventions and stored in a directory structures that closely follow those particular conventions [139] [178]. Depending on the actual community, naming convention and granularity of data access differ. These naming conventions are often created for convenience - in this case, the file names are human readable to allow the scientists to gather information from the names. Additionally, the directory structures often duplicate most of the information present in the filename. The following sections describe existing naming in three scientific domains - Climate Science, High Energy Particle Physics, and Genomics.

The climate communities usually access individual files [72]. Each of these files has several components that describe various attributes of the data, such as which project generated them, the temporal range of data in the file, the institute name where it was generated, and other. The file name components are often duplicated in the directory structure as well.

High-energy physics (HEP) communities generally utilize "datasets", a collection of files. In this community, there is little importance of individual file names, and they can often be hexadecimal strings [166]. However, the directory structure is critical since it contains vital information such as parameters to expect in the data, when the data was generated, and which experiment generated the data. Not knowing the exact filenames does not impede data access; data consumers can ask for all files under a specific namespace.

In Genomics, files are individually named, just like the climate communities [32]. However, genomics communities use multiple files with the same names but different extensions. The common part of these names is the base portion of the name, and the extensions denote what types of data they contain. For example, one file might contain the base pair of a Genome, and another might content annotations [21]. Since all files in the dataset have the same base name with different extensions, applications that use these names might be able to infer the existence of various files once they know the base name. Being able to use only the base name for inferring filenames is convenient but on the other hand, implicitly assuming the existence of files based on a name might be problematic in some cases, for example, when files are missing.

There are many similarities in how these communities name their data. Names are almost always hierarchical and composed of several distinct components. The components (and therefore the names) are human-readable in most cases. However, there might be one or more component that might not be human readable, e.g., the file names in High Energy Physics communities. Looking at the complete name, a domain expert can identify various information about the file and the data without actually looking into the file.

These are properties that fit naturally into the NDN paradigm. However, all names are not created equal, this thesis has often encountered names where components were arbitrarily ordered in the naming schemas, components were missing or transposed, and components disagreed with the actual data in the files. Though these names are almost immediately usable in an NDN network, benefiting from an NDN network requires consistent naming. For example, a name with a more general component at the beginning (e.g., /CMIP5/CSU/data1) allows NDN to perform transparent failover if multiple repositories serve /CMIP5. However, the same name with components transposed, /CSU/CMIP5/data, is usable in an NDN network but may not provide automatic failover benefits.

Another example is how naming affects caching. Take for example a database that holds the daily temperature of a region. Let's assume the names are in the format - /region/year/month/day. An application trying to get data from January 30th to Feb 2nd, 2019 would request data in the form of /region/2019/01/30 to /region/2019/02/02. These data are easily cacheable in the network. If another application requires data from January 30th to Feb 3rd, 2019, it can simply ask for /region/2019/02/03 and get the rest from the network cache. However, naming the data in the format /institute_name/region/year_month will still work, but will not have the caching benefits described above.

NDN naming, while flexible, also imposes the requirement for consistent naming. By naming data consistently, the communities can not only standardize data management logistics but also can get multiple in-network benefits, such as request aggregation, caching, failover, fast transfers, and others. The following sections provide more insight into current naming efforts and how we can convert existing names to consistent names that can benefit from a name-based network.

5.2 Naming Data for NDN

NDN places few restrictions on naming, merely requiring that, (a) the name structure is hierarchical, (b) naming rules are globally agreed upon among content users, (c) name prefixes are allocated to publishers (similar to how the current DNS system assigns domain names), and (d) names are human-readable providing some additional level of assurance [196]. NDN names can conceptually be divided into two variable parts, the routing prefix, stored in router FIBs and used to forward Interests, and the application-specific portion of the name. NDN Routers currently match the full name when de-duplicating Interests or matching content in their cache but only use the routing prefix to forward Interests.

As discussed earlier, the data names in scientific communities are already hierarchical. In some scenarios, converting the names to NDN names requires simply replacing the delimiters. In other cases, names need to be checked for errors, duplicate components need to be consolidated, and unnecessary elements removed. In most of the cases, the NDN names can be derived from the actual filename and the directory structure. However, not all name components can be gleaned from the directory structure and filename of the dataset. In some cases, consulting the metadata in the files, and in some cases even the data itself to mine for missing name components is necessary.

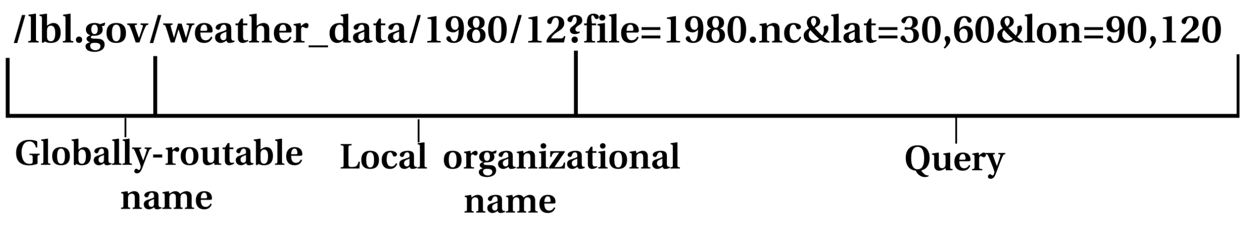
After creating names and naming rules for each community, the author has consulted with domain scientists and confirmed that these naming rules are acceptable for their datasets and are appropriate for global distribution. However, global naming conventions are hard to implement and enforce. This work assumes that each of these communities will have some top-level prefix they will use for publishing their datasets, e.g., /cmip5 for a climate community. These organizations will allocate sub-namespaces as they see appropriate, e.g., CMIP5 many allocate Colorado State a sub-namespace of /cmip5/csu.

Since NDN does not impose any restrictions on naming, the network can support any number of organizations as long as they have unique prefixes. Among scientists in big data domains, there is a general consensus that global namespaces are needed, especially given the current explosion of data [178] [166] [72]. For example, the CMIP5 project [178] that collects datasets for global climate research, prescribes precise specifications and tools to convert local names into CMIP5-compliant names. The Data Reference Syntax (DRS) document [178] describes in detail these specifications.

A naming convention is required to move from the filesystem to an NDN realm. However, this thesis does not propose the renaming of all existing data but a file (or dataset) name to NDN name mapping, so the files are available over NDN with minimal user intervention. The requirements for NDN names compliment natural naming schemas for most data. As an example, a large portion of the climate community utilizes the Coupled Model Intercomparison Project (CMIP5) [178] naming schema. This schema is fully NDN compatible without any changes. Since an essential consideration for naming existing data is to ensure the least disruption to current workflows, the next sections describe (a) how to design names to NDN-compatible names and (b) how to automate existing name to NDN-compatible name conversion with the help of a name translator.

The following section shows three naming scheme and demonstrates how to convert a set of existing names into NDN compatible names. It depicts three naming schema that has been developed and deployed on the testbed - for Climate Science, High-energy particle physics, and genomics, respectively.

5.2.1 Climate Data Naming



/lbl.gov/weather_data/1980/12?file=1980.nc&lat=30,60&lon=90,120
Globally-routable name
Local organizational name
Query

Figure 5.1: Naming a climate dataset into NDN

Climate data names are hierarchical. The NDN names for climate applications can be divided into the routable prefix of the name that has a relatively fixed structure, and the model-specific portion of the name. Both these portions together form the complete name. In addition, as Figure. 5.1, the name might also contain a query portion at the end.

The goal of the routable part of the name is to get an Interest routed to the set of machines that host the data. The following name components are standard across many different climate projects [133].

Activity - defines data collection (or similar) activity

Product - Sub-activity

Organization - Institute responsible for the results

Model - Model used for generating data

Ensemble - Distinguishes among closely related simulations

Experiment or Field Campaign - Name of the experiment

Thus, a routable name prefix may be defined as follows:

/Activity/[Sub-Activity or Product]/Organization/Model/

For example, one such name might start with $/cmip5/output1/csu/MPI-M$, where CMIP5 is the activity, output1 is the product, CSU is the organization, and the MPI-M is the model. This captures the fact that CMIP5 is a globally unique effort, followed by more specific information about the names. In NDN, organizations are free to choose any unique name they want, not just those limited to DNS domains. An NDN network will use this routable prefix for forwarding Interest, retrieving data, and intelligently distributing requests to available CMIP5 data replicas.

For more specific information such as those from the output of a specific climate model, the naming scheme might add additional components unique to the model [178]. These may include the following:

Experiment - the name of the experiment

Start Time - When data collection started

Modeling realm - which high-level modeling component was used

Variable Name - sampling frequency and modeling realm

Ensemble Member - a string used for distinguishing among closely related simulations

Table 5.1 shows the components of one such climate data name - " /CMIP5/output/MIROC/MIROC5/historical/6hr/atmos/psl/r1i1p1/1984010100-1984123118/".

Table 5.1: Name components of a climate data name

Component	Value
Activity	CMIP5
Product	output
Organization	MIROC
Model	MIROC5
Experiment	historical
Frequency	6hr
Modeling_realm	atmos
Variable_name	psl
Ensemble	r1i1p1
Time	1984010100-1984123118

In designing the model-specific portion of the NDN names, this thesis determined that names must include enough information to uniquely identify data. Examples of such information might include models such as CESM [103] and SAM [194], which are other well-known climate models. The model-specific names then might be structured as follows:

CESM /Ensemble/Experiment/Sample Granularity/Start Time

CMIP5 /Experiment/Frequency/modeling realm/ variable name/ensemble member/

SAM /field campaign/optical properties for radiation/grid resolution/output type-/timestamp/

Note that this model specific portion of the names are primarily used by the applications. An NDN network will also use them for caching and request aggregation.

It is relatively easy to see how to extend the names above to support operations such as subsetting. For example, the communities can add a portion (e.g., a suffix) to the NDN name to indicate which variables the subset should contain, for example using key-value pairs such as

```
subset_variable=temperature and latitude = 30,60 and longitude = 90,120$.
```

Figure. 5.1 shows an example of climate data naming. The first part of the name is the globally routable name, potentially an unaltered existing domain name. The second part is the local organizational name followed by a string of query. Note that the last two parts can be encrypted to ensure privacy; only the first part is necessary for routing.

5.2.2 High Energy Particle Physics data naming

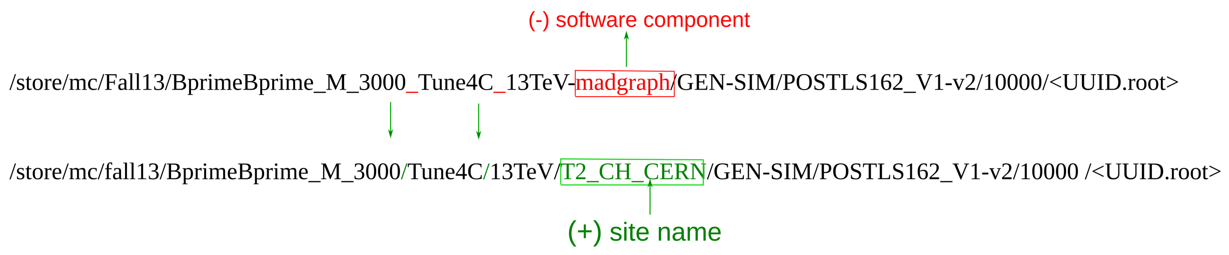

(-) software component
/store/mc/Fall13/BprimeBprime_M_3000_Tune4C_13TeV-madgraph/GEN-SIM/POSTLS162_V1-v2/10000/<UUID.root>
/store/mc/fall13/BprimeBprime_M_3000/Tune4C/13TeV/T2_CH_CERN/GEN-SIM/POSTLS162_V1-v2/10000 /<UUID.root>
(+) site name

Figure 5.2: Naming a root dataset into NDN

Similarly, the High Energy Particle Physics (HEP) dataset names are easily translated into NDN names. The HEP community has been able to agree on specific naming schemes and decide which name components are necessary. In this example, the original name is simply divided into logical segments separated with "/". This exercise (arbitrarily) decided that the software-version element is not needed and can be replaced by a site name. This change, however, does not affect the underlying NDN layer as long as the names are hierarchical.

The hierarchical naming structure of NDN is especially suited for HEP datasets, which are already named using a hierarchical name schema. Here is an example of a name for a collection of HEP data [166]: $/store/mc/fall13/BprimeBprime\_M\_3000/GEN - SIM/POSTLS162\_v1 -$

$v2/10000/ < UUID.root >$. The last component signifies a set of files with a .root extension having a unique ID. The translator takes the HEP name that signifies a directory structure and turns it into an NDN name by removing the "/" delimiters and converting all directory names into NDN name components. The file name is also converted into an NDN component. A user who wants a file with a specific ID can issue an Interest as follows (where the "/" now signifies the NDN name component delimiter) : $/store/mc/fall13/BprimeBprime\_M\_3000/GEN-SIM/POSTLS162\_v1-v2/10000/001.root$. This Interest will return the contents of the file named under the directory structure. However, the full NDN name of the file will be the name expressed in the Interest. Now suppose a user does not know the names of the specific files, but only the prefix. In NDN, the user can issue an Interest with just the prefix: $/store/mc/fall13/BprimeBprime\_M\_3000/GEN-SIM/POSTLS162\_v1-v2/10000/$ and the application at the data producer might return the first object that lexicographically matches the name prefix. Finally, suppose a repository wants to advertise an entire collection of files as available to users. The repository will advertise a prefix that covers the collection, for example, $/store/mc/fall13/BprimeBprime\_M\_3000/$. This advertisement tells NDN to forward all Interests with that prefix to the repository. In the implementation of the HEP translator, it converted the HEP names directly into NDN names because the HEP names are well-formed.

5.2.3 Genomics data naming

Just like Climate, and High-energy Particle Physics (HEP), the Genomics community needs to store and distribute large amounts of data. However, data can be spread around the world, named arbitrarily, and may not always be easy to discover, retrieve and use.

By carefully examining a genomics workflow this thesis has investigated how Named Data Networking (NDN) can facilitate such workflow by augmenting mechanisms such as discovering genomic data repositories around the world, and searching and retrieving data efficiently. For example, the National Center for Biotechnology Information (NCBI) in Maryland, USA, contains 25.5 petabytes of high-throughput DNA sequence data with varying degrees of associated metadata resolution [21]. Genomic datasets eventually make it to public repositories like NCBI, but individual research labs maintain the data before publication and often share data with collaborators through ad-hoc approaches or shared data grids like iRODS (the Integrated Rule-Oriented Data System) [144]. Regardless of where the dataset resides in the data life cycle, the data could be quite useful for distributed research teams and potentially thousands of genomics researchers if the datasets became discoverable.

```
Genome/
[genus]_[species]{_[infraspecific name]}/[assembly_name]
[genus]_[species]{_[infraspecific name]}-[assembly_name].fa
[genus]_[species]{_[infraspecific name]}-[assembly_name].extn
```

Figure 5.3: Naming convention for DNA sequence datasets

Modern genomic DNA data comes in the form of "static" reference genomes with coordinate-based annotation files, and "dynamic" measurements of genome output (e.g., RNAseq data files that contain RNA molecule snapshot strings in the tens of millions of sequence records) [56]. A common aspect of genomics datasets is that they are already named in an evolution-based, hierarchical manner, which is easily mappable to an NDN framework.

This work has translated the names of multi-purpose static genome files and demonstrate how to map them in NDN. Later sections show how NDN helps with locating and retrieving static datasets as well as pushing computations to the edge for generating dynamic datasets on-the-fly.

5.3 Translating Existing Names to NDN names

The problem of organizing science data is hard due to the size, diversity, and number of datasets generated. However, the NDN naming paradigm fits nicely with the existing naming conventions of data. These scientific communities often contain simple rules about how to organize their output, typically in directories and files with meaningful names. This helps manage datasets in local file systems but is not an adequate naming solution for sharing data, a deficiency well recognized by the science communities [133] [166]. Recognizing that it is simply impossible to rename the vast collection of datasets and update all the assorted tools that manipulate them, this work develops translators that ease the burden of migrating existing datasets to NDN. Translators are not always easy to implement; they often need to ingest arbitrary names that do not describe the data well and turn them into NDN names. Often translators need to mine the directory structure, filenames, and potentially file content to compose appropriate NDN names.

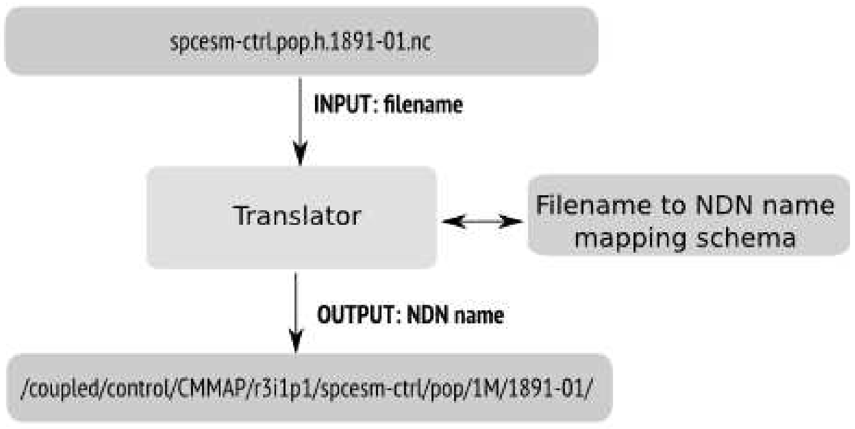

spcesm-ctrl.pop.h.1891-01.nc
INPUT: filename
Translator
Filename to NDN name mapping schema
OUTPUT: NDN name
/coupled/control/CMMAP/r3i1p1/spcesm-ctrl/pop/1M/1891-01/

Figure 5.4: Name Translator

This work envisions that there will be multiple domain-specific translators depending on the data and even the version of the models. Collaborations with domain

```
[Name]

### Target component sequence for \gls{NDN} name ###
ndnMapping = project_id , product , institute_id ,
model_id , experiment_id , frequency , modeling_realm ,
variable_name , parent_experiment_rip , start_time

### Example File Name:
hurs_Amon_C\gls{CS}M4_decadal1961_r4i2p1_196101-199012.nc ###
### The schema describes each component ###

filenameMapping = variable_name , mip_table , model_id ,
experiment_id , parent_experiment_rip , start_time , filetype

### Delimiters for separating the name components ###
seperators = _,.

### This component value  will come from actual data
in the file ###
compsFromData = frequency

### These component values will be read from the metadata
 in the file . Can be used for sanity checking as well ###
compsFromMetadata = project_id , product , institute_id , model_id ,
experiment_id , frequency , modeling_realm

### Manual override of component values , if necessary ###
userDefinedComps = activity:cmip5 , subactivity:atmos ,
organization:csu , parent_experiment_rip:r3i1p1
```

Figure 5.5: A translator Schema File

scientists have shown that translators need information gleaned from the filesystem path, the filename, limited user-provided configuration information, and metadata mined from within the data itself to construct appropriate NDN names. Thus, this work expects the need for some intelligence in the translators.

Metadata determines the behavior of a translator to NDN name mapping. Figure. 5.5 shows how this mapping is accomplished using a schema file. This schema file provides a list of ordered NDN name components to the translator along with instructions for retrieving the data from the directory structure, filename, file content, or user-defined configuration. The translator takes two arguments as input, the full filename and a file that describes how to map the name components to NDN name.

Figure. 5.4 provides an overview of the operation. The schema file is straightforward but requires both domain knowledge and NDN insights for producing NDN names that enable applications to benefit from an NDN network. The schema file provides two lists - the first one describes the name fields of existing data and the second one describes the desired fields in NDN name and their respective orders. For example, the following filename represents a climate data file - $psl\_6hrPlev\_bcc-csm1-1\_historical\_r3i1p1\_198001010000-201212311800.nc$. The filename list in the configuration file will look like [variable_name, mip_table, model_id, experiment_id, parent_experiment, start_time, filetype], components that were used build the file name. Now, let's say assume the scientists want the following changes in the translated NDN name - all names are preceded by a "$/CMIP5$" prefix, followed by [product, institute_id, model_id, experiment_id, frequency, modeling_realm, variable_name, parent_experiment, start_time]. Note that the value of the component "frequency" is not in the file name and must be derived by looking at the actual data. In this example, let us also assume that the "mip_table" and "filetype" fields are unnecessary. The translator would compare both lists and come up with the NDN name - $/CMIP5/output/NCAR/CCSM4/decadal1961$

$/monthly/atmos/hurs/r7i1p1/196101-199012/$. Note that the use of a schema allows name specifications to be shared, and if agreed, the schema can be easily updated to add or remove fields in the derived NDN name. This thesis expects the routing prefix to be part of the schema, so once the NDN name is translated, the content is automatically published under that particular name prefix.

Since NDN naming is flexible and virtually any appropriate translation schema can be plugged into, a translator works across many current naming schemes as long as the existing name can be broken down in hierarchical components, as demonstrated by the domain-specific translators above.

5.3.1 General NDN Discussion

The experiences in naming data for various use cases allow this thesis to generalize naming conventions in NDN. This section presents guidelines about how to name data for future applications that will be compatible with NDN networks and enumerates the considerations for designing such names.

NDN names for scientific data should be both expressive and human readable. The advantages of long, expressive names versus short, easy to remember names need to be considered. Scientists often use the names to understand the content, and the structure of the data, hence maintaining human-readability is essential. At the same time, longer names with many components might be easier to index for searching, but it makes readability a problem. Fortunately, NDN allows multiple names to refer to the same object, and also provides the concept of a "link object" (a redirection from a shorter name to a longer name, or vice-versa), perhaps easing the decision process. For example, a shorter name with essential name components might help the scientists to understand the data, and at the same time, it might point to a larger name with additional components that can be used for cataloging and indexing.

Names need not be restricted to existing, static data items. The names may refer to derived data objects, for example, subsets of existing files, a dataset, which may be composed of sets of many files, or even future data that is yet to be generated. A hierarchical naming pattern in NDN makes this varying level of granularity possible, as long as the applications understand how to interpret the names. For example, a name component can include a lambda function that (see Figure. 5.1) extracts specific data from the raw dataset. In addition to providing for varying levels of granularity, the hierarchical names will also be used in the routing of the data. A data provider publishes a data prefix which acts as a globally routable prefix covering all of the data made available by the provider.

Additionally, naming plays a vital role in routing. The trade-offs of placing components at the beginning vs. at the end should be carefully considered. For example, hosting CMIP5 data under the namespace $/cmip5/organzation\_name$ is simpler to manage. However, organizations participating in the project must advertise this prefix through their routing system. CSU will announce $/cmip5/csu$ and UCLA will announce $/cmip5/ucla$ in addition to their normal prefixes, e.g., /CSU and /UCLA. The alternate approach might be to have /cmip5 as a sub-namespace of participating organizations - such as $/CSU/cmip5$ and $/UCLA/cmip5$. However, this approach creates a fragmented namespace and maintaining a name catalog becomes more difficult. However, more critically, this approach also binds data to locations and should be generally avoided. With this naming approach, creating an in-network transparent failover mechanism becomes tricky. For such mechanisms to take place, the UCLA node either needs to announce a /CSU prefix, or the name discovery system must maintain a mapping of CSU's names to corresponding names at UCLA, a cumbersome prospect either way.

Finally, within a name, the tradeoff of placing components earlier vs. later should be carefully considered. Consider two applications at running at CSU. Placing the application name immediately after general name component speeds up name parsing. For example, a name like $/cmip5/csu/filtering\_application$ requires parsing only three name component before handing the data over to the appropriate applications. On the contrary, placing application names at the end requires parsing the full name before deciding which application should receive the request. Consequently, naming designs must weigh the trade-offs of placing name components earlier vs. later.

In some cases, names do not aggregate very well, especially in the cases of partial data replication. For example, in the genomics use-case, names in the format $/genome/genus/species$ do not aggregate very well since there might be hundreds of species under a genus. In case of full data replication, only the $/genome$ portion of the names may be advertised in the routing system. However, let's assume CSU hosts data for $/genome/genus/species\_(1-10)$ and UCLA hosts data for $/genome/genus/species\_(10-100)$. In this case, these two sites must individually announce all distinct names up to $/genome/genus/species$ since there can not be any name aggregation up to that component. For example, the request for $/genome/genus/species\_1$ needs to go to CSU. If UCLA announces $/genome/genus/$, the request for $/genome/genus/species\_1$ might end up at UCLA, though it does not have that piece of content. However, in case of data is replicated or published at the genus level (i.e., both CSU and UCLA have $/genome/genus/species\_(1-100)$, the number of routes would be only the combination of the genome and genus components. Therefore, this thesis observes that in some cases, in-network functionality (such as automatic network supported failover) can be at odds with the routing and forwarding table size.

However, getting the names right benefits applications. For example, a common data movement operation in scientific communities is to retrieve a subset of an object in order to use it for analysis. The ability to retrieve only a specific subset of an object is critical due to the volume of the data stored after the initial model simulation. The following is an illustrative example using a water vapor objects.

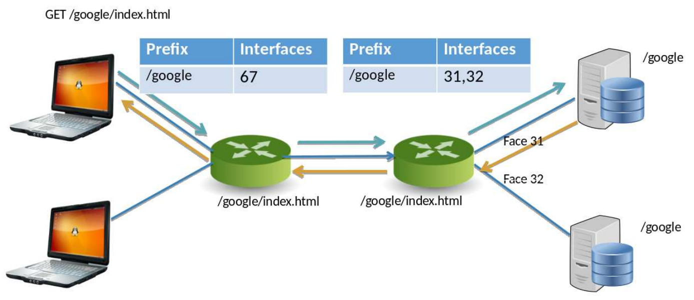


Figure 5.6: NDN retrieval example

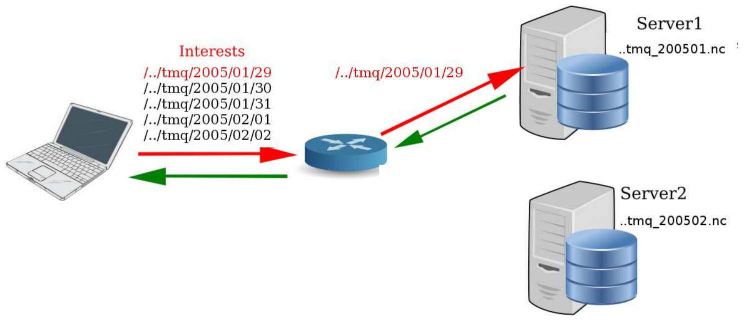


Figure 5.7: NDN partial retrieval

The dataset this example uses is organized by date (an object is one month's worth of data) and contains several water vapor related metrics. Assume that a user would like to retrieve all of the metrics from objects spanning 4 days of January and February of a given year. Further, assume that each monthly object resides on a different server. Using the current Internet architecture to retrieve the desired subset a user would (a) have to determine where the two files reside, (b) manually fetch files from each server, and (c) merge and trim the files into the final dataset. Note that in addition to requiring a priori knowledge of the location of the datasets (often very hard in itself) this process involves several manual steps.

Contrast the above steps with those required with NDN. A researcher interested in 4 days of data sends out one Interest for each day. Since data is distributed among multiple servers, the intermediate routers forwards Interests to the appropriate servers automatically; the user need not take any manual steps. Only the required subset of the data arrives, which is in turn presented to the user. Figure. 5.6 shows the process - a researcher asks for data from January 30th to February 2nd. Note that the data is distributed between two servers as indicated by the name prefixes they advertise. The intermediate router can now forward requests for January 30th and 31st to Server1 and February 1st and 2nd to Server2. Upon receiving the Interests, each server sends back the matching data. The retrieved data objects are cached at all intermediate nodes. In situations when the same data or any subset of a previously requested data is requested again, it is served from the intermediate caches without the need to go to the original data sources. As shown in Figure. 5.7 - beyond the obvious speed-up advantage in data retrieval, the data may still be available in case Server1 or Server2 goes down.

5.3.2 NDN Naming Recommendations

Based on these observations above, this section proposes a few guidelines for designing NDN compatible namespaces for science data:

1. Names should be built as a set of well-defined *components*.
2. The components should be organized in a hierarchical pattern.
3. More general (more common) name components should appear earlier in the hierarchy.
4. The routing portion of the names should not contain private information and should not be encrypted. The non-routing part can be encrypted without any loss in functionality

5. Care should be taken not to bind names to locations when data is expected to be replicated. The top-level prefix should describe the data, not an organization. For example, a name /CSU/CMIP5 binds the data to a location (CSU), but a name like /CMIP5/CSU does not.

6. Naming should carefully consider the trade-offs of placing name components earlier in the hierarchy vs. later. For example, names like /CSU/CMIP5/<> and /CSU/HEP/<> might direct requests to directs requests to climate and physics applications, respectively, without parsing the full name. However, placing those components at the end of the name will require parsing the full name and might be resource intensive.

7. There might be cases where names do not aggregate well, e.g., in the genomics use case above. Name developers should be mindful that routing/forwarding table sizes might be at odds with intelligent in-network functionality.

Naming is of paramount importance in NDN. This section shows how to name data to be compatible with an NDN network, how to translate existing data names into NDN, and presents general naming guidelines for NDN based applications. The next section will demonstrate how to find names once they are published. This thesis shows that naming data and finding names are the most critical components of the whole name-based ecosystem. Once data is named, and infrastructure for finding names is in place, all other functionality in the network becomes easy to implement. The effect is profound - scientific applications are not only simplified, but the intelligent functions in the network are shareable among various communities, reducing cost and efforts of implementing similar but domain-specific solutions.

Chapter 6

Name Discovery in NDN

The previous chapter shows how to name scientific data to be compatible with a named based network. However, only naming the data is not sufficient - mechanisms must exist for discovering names as well. Once names are discovered, scientific applications and users can retrieve data and benefit from in-network intelligence that NDN can provide. The reader to should note the subtle difference between name discovery and search in the context of this thesis. “Search” is a specific function that returns results based on user queries. The web search engines are good examples of this. Though overly simplified, this is how they work [42] - they download pages that are published on the Internet, index these pages based on the words and phrases they contain, and rank these pages according to some matrix (e.g., current popularity, history of user interactions, and other proprietary algorithms) . When a user searches for content using some keywords, the search engines matches these keywords with the database of pages and keywords and returns the ones that best match the user’s query.

Name discovery in the context of NDN is a subset of this generic search functionality. The process for name discovery works similarly, but the scope of the search is contained only to the content names. The name discovery process in this work is accomplished using a “catalog”, a database that holds and indexes the data names. The users enter the keywords on a web-based front-end that is similar to web search engine front-end. The keywords are then converted into a database query, a lookup is performed, and the content names are returned to the front-end. An application can also interface with the catalog using an API that is exposed. Once the names are known to the applications (or the users), they are free to utilize these names in any way they see fit. For example, the applications might use these names to transfer the actual data. Unlike a traditional network, no other additional information is necessary for these subsequent actions (e.g., retrieval) - only sending out an "Interest" into the network is sufficient.

Chapter 5 discusses a fundamental building block of an NDN-based ecosystem for big science, naming. However, only naming data is not sufficient - an infrastructure must exist that allows participants of this ecosystem to look up names easily, accurately, and within a reasonable timeframe. This chapter discusses this other fundamental building block of a name-based, NDN-supported ecosystem, name discovery. This chapter discusses the limitations of real-time name discovery protocols for large scientific data (time and difficulties in enumerating a complete namespace) and why a catalog-based system works better.

The idea of a catalog is not new. In fact, various scientific communities use catalogs extensively [145] [79] [45] [152]. However, the current way of cataloging is often loosely coupled [79]. For example, a scientific community often use multiple catalogs [166] for storing names, metadata, auxiliary data, derivative data, replication data, and others. These catalogs often use multiple technologies depending on data type, access pattern, and resource requirements [145] [79] [45] [152]. As a result there is a certain disconnect between these catalogs making them harder to use.

In an NDN-based system, only one catalog is needed, the catalog that holds the names of the data. Once the name is known, it can be used across all catalogs holding different pieces of information. Though separate catalog for different functionality will still exist, the names can work as a universal key across all these catalogs. The name discovery catalog, therefore, is a fundamental construct in an NDN-based ecosystem. This chapter discusses how this work designed such a distributed name-based catalog, what are the components that were necessary for building a catalog over NDN, and what are the security and performance considerations. Finally, this chapter also discusses how this catalog performs in a real-world deployment, both qualitatively and quantitatively.

This chapter describes the protocols that bind the actors in a scientific workflow and the software components together, specifically, the data/name publication protocol, the catalog updation protocol, the catalog synchronization protocol, and the data query protocol. All these protocols are also name based and supported by NDN. Protocols for data publication, query, synchronization, update, delete, and authentication all of which use names. This is the first scalable name discovery catalog that has been built on top of NDN. This chapter discusses the experiences and the lessons learned from this exercise.

6.1 NDN-based Distributed Name Catalog

A catalog holds a list of NDN names to provide ease of discovery of names. Once the names are known, all other subsequent operations, such as metadata lookup, data retrieval, or remote data placement can be performed using the names. Figure. 6.1 provides a high level overview of the catalog system. The basic building blocks are a publication tool for the data producer, a query tool for the user, an NDN database built around a traditional SQL database, and a number of protocols for publishing, catalog synchronization, and query. The data publisher publishes the names of data to a local catalog through the update protocol. Multiple distributed catalogs are synchronized among themselves to provide a consistent discovery experience across catalogs. The catalog also supports user query thorough a web-UI or a command line interface - allowing users to discover NDN-based names through the use of common search techniques such as auto-complete, specifying keywords, or browsing a name tree.

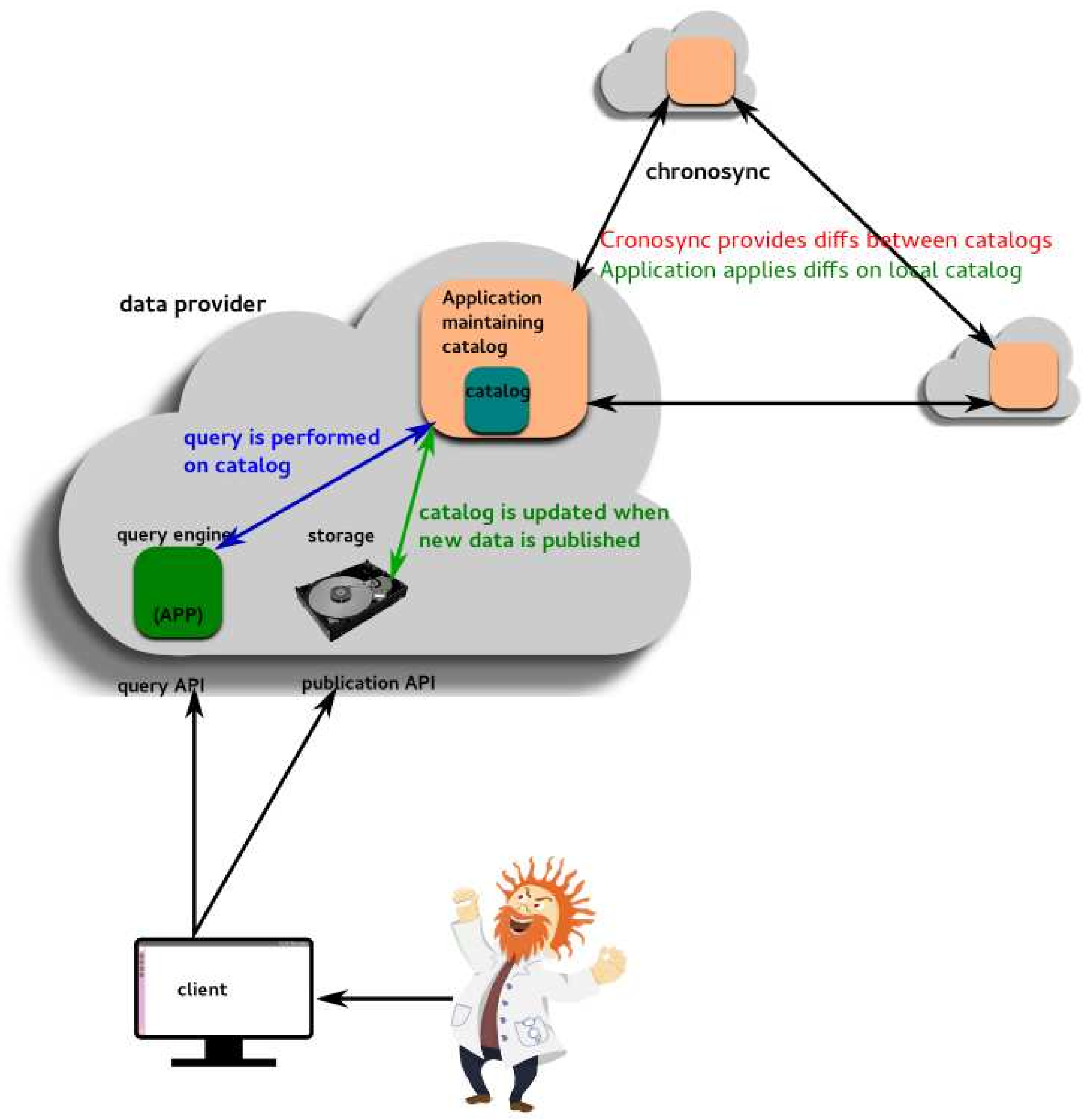

chronosync
Cronosync provides diffs between catalogs
Application applies diffs on local catalog
data provider
Application maintaining catalog
catalog
query is performed on catalog
catalog is updated when new data is published
query engine
storage
(APP)
query API
publication API
client

Figure 6.1: Catalog Backend

The high-level functionality from the scientists' perspective, as Figure. 6.5 shows, are (a) Publication: wherein a scientist produces new datasets and adds (or updates/removes) their names to the catalog system, making them discoverable to the world and (b) Query: wherein the scientist search for existing names. Also, the system needs to make sure the scientist has permission to do what they are requesting and after the scientist make local changes to the system, the rest of the system is also updated.

6.2 Actors in the System

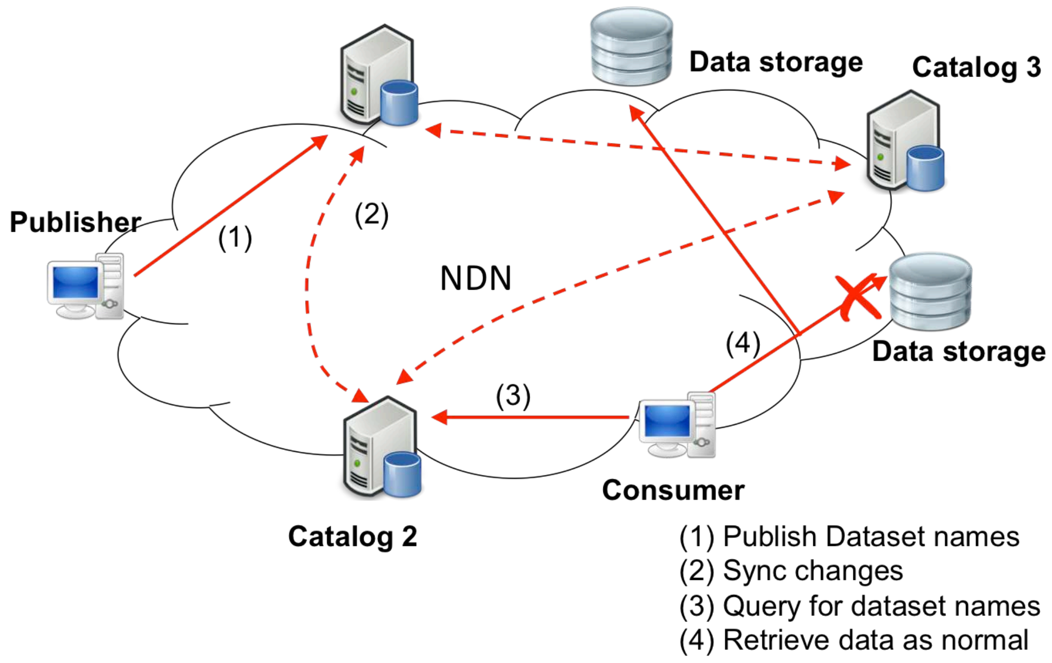


Figure 6.2: The Catalog Architecture

This section outlines the actors in the system and how they interact with the catalog. Three main actors are involved - the catalog provider, the data publisher, and the user. Figure. 6.2 shows the interactions between these actors.

The Catalog Provider

The primary goal for catalog providers is to answer user queries. They take the user queries and return NDN names in response. The catalog can coexist with a data provider. If the catalog provider is not co-located with a data provider, they can not make changes to the catalog themselves. They can only sync the changes other data providers have done since the changes to the catalog need to be with names signed by the original data publisher and communicated to the local publisher. Each catalog provider publishes its local catalog uses a shared namespace, e.g., $/cmip5/catalog$ in this example.

The Data Publisher

The data publisher publishes data into NDN. The data provider is also responsible for providing data in response to users' requests. The data provider translates the atmospheric simulation data into appropriate NDN format, stores, and serves it when a request for data comes in. Furthermore, the data provider is responsible for delegating cryptographic keys necessary to sign the NDN data to the data publishers (e.g., a scientist) . The publisher signs the data with the key received from the data provider. Once the data is signed and published into a repository, the data publisher triggers a callback which updates the catalog. The publisher can only remove the data they published. They can sign such a change and notify the data provider. The data provider applies the changes after verifying its authenticity.

The User

The user uses the catalog for discovering names. Once the user knows the name of the dataset he/she wants to retrieve, the rest of the retrieval process is straightforward. The user may also perform other operations such as remote subsetting, remote computations that the next chapter describes. The fundamental operation the user performs is content name discovery.

6.3 System Components

The goal of the catalog system is to provide reliable and seamless dataset discovery and retrieval using NDN-layer protocols. The primary functional components for the catalog are the following -

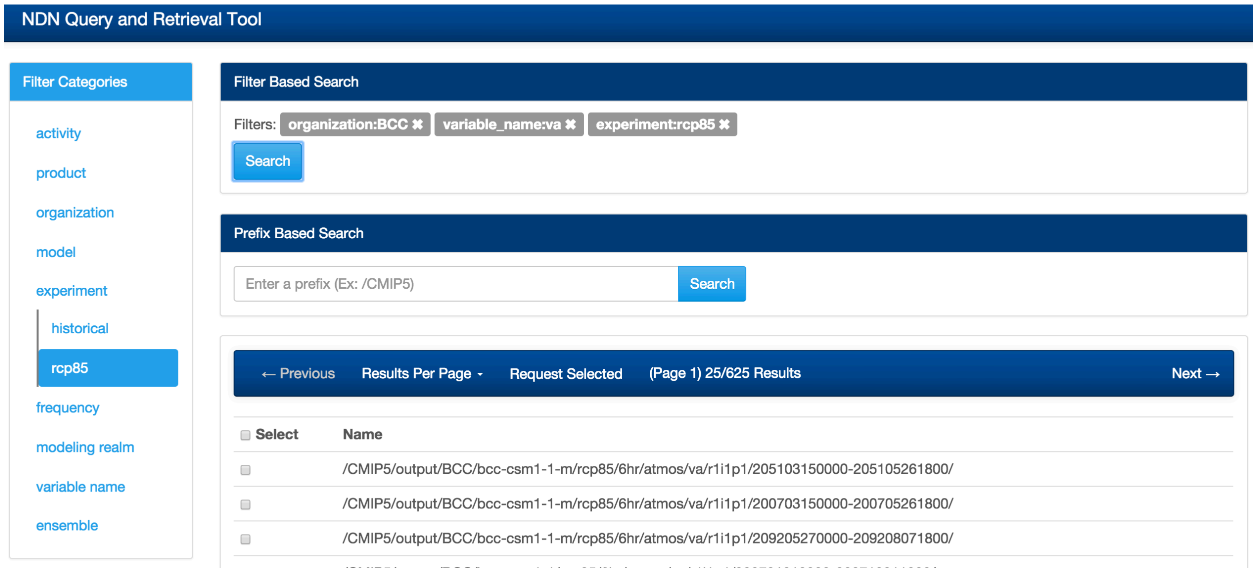
The Name Database: This work uses a standard SQL database (MySQL) for this particular implementation. Figure. 6.3 shows the table for climate data, tables for other domains are also similar. The fields are typically the name components we described in Chapter 5. Additionally, the table includes two other fields - one is ID, a serial number which acts as the primary key of this table. The second one is the hash value of the full name. This work often found duplicate files and datasets during translating community names - a hash value of the full name helps to deduplicate these names.

```
+----------------+----------------+------+-----+---------+----------------+
| Field          | Type           | Null | Key | Default | Extra          |
+----------------+----------------+------+-----+---------+----------------+
| id             | int(100)       | NO   | PRI | NULL    | auto_increment |
| sha256         | varchar(64)    | NO   | UNI | NULL    |                |
| name           | varchar(1000)  | NO   |     | NULL    |                |
| activity       | varchar(100)   | NO   |     | NULL    |                |
| product        | varchar(100)   | NO   |     | NULL    |                |
| organization   | varchar(100)   | NO   |     | NULL    |                |
| model          | varchar(100)   | NO   |     | NULL    |                |
| experiment     | varchar(100)   | NO   |     | NULL    |                |
| frequency      | varchar(100)   | NO   |     | NULL    |                |
| modeling_realm | varchar(100)   | NO   |     | NULL    |                |
| variable_name  | varchar(100)   | NO   |     | NULL    |                |
| ensemble       | varchar(100)   | NO   |     | NULL    |                |
| time           | varchar(100)   | NO   |     | NULL    |                |
+----------------+----------------+------+-----+---------+----------------+
```

Figure 6.3: Database Schema for the *Climate* Catalog

This work decided to build adapters into an existing database system rather than modifying or building a database since building a new database would not necessarily improve the state of this research. At the same time, the reader should note that though this work uses MySQL databases, there is no reason why other high-performance databases cannot be used.

The User Interfaces (UI) : There are two UIs for this system: a graphical user interface (GUI) for discovery and a command line user interface (CLUI) for publishing. The discovery tool is similar to the climate community's ESGF interface [14] as the atmospheric scientists interviewed stated a preference for and comfort with it. Similarly, the CLUI matches their existing publishing tools. However, this interface has proven to be sufficient for both physics and genomics communities.



Select	Name
☐	/CMIP5/output/BCC/bcc-csm1-1-m/rcp85/6hr/atmos/va/r1i1p1/205103150000-205105261800/
☐	/CMIP5/output/BCC/bcc-csm1-1-m/rcp85/6hr/atmos/va/r1i1p1/200703150000-200705261800/
☐	/CMIP5/output/BCC/bcc-csm1-1-m/rcp85/6hr/atmos/va/r1i1p1/209205270000-209208071800/

Figure 6.4: Data Discovery UI

The Web UI loosely based on the ESFG [14] to help consumers discover and download datasets. The Web UI is shown in Figure. 6.4. Users discover datasets in three ways: (a) by specifying a set of filters, (b) typing the name prefix directly or (c) browsing through a name tree. The filters are automatically populated based on the name components in the database. To help users type a name prefix directly, the UI provides name auto-completion as users type in the search box.

The CLUI does the reverse. It allows the data producers to add, modify, or remove names to/from the catalog. A GUI for data publication is not only cumbersome but mostly unnecessary according to the scientific collaborators.

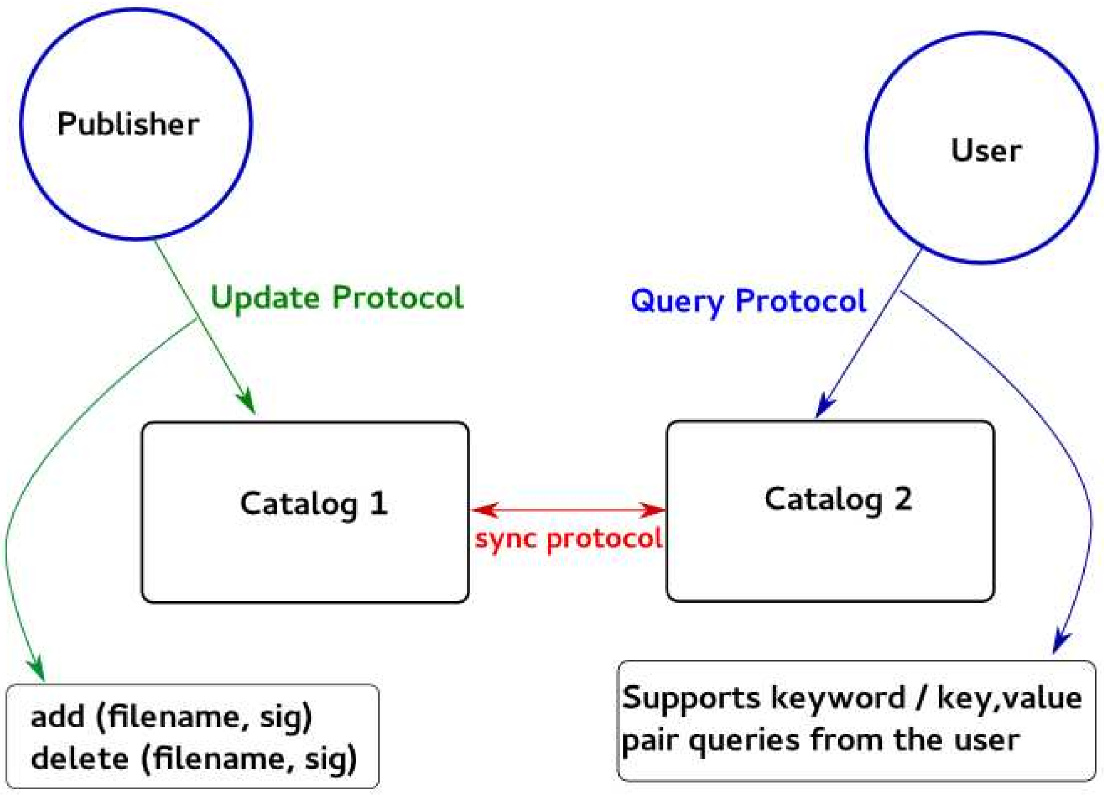


Figure 6.5: Catalog's Interaction Diagram

The Data Storage The data storage is separate from the name discovery process and is not an integral part of the catalog. In this system, the data storage part is implemented and described separately in the next chapter.

6.4 Protocols for a Scalable NDN Name Discovery System

The catalog design is independent of data types or specific namespaces, only requiring the published names to be hierarchical. This thesis assumes these NDN names, while not required, are under the jurisdiction of some specific entity, such as CMIP5. Inspired by the similarly named effort, this work will use the prefix /CMIP5 for illustration and brevity but does not claim that the NDN names used in this work should be the final names.

The catalog system is a loosely-coupled (only by name) distribute federation of multiple independent catalogs that use appropriate forwarding strategies in NDN to ensure that publishers and clients reach the best (e.g., the closest or the least congested) instance when making requests. The clients do not have to use the same instance, and the catalog allows them to move between instances even in the middle of a query if the network condition changes. Also, the system allows publishers to publish data into any catalog instance using their NDN publishing key - no other form of authentication is required. The trust system is also based on names, as Figure. 6.8 will show. The publishers may communicate with any of the catalog instances, typically the closest one as selected by NDN. Upon publication request, the catalog authenticates the request and updates the name listing. The catalogs also run a synchronization protocol among them that ensures all instances maintain the updated name listing. Finally, the catalog accepts search requests from clients and returns appropriate matches.

Figure. 6.2 shows the high-level view of the protocols. The protocols work together to achieve the following functionality -

- Publication: The data publishers should be able to publish names and other ancillary information to the catalog. Ancillary information may contain metadata or application-specific content that might be needed for various workflows. Note that the actual data must be published separately under the specific name. The act of publishing a name makes the data discoverable.

- Update and Delete: A data producer or an authorized agent of the data producer should be able to update published data. However, a data producer should not be able to change data published by other producers. Similarly, when a data publisher wishes to remove data from circulation, it can remove corresponding names from the catalog. Similar to the update operation, a data producer is only allowed to delete names published by it.

- Query: The catalog should be able to support user queries. Queries can be keyword based, key pair value based or both.
- Synchronization: Different data producers should be able to change their portions of the catalog. The producers will apply these changes to their local catalog and without coordination. Changes will propagate to other data providers through the sync protocol (described below) . Once synchronized, all catalogs will have uniform information. If there is a delay or failure in synchronization, the catalogs will continue to return results. This work prioritizes availability over consistency.

The following sections expand each of these protocols.

6.4.1 The data publication protocol

In this design, the catalogs are independent but trusted entities that all exist under a specific namespace, e.g. $/cmip5/catalog\_name$ or $/hep/catalog\_name$. Each catalog syncs to the other catalogs to fully populate its index of all the NDN names of NDN objects published under a certain namespace, e.g., $/cmip5$. The catalog can be stored as a text file, a database object or in any other binary form

This work assumes a distributed publishing model where publishers are various institutions that own datasets and wish to publish them under the same /CMIP5 prefix. The owner of /CMIP5 prefix delegates publisher keys to each institution. This publisher key enables each institution to publish names under a prefix such as </CMIP5/.../institution>/. In this work, all catalog instances serving CMIP5 names operate under the same namespace, e.g., /CMIP5/catalog . The catalog holds only names of datasets, not the actual data. The actual data is stored in the repositories

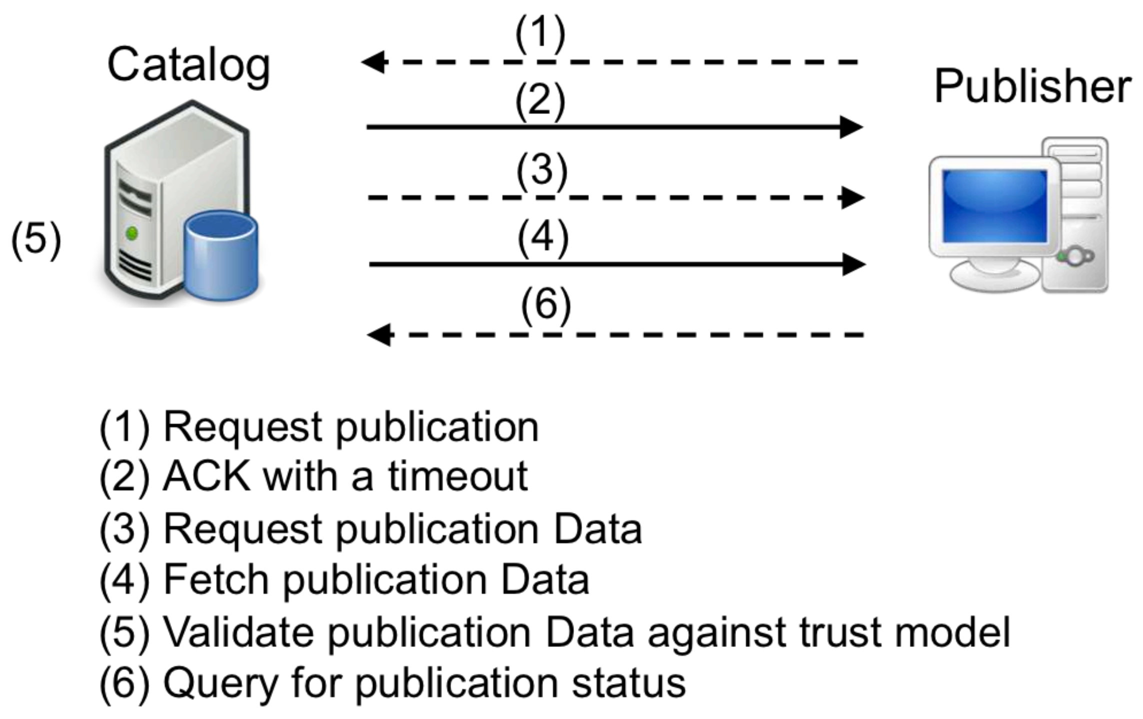
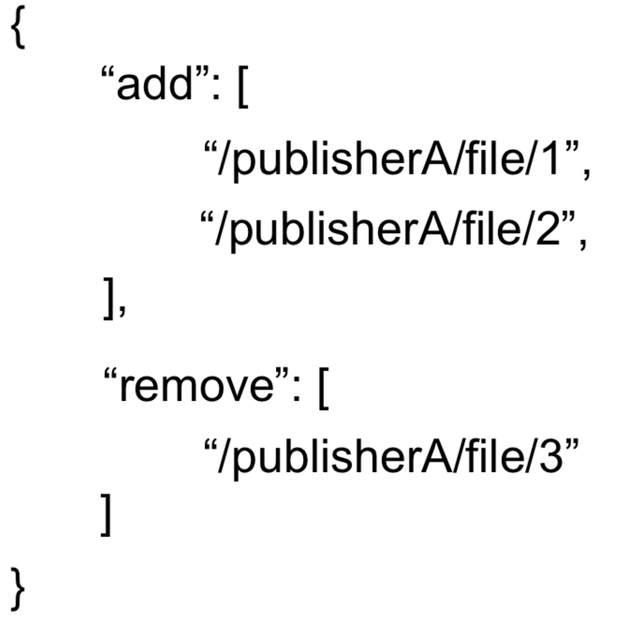

Catalog
Publisher
(1)
(2)
(3)
(4)
(5)
(6)
(1) Request publication
(2) ACK with a timeout
(3) Request publication Data
(4) Fetch publication Data
(5) Validate publication Data against trust model
(6) Query for publication status

Figure 6.6: Data Publication

operated by the publisher. Figure. 6.6 describes the publication protocol. When a catalog is launched, it registers a prefix with the local NFD. The registration prefix is $/CMIP5/catalog/publish$. For publication. A publisher encapsulates and signs a list of NDN names along with associated actions such as “add” or “remove” in one or more NDN data packets. The publisher then sends an Interest to the nearest catalog communicating its intention to update the catalog database. This is the typical way in NDN uses to upload information to a server - the publisher asks the server to pull newly available data. Upon receiving this publication Interest, the catalog replies with an acknowledgment. There are several reasons for sending an acknowledgment rather than immediately asking for the data; the update may be big, and if the catalog is currently busy it may not want to pull the data immediately. Without an acknowledgment, this would result in the publisher timing out and trying again. An acknowledgment can carry back an estimate for when the catalog will retrieve the data, allowing the publisher to retry on an informed timeout. An acknowledgment also erases state in the network, thus saving router resources. As Figure. 6.6 shows, the catalog pulls a set of instructions by the publisher. These instructions may include anything that the protocol supports, which, in this example, a list of new dataset names to be added or removed from the catalog.



```
{
    “add”: [
        “/publisherA/file/1”,
        “/publisherA/file/2”,
    ],
    “remove”: [
        “/publisherA/file/3”
    ]
}
```

Figure 6.7: Data Publication Payload

As soon as the catalog responds the publication Interest, it constructs Interests to fetch the published name list from the name prefix /<publisher-prefix>. To enable both add and remove operations, each published name list contains the JSON format (Figure. 6.7 shows the format) as the payload. The keys “add” or “remove” indicates associated operations with each name. Each key is followed by a list of dataset names so that the catalog could process dataset names in a batch. The publication data must also conform to the trust model to be successfully inserted into the catalog.

The publication API provides the ability to publish to the catalog. For example, let’s say a publisher, CSU wants to publish two files, $/cmip5/file1$ and $/cmip5/file2$. The publisher calls the $publish$ method which does the following:

- Calls a method $publish\_file$ with the file names. This publishes the files into NDN.
- Upon completion of the publishing, the $publish\_file$ method triggers a callback $update\_catalog$ that appends the following to the local version of the catalog.
- Once updation is done, the catalog is synced between other data providers (described later) . Upon reaching a steady state, all providers will have the same catalog.

Publication Access Control

As mentioned earlier, catalogs make use of the digital signatures provided by NDN to enforce access control for publishers. Keys for these digital signatures are distributed separately, and NDN does not dictate how they are distributed; they may be distributed through a PKI, web-of-trust mechanisms, or anything else agreed upon by the applications. This work uses PKI as the mechanism to distribute the keys. When an Interest initiates a pull for instructions from a publisher, the catalog will fetch the instructions but not accept them if they are not correctly signed. This forms the first line of defense against illegitimate publishers from making changes to the catalog. Although we have not implemented this, the catalog may also maintain a list of approved publishers, either built-into the distribution process or distributed out of the band. The next line of defense is designed to prevent publishers from making unauthorized changes to catalog entries. This work assumes that the policy in place is to allow a publisher to make changes only to entries published under the publisher's prefix; a publisher in CSU is only allowed to make changes to datasets under the prefix /CMIP5/CSU/. This process is depicted in Figure. 6.8.

The trust model takes both the valid publisher and the malicious publisher into account. Since signatures are built in NDN Data packets, NDN naturally can reject malicious publishers by checking the signature in each Data packet. Note that the

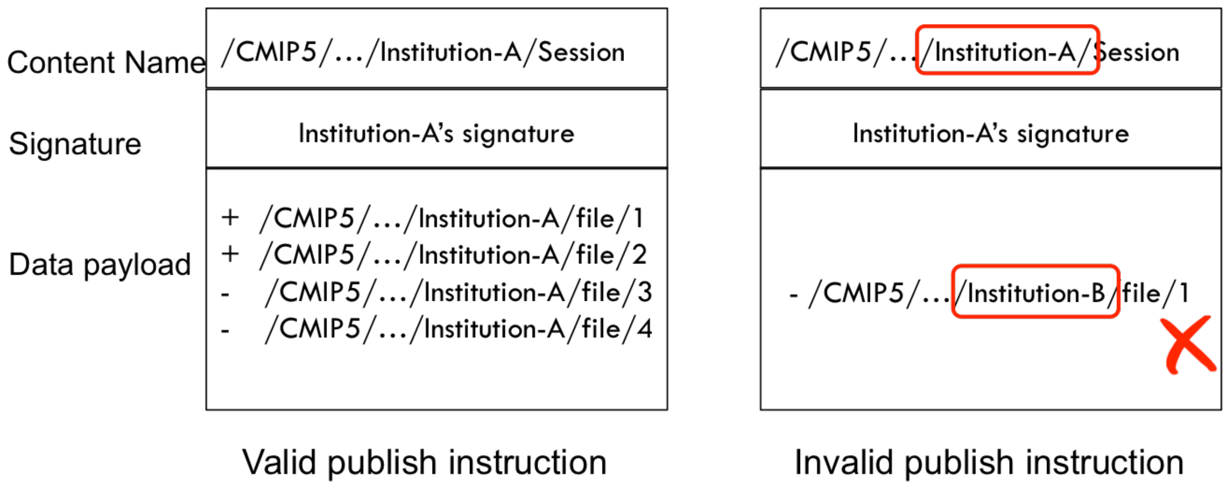

Content Name
Signature
Data payload
/CMIP5/.../Institution-A/Session
Institution-A's signature
+ /CMIP5/.../Institution-A/file/1
+ /CMIP5/.../Institution-A/file/2
- /CMIP5/.../Institution-A/file/3
- /CMIP5/.../Institution-A/file/4
Valid publish instruction
/CMIP5/.../Institution-A/Session
Institution-A's signature
- /CMIP5/.../Institution-B/file/1
Invalid publish instruction

Figure 6.8: Valid vs. Invalid Catalog Instructions

signature checking procedure does not bind to any specific key management model. The hierarchical key structure, for example, assumes that there is a root key that all users trust (a trust anchor) . If the catalog adopts the hierarchical key structure, the validation procedure will check the signature until it reaches a trust anchor.

However, signature validation is unable to prevent valid publishers from adding/removing names of datasets that do not belong under their namespace. To assure valid publishers perform changes correctly, the trust model further checks if all dataset names that contained in the payload are under the publishers' namespace (Figure. 6.8) .

6.4.2 Synchronizing the name catalogs

A single centralized catalog instance can easily become a bottleneck when subjected to a large number of queries or publication requests. Moreover, it becomes a single point of failure and introduces increased latency for distant clients. The research community has worked on distributed synchronization from the very early days of the Internet [109] [38]. A plethora of centralized solutions such as Dropbox, Office365, Google Docs, as well as peer-to-peer synchronization solutions such as BitTorrent has also been proposed and deployed.

The requirements for this particular catalog were as follows:

- The scientists should be able to publish names to any of the catalog instances. The ability to publish to any catalog instance allows the network to choose the most appropriate catalog instance (e.g., the nearest) for the publisher.
- The catalogs should be automatically synchronized to each other.
- In case of a network failure or partition, the catalogs should be able to serve the most recent data. If a network partition happens, the available names might differ between two catalog instances.
- The catalogs should be able to recover from failure and reconcile their state automatically.
- In normal network conditions, the catalogs should return consistent results. The synchronization delay might affect some queries; therefore this delay should be kept to a minimum.

NDN, with named data and native multicast support, presents a new opportunity for distributed synchronization [203]. To make the name catalog scalable, this work adopts a federated architecture of several catalogs running an NDN-based synchronization protocol, Chronosync [203]. At the time of the implementation, this was the only NDN-based sync protocol available, therefore was the only choice. The goal of this section is not to *engineer* a distributed sync protocol over NDN but to *design* a distributed catalog that utilizes *an* NDN-based sync protocol. Other more recent NDN-based sync protocols such as PartialSync [200], VectorSync [158], or FileSync [114] will also work for the requirements enumerated above.

In this work, all catalog instances in the federation announce a single prefix; this method allows the network to route queries to the most appropriate instance as determined by the NDN strategy. Depending on the strategy queries may go to the nearest catalog instance or the least congested one. This work uses a synchronization protocol called ChronoSync [203] to ensure consistency. Chronosync is a digest tree-based state exchange protocol. As shown in Figure. 6.9, the state is calculated based on the published content and its sequence number. When new content is published, a new state is calculated, and sync messages are propagated to other nodes. A multicast namespace $/ndn/broadcast/sync$ is reserved to exchange sync messages. In our application, the sync messages are simply the instruction messages described in the previous section. Whenever a catalog instance receives such a message, it validates the payload and applies the updates to its local database. It then propagates the sync message to other catalog instances. A periodic database snapshot is generated to serve as a recovery state in case of failure or a database restart. The following section describes the mechanisms in details.

Upon startup, data providers register local dataset with the local cataloging service instance. Using chronosync protocol, each data provider then retrieves differences between its own catalog and other copies of the catalog. An application authenticates and applies all differences on the local copy. Once the system stabilizes, every data provider has all the object names under the given name prefix, $/catalog/cmip5$. We call this a “master catalog”. This catalog will then be used for data discovery, search and other functionalities. Similarly, when something changes in the catalog, chronosync carries the changes to every data publishing node and catalogs are updated.

Chronosync for exchanging diffs

In chronosync, every node keeps a standing Interest for the master catalog name $/catalog/cmip5$ appended by the digest of the local version of the catalog, i.e., $/catalog/cmip5/current\_digest$. When someone publishes more data, the catalog, and the digest changes. The new catalog is published under $/catalog/cmip5/current\_digest/new\_digest$. Data with longer names fulfill Interests with a shorter prefix. Therefore, all the nodes with standing Interests will receive recent changes. The data providers can take these changes and apply to the local catalog. The new catalog's hash should be equal to the new_digest. Once the data is received and catalogs updated, the new_digest becomes current_digest, and the same process continues.

Chronosync uses two prefixes, a broadcast prefix, and a data prefix to exchange updates and data, respectively. Broadcast prefix for the catalog will be, e.g, $/ndn/broadcast$
$/cmip5/catalog$. Data producers can publish data under $/cmip5$. There is a monotonically increasing sequence number starting from 0 and is updated when new data is produced. This sequence number makes it easier to infer all data produced by a certain publisher.

Each node keeps a digest tree which keeps track of everyone else's Name prefix and max sequence number. Together all these digests create a state digest. The nodes keep a standing Interest for the state digest.

For example, let us say the state digest for the catalog is $< ABC >$. The nodes keep a standing Interest for $/ndn/broadcast/cmip5/catalog/ < ABC >$. When Bob makes changes to the catalog, Bob's digest change. Since the state digest is dependent on Bob's digest, it changes too. Let us say this new digest is $< DEF >$. The node publishes the sync data under $/ndn/broadcast/cmip5/catalog/ < ABC > / < DEF >$. Since this is prefix is longer than the standing Interest, it satisfied the standing Interest.

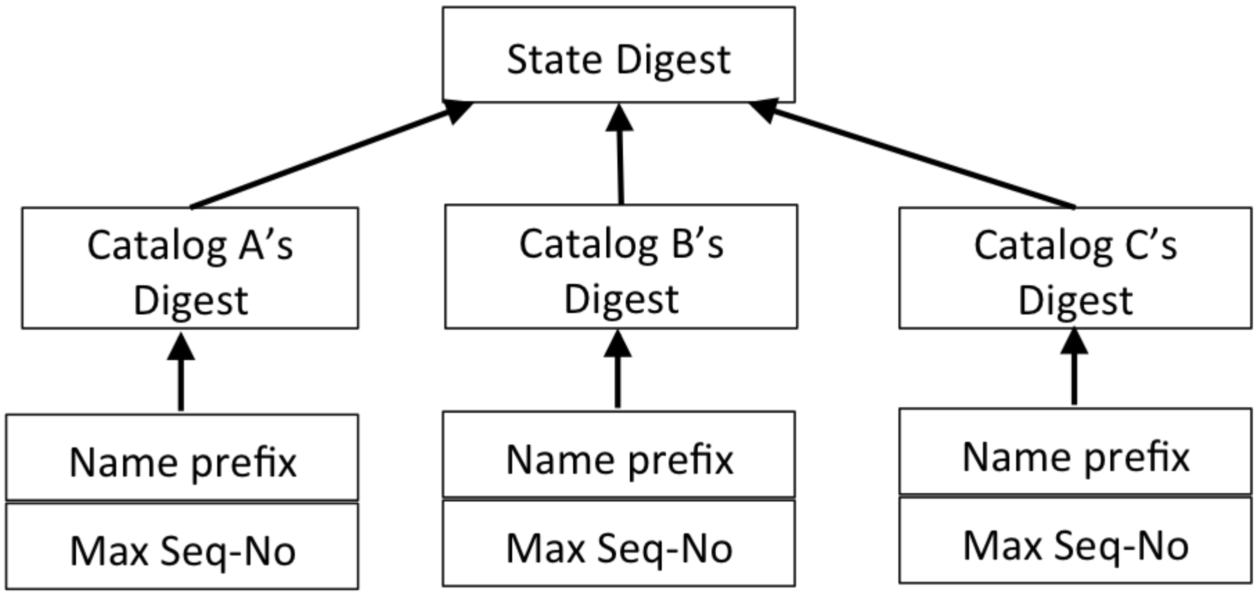


Figure 6.9: An example of digest tree used in catalog federation [203]

Once the sync data is received, each node applies the data to its internal digest tree and updates it. For example, let us say 1010 is the new max sequence number after sync. If a node had data objects up to sequence number 1000, it simply expresses 10 Interests for new data objects. Note that this list will contain names under the data prefix, not the broadcast prefix. So the expressed Interests would look like $/ndn/cmip5/1001$.

Once the sync is applied, the nodes express a standing Interest for the current state digest $/ndn/broadcast/cmip5/catalog/ < DEF >$.

Bootstrapping for sync

For joining a sync group, a new node only needs to know the broadcast address for the catalog, e.g., $/ndn/broadcast/cmip5/catalog/$. This needs to be communicated out-of-band.

Once a node boots up, it knows nothing about the catalog. It sends out an Interest for $/ndn/broadcast/cmip5/catalog/$ and gets back a reply with the current state digest, e.g., $/ndn/broadcast/cmip5/catalog/ < DEF >$. This node does not have a digest tree, and it can send out a special Interest for

$/ndn/broadcast/cmip5/catalog/bootstrap/ < DEF >$. Once a node which has digest $< DEF >$ receives the Interest, it sends back the maximum sequence numbers along with all the publisher data prefixes. The new node then can pull all the data and update its digest tree.

Chronosync recovery

When a node receives an unknown digest (e.g., after network partition), it sends out a recovery Interest. The recovery interest has the Interest name followed by "/recovery and the unknown digest. Upon receiving this Interest, nodes who produced the particular digest sends back the entire state of their producers. The recipient compares these and applies the differences to the local tree.

Authentication of changes

Chronosync does not provide a mechanism to authenticate changes. All changes must be authenticated at the application layer. The data publisher signs each line (each line represents a name) of the catalog. The application verifies if any changes to published data is signed by the producer or some entity higher in the trust keychain. If this is true, the application then applies the changes to the local catalog.

6.4.3 The catalog updation protocol

```
update_catalog
{
 "add" : [
    "/publisherA/file/1",
    "/publisherA/file/2",
    "/publisherA/file/3",
  ],
  "remove" : [
    "/publisherA/file/4",
    "/publisherA/file/5",
  ]
}
```

For publication, data need to be signed. Scientists can obtain a key from the data provider. In case the scientist has multiple affiliations, he/she can choose one to publish the data. For example, a scientist with affiliation with LBL and CSU will choose one key to sign the data.

Deleting Names from the catalog

Since many data providers update the catalog, only the original publisher should be allowed to update the particular listing. For deleting $/cmip5/file1$ for this example, CSU will delete the file, which will trigger a callback for updating the catalog with a delete action. When this change propagates to other data publishers, the data publishers verify and apply the change if the same signature signs the original entry and the new change.

```
update_catalog(action, file_name)
{
    delete {filename, publisher name, signature to catalog}
}

delete_file(file_name, update_catalog)
{
publish file1 under /cmip5/file1
update_catalog(delete, /cmip5/file1)
}
```

6.4.4 The Data Query Protocol

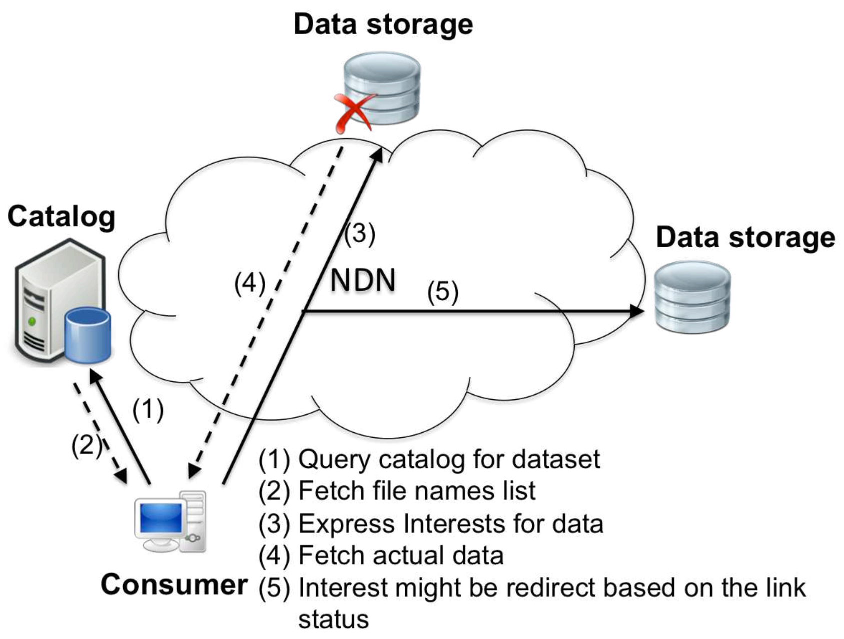

Data storage
Catalog
Data storage
NDN
(1)
(2)
(3)
(4)
(5)
Consumer
(1) Query catalog for dataset
(2) Fetch file names list
(3) Express Interests for data
(4) Fetch actual data
(5) Interest might be redirect based on the link status

Figure 6.10: Data Retrieval over Catalog

This section describes consumers' interaction with the catalog federation for discovering NDN names. The message exchanges between consumers and the catalog are shown in (Figure. 6.10) . The catalog design does not restrict consumers to a specific catalog instance. Since the query name captures all the necessary parameters, all catalogs will generate consistent results for a given query.

When the query Interest arrives, the catalog responds with an ACK. The ACK name contains the catalog ID, the query parameters, and the local database version (/CMIP5/catalog
/query/<catalog-id>/<query-params>/<version>) . The freshness time for the ACK

is set to 0, which means it will not be cached in NDN routers. This ensures consumers receive responses not from a cached response but a live catalog instance. After the ACK is sent, the catalog converts the query parameters into an appropriate SQL string and issues a SQL request to retrieve the results. The query results contain a list of names. They are packetized and published into the memory storage under /CMIP5/catalog/query-results namespace. As soon as the ACK arrives, the consumer constructs the corresponding query-result content names by replacing "query" with "query-results" (/CMIP5/catalog/query-results/<catalog-id>/<query-params>/<version>) . Currently, the NDN strategy directs queries consistently to the same catalog instance.

Note that query-results packets are cacheable and therefore same query Interests do not trigger multiple database queries which also protects the catalog from DoS attacks.

The catalog, either directly or through other means (e.g., by being stored in a database) need to support user queries. The users enter queries in the front end, which gets translated into keyword-based query or a key, value pair lookup. A layer between the catalog and the user translates the user query into an appropriate format that the catalog understands and can respond to.

6.5 Performance Evaluation of NDN-SCI

The previous section has described how name discovery can be accomplished using NDN and a distributed catalog. This section evaluates the performance of the climate instance of the catalog deployed over the ndn-sci testbed. This section shows that the delay for publication, search and parallel queries are all in the order of tens of milliseconds. While this work finds (and the anecdotal evidence from the scientists back these up) these numbers to be reasonable, note that the current implementation is a prototype and therefore, is not optimized for performance. The goal of this study is not to provide performance numbers but to simply show that NDN is capable of supporting features that can be beneficial to scientific communities. As NDN adaptation grows, the performance numbers should improve. The following sections evaluate the performance of NDN-SCI in terms of publication and search performance.

This work used the complete CMIP5 namespace. The namespace had 2.7 million names; these names were translated and published into a local catalog. Once the names were inserted, the catalog instances exchanged and synchronized the names. Once the names are synchronized, users can query any catalog instance.

6.5.1 Publication latency

The central components of this framework are NDN names. Rest of the functionality such as retrieval and search built on top of the published names. It is therefore vitally important to be able to publish names in the system quickly since the delay in publication will render content unreachable. In addition to the publication, the synchronization delay between the catalog instances are also important. If the names are not properly synchronized, the same query might bring different answers depending on which catalog instance it reaches.

To estimate how fast names can be published and synchronizes in NDN-SCI, this experiment took two instances of the catalog deployed on the testbed and measured the name publication and synchronization times. The RTT between these two machines hosting the catalog instances is approximately 3.5 ms.

This experiment then randomly picked some names from the list of 2.7 million CMIP5 names and published them in the catalog. For statistical accuracy, each experiment was repeated ten times. Each of the experiments measured how the number of published dataset names affect the combined synchronization and publication delays between catalog instances. This delay in the context of this experiment is defined as the time interval between the arrival of the first publish request at a catalog and final synchronization of all catalog instances. According to the results, the median synchronization delay is around 50ms, including the propagation delay, processing time, signature verification and authentication time of the publishing request. This delay increases as the number of dataset names increases but remains below 200 ms. Since the catalog does not synchronize the actual data but only the names, the operation is lightweight.

6.5.2 Name discovery latency

Query speed is essential for searching data, to test the performance of the catalog, this experiment selected 10K names of random lengths, with lengths varying from 1 to 9 components. This experiment then uses these names as actual queries to the catalog systems. The reader should remember from the previous section that the catalog supports path query (similar to an SQL command) and auto-complete based search. First, the experiment runs all queries synchronously to guarantee that no query affects others. Depending on the name length, these queries took between 20-120ms. Note that the delay is comparatively more substantial for the first queries; after that, responses are cached. Not having to perform repeated database lookups significantly reduces the delay for subsequent queries.

6.5.3 Query latency for parallel requests

This experiment measures catalog query delay, i.e., the time between user issues a query and the first name appears on the screen during high-load situations. This experiment randomly chose a number of parallel queries and sent them to the catalog. Figure. 6.12 shows that parallel delay increases as the number of parallel queries grow but remain under 70ms for 128 parallel queries. Note that given the ability of the NDN's intelligent forwarding plane to balance requests transparently to the requester,

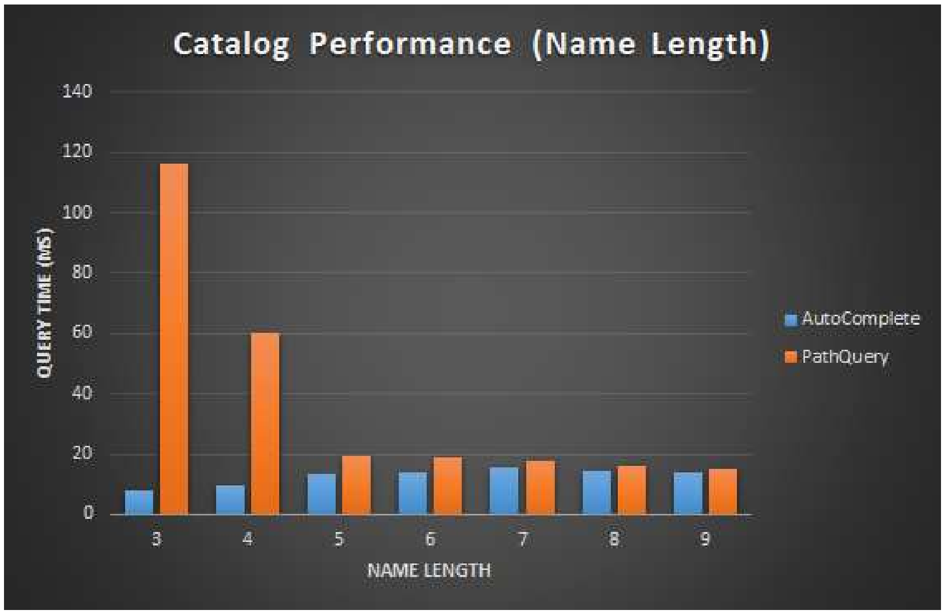


Figure 6.11: UI performance

it is indeed possible to spread the query load among multiple catalogs or dynamically spawn new instances of the catalog to reduce delays.

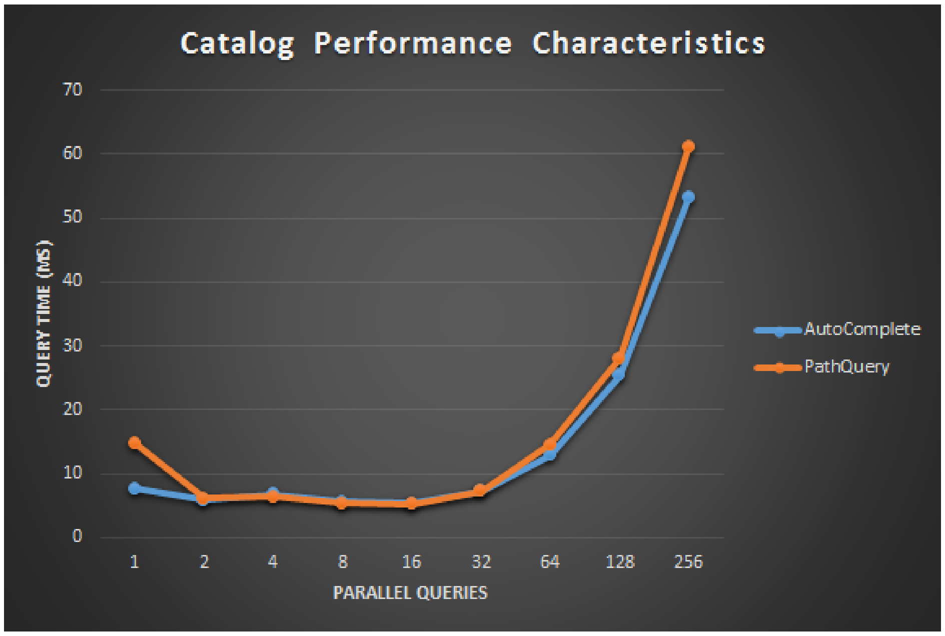


Figure 6.12: Parallel Query Time

6.5.4 Qualitative Evaluation

Table 6.1: Qualitative Evaluation

Question	**Positive Response (%)**	**Negative Response (%)**
Is the UI intuitive and easy to use?	100	0
Is the data naming scheme appropriate?	100	0
Were you able to search through the datasets?	100	0
Were you able to view any metadata?	100	0
Are all three search methods (filter, name and namespace) useful?	100	0
Were you able to stage data?	80	20
Were you able to perform subsetting?	80	20

In addition to performing quantitative measurements, the author also requested feedback from climate scientists at CSU. The sample size of this survey was deliberately small (less than 10) since the goal was to perform a preliminary evaluation before sending it out to a larger community. Table 6.1 summarizes the questions and the responses. Essentially, the scientists were asked questions about how easy and intuitive the software was and if the participants were able to perform the normal tasks. The responses were very encouraging, not only were they perform the normal functions such as search, retrieval, and staging, they were also happy with the ease of use and the intuitive UI design. The users found that performing subsetting, one of the specialized tasks, was a bit problematic - the software has since been modified to address this problem.

There are several takeaways from this work -

- Proper naming and a name discovery system is essential to any NDN based ecosystems. There are several ways to perform name discovery, such as network-supported queries and catalogs. This work chooses to use a distributed catalog for two reasons - catalogs can provide a complete enumeration of namespaces and catalogs can serve data with very few roundtrips.

- Even for name-based networks, some functions such as name discovery should be at the application layer. NDN better supports name discovery applications but should not itself perform name discovery for scientific data, which might take a long time and there is no way to assure all names have been discovered.

- Named discovery is one of the crucial parts of NDN-based ecosystem. Since names in NDN are globally unique, the names can act as the common interface between modules providing different functionality. For example, data retrieval, operations such as data staging and subsetting, data publication and deletion can be performed using the names. Having a common interface between various components is very desirable in scientific workflows [79].

- Name catalogs are lightweight and faster to synchronize. Furthermore, an NDN-based sync mechanism can provide novel distributed synchronization protocols using name-based packet forwarding and its native multicast support. Even when the network does not work as intended, the name catalogs can provide results, though the results might be outdated.

- NDN allows easy decoupling of operations. Once the names are known, the rest of the operations are decoupled from each other. The names can be used for any number of operations, and these operations can use the NDN network as they see fit. For example, a retrieval service might use the names for fastest retrieval over the NDN network, and a staging service may use the same names for the most reliable delivery of the data, and so on.

- NDN also provides a name based trust and security mechanism. This chapter discusses one such example where the names being published must belong to an organization that owns the namespace. More sophisticated trust schemes can also be used.
- Finally, the name-based operations are intuitive to the scientists, as the qualitative survey demonstrated. This thesis hopes that ease of use will lower the barrier of adoption for an NDN-supported, name-based workflow.

This section demonstrates how to build a service on top of an NDN network that enables name discovery. The software, simply referred to as the “catalog” that uses a database to hold the NDN names, synchronizes the names across distributed instances to provide a consistent view, and finally, allows users to query the database. Once the users (or the applications) know the names, they can perform other tasks, for example, retrieval. The next chapter discusses the network protocols. These protocols implement some of the common functionality in the network to facilitate common operations and are complementary to the naming, and name discovery system discussed so far.

Chapter 7

The NDN-SCI Data Management Framework

The previous chapters show how naming and name discovery can create a simplified and generic framework for scientific data distribution and management. However, simplification and generalization of applications come with the assumption that the network will provide some of the common built-in functionality, such as data retrieval, pushing computation to the data source, and others, at the network layer.

This chapter discusses these common in-network functionalities necessary to support a generic name based data management framework for large science data. The functionality that this section discusses is by no means exhaustive; this section aims to demonstrate NDN's capabilities at the network layer in the context of large scientific data transfer and management. In addition to in-network functionality, this section also demonstrates NDN's capability to accommodate new and novel functionality in the network such as strategic caching for reducing bandwidth consumption, and multi-source retrieval for better throughput that are hard to implement today. Some of these protocols described in this chapter are evaluated using NDNSim, and some are evaluated using the testbed. Though the testbed provides more realistic experiments, in some cases it did not have enough nodes to support some of the experiments, requiring the use of NDNSim. The protocols developed using NDNSim are directly portable and deployable on the testbed.

Some in-network functionalities necessary for the large science data management framework are included in the NDN architecture [196] [7]. Functionality such as choosing an alternate route when the existing route fails, forwarding an Interest to a data source using its name, are integral parts of the architecture [196] [7]. Other functions such as forwarding Interests to multiple data sources, finding the "best" route for retrieving a piece of content are not part of the architecture itself.

The NDN architecture provides a "strategy layer" for supporting functionalities that are not directly in the architecture. Section 2 discusses the strategy layer in details - but succinctly put, this layer allows network users and operators to plug-in in-network functionality that they see fit. This chapter extensively utilizes the strategy layer, as the following sections will describe.

The strategy layer makes it easier to implement generic, reusable functionality in the network. For example, various applications no longer need to implement the best-effort data retrieval protocols separately. On the other hand, some applications may require specialized protocols for retrieval, such as data transfer by a deadline or data transfers with the highest possible reliability. The strategy layer can also support these specific protocols. The strategy layer in NDN allows per-namespace strategy separation, e.g., fastest retrieval strategy for the /CMIP5 namespace and most reliable retrieval for the /HEP namespace, This feature that enables various strategies to co-exist in the network without interfering with requirements of different applications.

This chapter discusses protocols that utilize both NDN's architectural artifacts as well as strategies for providing in-network functionality required for big science data. During the process, the chapter also demonstrates some of the advantages and complexities associated with delegating functionality to in-network entities. Functionally, this chapter divides the requirements of big science data into three primary categories (a) High-performance data retrieval challenges (b) Time-constrained data retrieval challenges; and (c) Usability challenges. An NDN-based network can help with all three. NDN can address high-performance challenges through caching, multi-source retrieval, retrieval from the fastest data source, and even creating new paths on-demand. Time constraint challenges can be solved by finding the best sources that can serve data within the allocated time, creating new on-demand high-speed paths, or strategically caching content in the network. Caching and intelligent strategy can improve resource (e.g., bandwidth) utilization. Finally, NDN can address usability challenges by providing a standard interface at the network layer and also, by supporting novel mechanisms such as pushing computations to the data source, staging data to the remote computing nodes, and transferring only a subset of data required for computation instead of the full dataset.

The scope of this chapter is confined to large scientific data transfers over single-domain science networks such as Internet2 or ESnet. This chapter does not consider inter-domain reservations which can introduce a large number of operational challenges such as traffic engineering policies, peering, and economic incentives [138]. This work does not propose alternate QoS mechanisms for NDN networks equivalent to IntServ or DiffServ [90] in IP. Instead, this work provides the building blocks which such future NDN-based services will utilize.

By providing API-like abstractions for data retrieval and other operations, NDN can standardize implementations of common functionality and simplify applications. As the following sections show, NDN can implement high-performance data retrieval, time-constrained data retrieval, and similar other functionalities at the network layer that are agnostic of scientific domains or use cases. Climate, HEP, Genomics, and others, can use these functionality without domain-specific modifications, largely eliminating the duplicate efforts currently needed to implement domain-specific solutions.

7.1 High Performance Data Retrieval Challenges

High-performance data retrieval is one of the critical challenges in scientific communities [132] [65] [41]. In the research communities utilizing large volumes of data, data need to be transferred at the highest possible throughput because the network is still the bottleneck for such transfers [131] [132] [115]. The research communities are also deploying extreme-scale supercomputing facilities that need to have ultra-high-throughput data transfer capabilities for moving data in or out of them [115].

The current complexity of large scientific data retrieval comes from various artifacts of the current network design. For example, if a data source slows down, or if the intermediate routers lose packets, the TCP/IP congestion control algorithms might irrevocably slow down the transfer. While the application can connect to another faster server and resume download, there are two problems associated with delegating the application with this task - (a) it requires the application (or some intermediary) to monitor and track the network conditions, a problematic proposition in itself [89] (b) switching over to a new server is not straightforward. The application needs to perform periodic checkpointing, track available data replicas, find the fastest one, and move over the retrieval session to it. SDN based approaches may reduce the complexity of some of these problems, but configuring an SDN based control place is considerably complex [115] [132], and the network complexity increases with the application complexity and the size of maintained network [107] [35].

7.1.1 Caching

In-network caching of data packets is a novel feature in NDN. Named and signed data packets are cached in the network and can be served for subsequent requests. A considerable number of studies [88] [85] [193] [60] [190] [24] [199] [112] have shown the benefits (and the challenges) of caching in the network. For big-data transfers, caching provides three primary benefits, fast retransmission, shorter retrieval paths, faster retrieval in the presence of lossy/bottleneck links.

Currently, high-bandwidth delay networks do not fair well with packet loss and congestion [41]. Even losing one packet in *millions* might irrecoverably slow down large data transfers [4]. NDN's in-network caches can retransmit the packets reasonably quickly without having to go back to the data source. The ability to quickly retransmit packet may also help with avoiding congestion control algorithms to kick-in, albeit at the expense of a larger in-network cache. With NDN, the packet is likely to come back from a nearer cache without having to go back to the transmitter. Not having to go back to the data source is also less likely to trigger congestion control algorithm. However, the trade-off is larger in-network caches that are able to support packet retransmissions in case packet loss happens. The size of these caches will depend on the network and application characteristics such as traffic volume, retransmission time, and the bandwidth-delay of a network [160].

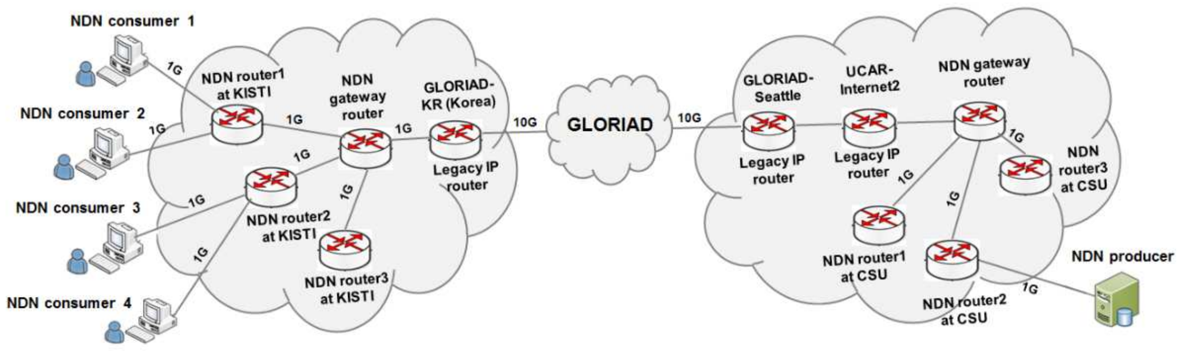


Figure 7.1: HTTP/FTP/ESGF vs NDN comparison topology

Caching also speeds up data retrieval. Chapter 3 presented a simulation study that showed the caching could help data distribution even when a 1-GB cache is present at the network's edge. Using the traffic pattern and the Zipf distribution parameter a

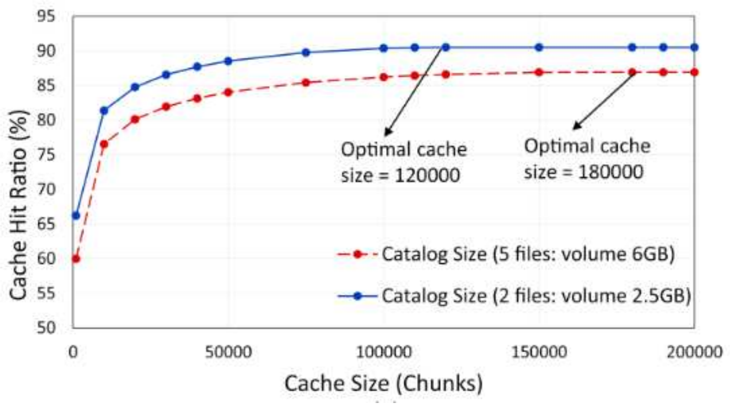


Figure 7.2: Cache size vs cache hit ratio

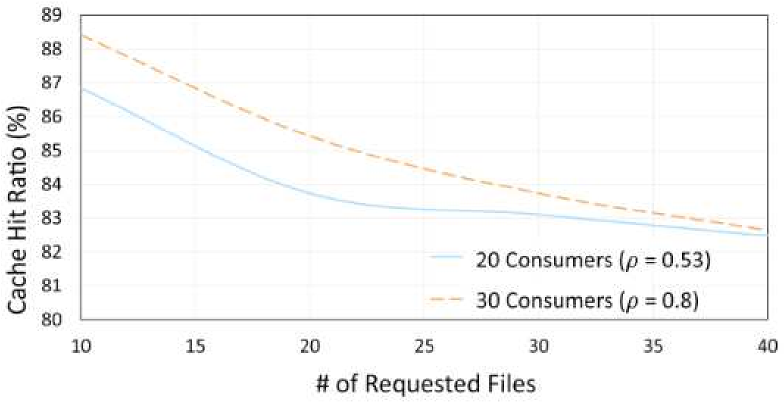


Figure 7.3: Cache hit vs number of files

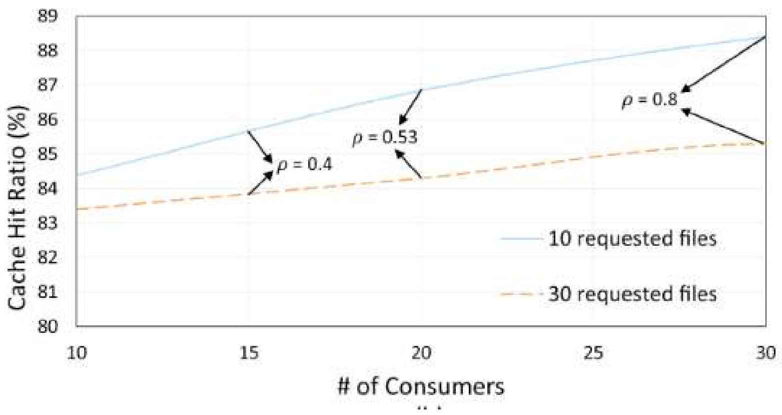


Figure 7.4: Cache hit vs number of consumers

follow-up study [113] over the testbed [72] was created. Figure. 7.1 shows the topology of this study and this experiment uses climate data as an example.

Figure. 7.4 and Figure. 7.3 show how NDN's caching helps data retrieval. This study requests files ranging from 1GB to 1.4GB, sizes around the median CMIP5 file size [166]. The number of consumers varied from 10-30 and each consumers requested 10-40 files. The size of the cache at the NDN gateway was fixed at 40,000 chunks. Figure. 7.4 shows that the cache hit ratio increased as the number of consumers increased, as expected. Increasing the number of consumers increases the probability of requesting the same file by more consumers, and thus the cache hit ratio increases. In Figure. 7.3, increasing the number of requested files decreases the probability of requesting the same file by multiple consumers, hence the drop in cache hit ratio. As for cache replacement policy, this experiment used Least Frequently Used (LFU) policy, which takes into account content popularity.

Figure. 7.4 compares NDN-supported retrieval with contemporary techniques [113]. A node in Korea fetched a 4GB file from the NDN testbed machine at CSU over NDN. It also fetched the same file from the closest ESGF node at NCAR (located in Boulder) over HTTP and FTP. As Figure. 7.1 shows, throughput of NDN-based retrieval improved with increasing packet size, and caching. The throughput saturated at 140 Mbps using the 1.5-MB packet size without caching and 600Mbps with caching. The same CMIP5 file was consecutively fetched by the consumers along the NDN testbed routing path using classical delivery techniques (HTTP and FTP) . HTTP and FTP-based deliveries had throughputs of 140–150 Mbps. With caching, the NDN-based delivery was able to achieve a throughput over 600Mbps, approximately six times more than what conventional delivery techniques could [113].

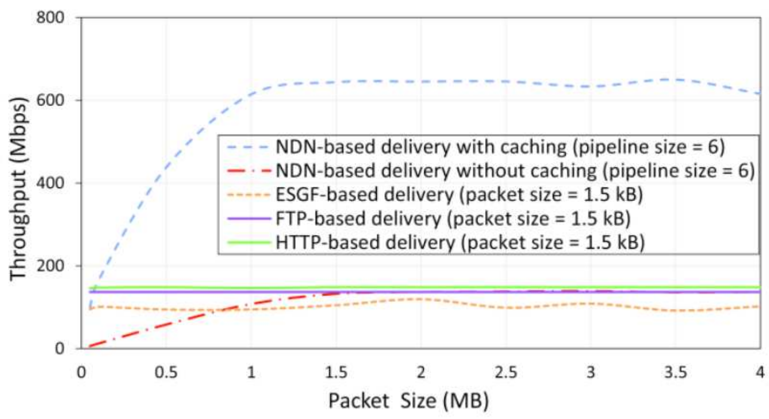


Figure 7.5: Caching Benefit of NDN

7.1.2 Multi-source Retrieval

Supporting movement of large data can be very challenging, and requires parallelism in several dimensions, such as storage, network, and applications [115] [72]. Today, high bandwidth transfer is achieved by concurrent retrieval of several files or retrieval of a large file in parallel [115] [57] [27]. However, due to the lack of knowledge of the underlying network, applications often have to rely on middleware or manual tuning of transfer protocols and associated parameters [4] [80] [82].

If multiple producers are reachable from a node, NDN can route packets simultaneously over multiple interfaces for fast data retrieval. NDN-supported applications may create different threads and assign each thread for downloading a single file. Alternatively, NDN may ask for different "chunks" of the same file in parallel. Depending on how the network is configured, these Interests might be forwarded to different sources of the same data (assuming the content is replicated) . As Section 2 mentioned above, the appropriate strategy depends on the application; for example, an application requiring high redundancy might forward the same interests on all available interfaces. Another application trying to maximize the transfer rate might load-balance the requests between all available links.

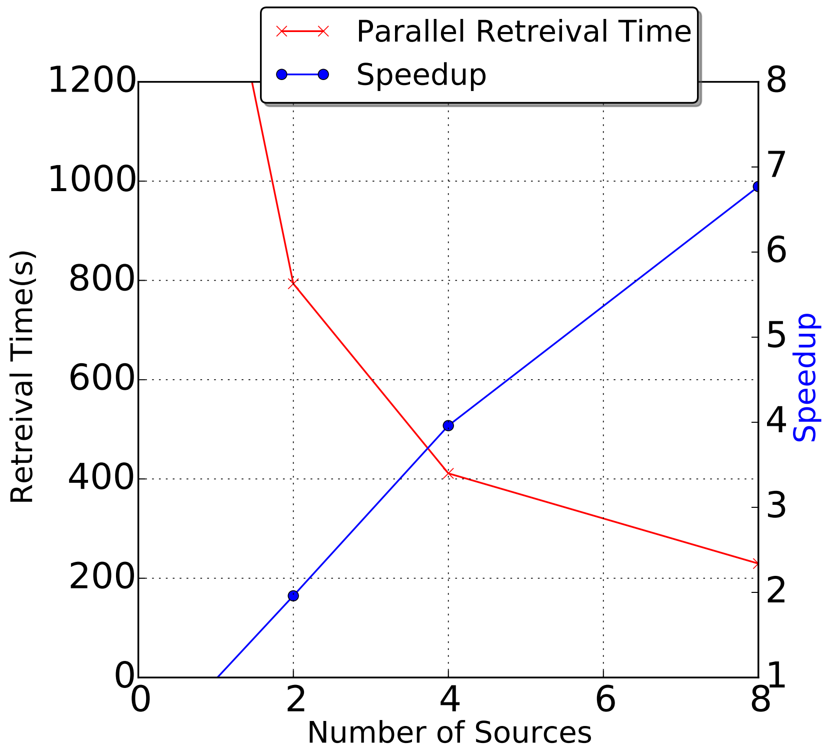


Figure 7.6: Scalable Retrieval using NDN

For high-performance applications, it might be desirable to retrieve content from multiple sources in parallel. Towards this goal, this work implemented a simple round-robin load balancing strategy in the network. The topology for this example was simple - a requester and eight data sources connected through an NDN router. The strategy is simple; each time an interest comes in, one outgoing interface that can satisfy the request is chosen at random. Note that this method does not consider all available interfaces for forwarding the Interests, only those which matches the Interest namespace (e.g., /CMIP5) and can satisfy the incoming Interests. This work replicated the climate dataset over multiple sources, started retrieval in parallel using our prototype implementation. Figure. 7.6 shows that the retrieval time reduces (almost) linearly as the number of sources increases.

Figure. 7.7 shows that NDN supports graceful degradation as servers fail. This experiment set the cache sizes to zero; as data is retrieved from multiple sources, one or more nodes were randomly failed. As expected, the content remained available until all data producer node failed. In the case of parallel retrieval, not only could NDN gracefully support service degradation, but it could also automatically redistribute the load among surviving nodes.

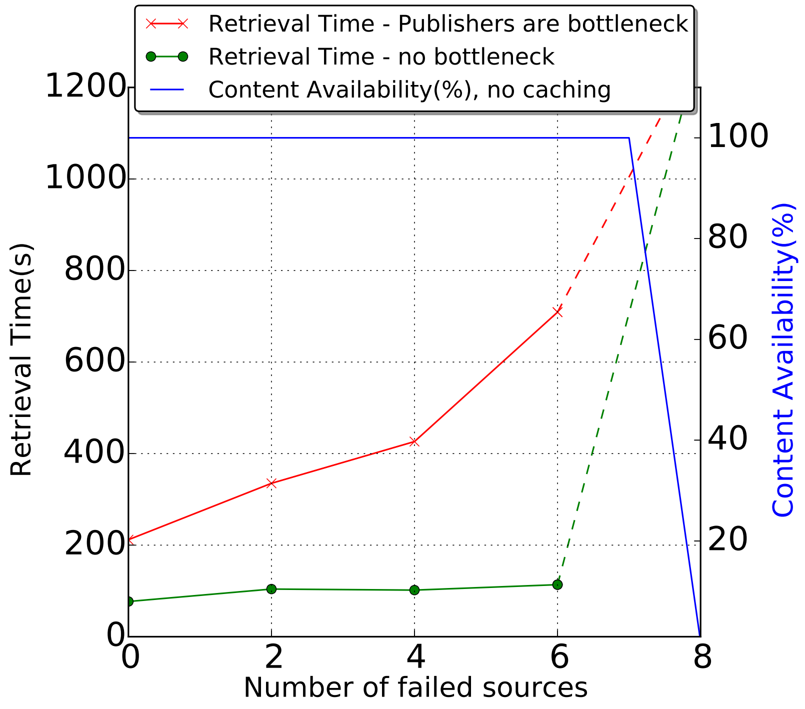


Figure 7.7: Content Availability and graceful degradation for retrieval using NDN

More sophisticated strategies can further improve data distribution. Instead of simple round-robin strategies, the network may be able to choose the “Best” data source for content retrieval, as the following section discusses.

7.1.3 Retrieval from “Best” Data Source

NDN is capable of choosing data sources that are “best” for an application. The definition of “best” is application and context specific. For example, as Chapter 3 discussed before, NDN can redirect requests to the data source with the lowest latency. This experiment replicated climate data over five data producers; two in China, one in France, one in the US, and one in Germany. These locations were deliberately chosen to match actual climate data server locations. This work used an NDN strategy that finds the lowest latency path from a client to its nearest data producer using the following procedure: on receiving the first Interest for a namespace (e.g., **</xrootd/data1>**), the strategy sends it over all matching interfaces (multicast) . Once data comes back, it records the RTT of each incoming Data packet before forwarding it downstream. It then ranks the faces based on RTT and uses that ranking to forward subsequent Interests. Periodically the strategy tries out other, lower ranked interfaces, and as Interest/Data exchange continues the strategy adjusts the ranking based on new observed RTTs. At any given time this strategy chose the lowest latency path for data retrieval [164].

Figure. 7.8 shows how this strategy can help with forwarding the requests to nearer servers. In the original request log [161], server 6 was the only producer and served 100% requests. With the new strategy, server 6 received only 0.03% of the requests, and the remaining requests were redirected to other servers closer to the clients. The delay based strategy redirected 96% of the requests to the two servers in China (server 1 and 2), freeing up resources at the LLNL node.

Choosing the server with the lowest latency also reduced the delay at the clients. Figure. 7.10 shows the mean delay at each client. Most clients saw a reduction in mean delay, but the level of reduction is also interesting; the mean delay for Client 3 was reduced from around 200ms to around 25 ms, an order of magnitude improvement. Other consumers also saw a significant reduction in latency. Note that this only an example strategy. In a window-based protocol (such as TCP), sudden changes in latency can adversely affect throughput [49]. With NDN, the effect of such changes can be normalized by selecting a server with low latency. In addition, an enhanced strat-

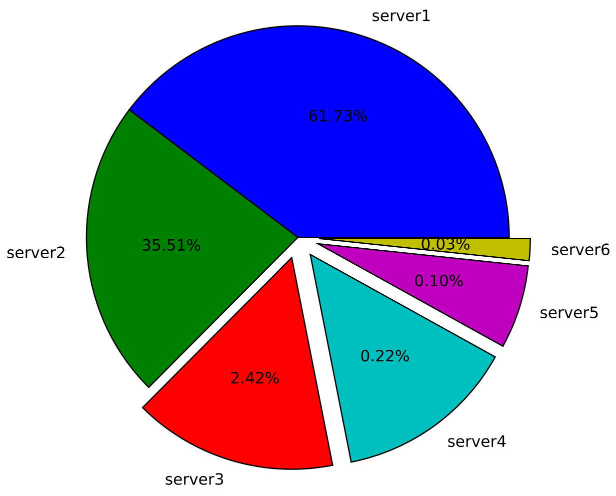


Figure 7.8: Percentage of requests served by each server. Server 6 is the original data producer.

egy can take into account application processing delays, loads on server, and other parameters that might slow down a transfer.

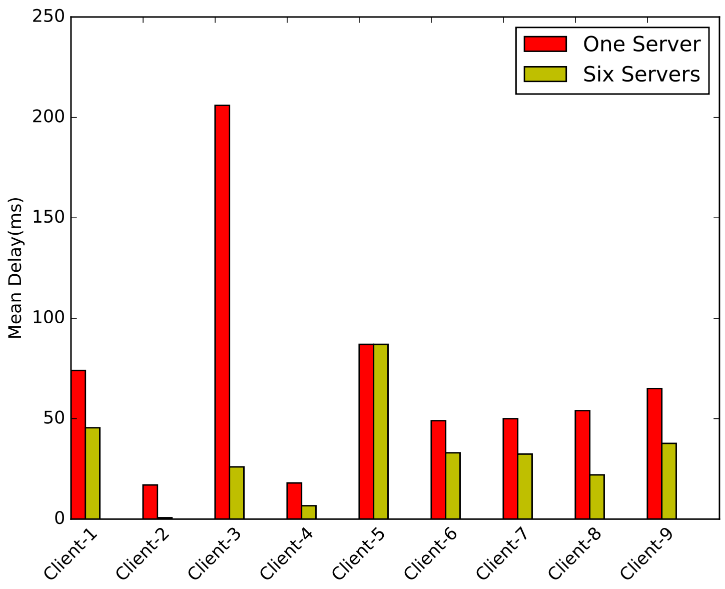


Figure 7.9: Mean delay of retrieving a data packet at clients

The strategy the previous section described is not the only strategy for large data transfers. In addition to high-speed, CDN like services and NDN network can also provide services such as strategic caching, retrieval from a peer, and managing requests to exploit in-network caching, as the following sections discuss.

7.1.4 Creating high-speed on-demand path

Scientific communities are globally distributed and routinely transfer traffic among various sites. Sometimes existing bandwidth between two nodes is simply not enough to satisfy a request by a given deadline. At the same time, creating permanent, high-bandwidth paths between all site pairs is not economically feasible. Moreover, as scientific data flows are often bursty [167], creating permanent paths is far from optimal. The scientific communities address this short-term bandwidth shortage problem by creating temporary, high-bandwidth paths for large data transfers.

ESnet's On-Demand Secure Circuits and Advance Reservation System (OSCARS) [80] is a service that allows users to create such guaranteed bandwidth-reserved paths. However, users are still responsible for knowing the endpoints, creating paths, and scheduling transfers. Arbitrary creation of reserved paths may create conflict between users, and the network resources may not be optimally utilized [164]. Intelligent strategies allow NDN to provide a network-driven approach to path creation. As Figure. 7.11 shows, an NDN strategy can invoke OSCAR's path creation mechanism when the available bandwidth on existing paths is not sufficient. If the path creation succeeds, the strategy adds a new route to the NDN's forwarding base. Since NDN consolidates requests for the same data, a fast path can potentially speed up other best-effort traffic if such flows are temporally close to the high-speed flow. Note that strategies are not constrained to using a specific lower layer protocol (such as OSCARS) but can interact with any other protocol or an even an SDN controller to create similar high-speed paths.

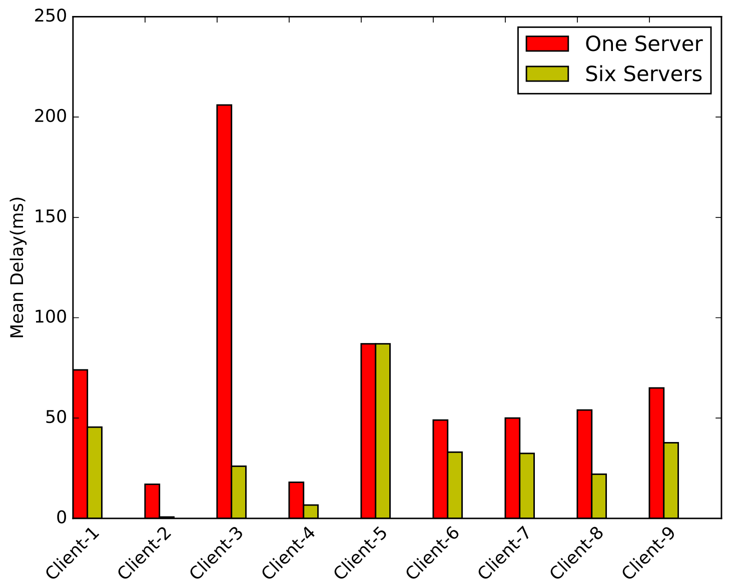


Figure 7.10: Mean delay of retrieving a data packet at clients

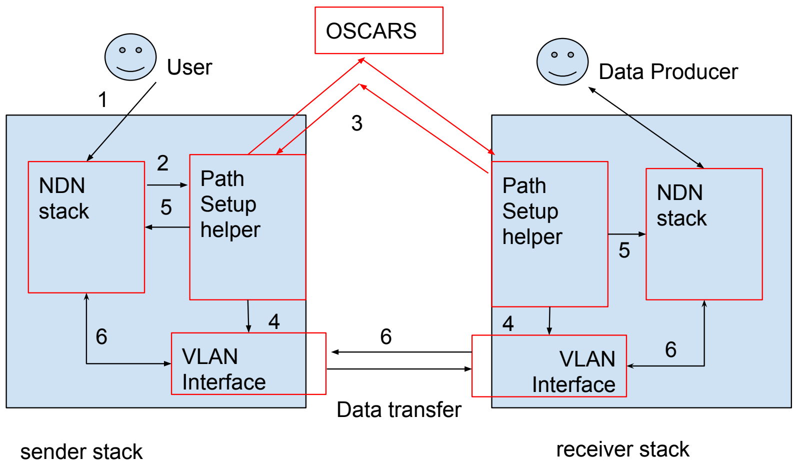


Figure 7.11: OSCARS with NDN

7.2 Time-constraint challenges

Proliferation of sophisticated scientific instruments, observational and simulation capabilities continue to increase the volume of valuable science data massively. Besides, scientific communities are increasingly moving towards a global scientific collaboration model. The datasets are so big, e.g., the Large Hadron Collider (LHC) alone generates approximately 50PB data per year [119], no central facility is capable of providing storage and computational capacities required to sustain these global collaborations. Scientific communities, therefore, must transfer raw or derivative datasets to local or institutional computation and storage facilities. For transferring these large datasets, scientific communities have created dedicated scientific networks such as the LHC Optical Private Network (LHCOPN) [119] and ESnet. While the available bandwidths in these networks are significant, they are still insufficient to accommodate all requests at the same time [166]. Data transfers, therefore, must complete within a given deadline freeing up network resources for the subsequent transfers, currently accomplished by manually scheduled transfers, orchestrated high-speed paths, and ad-hoc end-to-end bandwidth reservation by the user. Arbitrary reservations can create contention among clients and suboptimal resource utilization [80].

The current way of addressing the challenge of deadline-based data transfer is not optimal [115] [80]. Scientists create end-to-end reserved paths manually for a duration that they think they might need for a given transfer. Scientists then try to complete their transfers within this reserved time slot. If a transfer fails, the scientist either looks for a new time slot or retry if the current slot is still available. Tools like OSCARS and AL2S [175] has simplified the process by providing APIs and web interfaces that scientists can use to create reserved paths. If the requested resources are not available, these systems throw an error, and the scientist (or the operator) can try to reserve another time slot, possibly with different parameters.

The users often make reservations in an ad-hoc manner. A user trying to reserve bandwidth must know the data source and the destination, request a reservation, and transfer data within the reservation's validity period. There are several inherent problems associated with this approach. The users need to make sure the chosen source and the destination are optimal, need to know the operational details of the network, its capacity, and the size of the data. From the network's point-of-view, the whole affair can be highly inefficient; if a user creates a reservation but only uses a small portion of the available bandwidth, the rest of the reserved bandwidth is wasted. Contents of the end-to-end flows are not reusable even if multiple reservations share the same underlying path and retrieve the same content. Finally, failures during data retrieval may lead to restarts and wasted resources.

Several previous works have built solutions on top of RSVP [197] that can reserve bandwidth for end users or clients. Such works range from Energy efficient bandwidth reservation [134] to layer 3 protocols such as ESnet's OSCARS [81], from co-operative bandwidth scheduling [143] to time-shifted advanced bandwidth reservations [150]. However, all these protocols are either theoretical since they are hard to implement and deploy in IP networks, or very complex since they need to implement end-user authentication, point-to-point reservation, and data transfer scheduling. As the following section shows, supporting deadline based data transfers is simpler with NDN. Towards this goal, the following section first describes an RSVP-like bandwidth reservation protocol over NDN. Later, using the high-speed transfer protocols and the bandwidth reservation protocol described below, this chapter shows how NDN can support time-constrained data transfers. Additionally, it also shows how NDN can trade bandwidth for in-network storage to better facilitate such transfers and optimize resource usage.

7.2.1 Bandwidth Reservation Protocol

Bulk data transfers are common in distributed, connected scientific communities with many simultaneous users. They move a considerable amount of data over long distance links. For some workflows, such data transfers can reach more than a Terabyte per day [31]. In addition to The scope of this chapter is confined to large scientific data transfers over single-domain science networks such as Internet2 or ESnet. This chapter does not consider inter-domain reservations which can introduce a large number of operational challenges such as traffic engineering policies, peering, and economic incentives [138]. This work does not propose alternate QoS mechanisms for NDN networks equivalent to IntServ or DiffServ in IP. Instead, this work provides the building blocks which such future NDN-based services will utilize. transferring data for archiving, replication, or local analysis, researchers also preplace data copies in caches around the world for efficient, CDN-like access [31]. Data pre-placement has been particularly popular with communities such as the LHC [31], which routinely places a significant amount of data near the users. Bulk data transfers are not overly sensitive to RTT but require a significant amount of bandwidth for a long time and no packet loss. However, a substantial amount of dedicated bandwidth for an extended period is challenging to acquire on public networks.

Even networks dedicated to scientific communities such as ESnet or Internet2 routinely encounter resource contention and congestion leading to data transfer delays [80] [115]. Satisfying transfer deadlines in such an environment often requires dedicated per-flow bandwidth allocation [191]. This work proposes a protocol to create hop-by-hop reservations in an NDN network. Figure. 7.16 shows a high-level overview of the reservation protocol; for setting up a reservation an NDN node sends a special reservation Interest that is forwarded hop-by-hop upstream. A tuple containing **<data name, requested bandwidth, start_time, deadline>** represents the reservation request. Each node has a reservation manager that checks if the requested bandwidth is available during the requested period. If the request is successful, the reservation manager forwards the Interest upstream. Upon reaching a data producer or a repository, the Interest brings back a reply with a success message. The reservation manager keeps track of the reservation using a reservation table similar to TABLE 7.2.

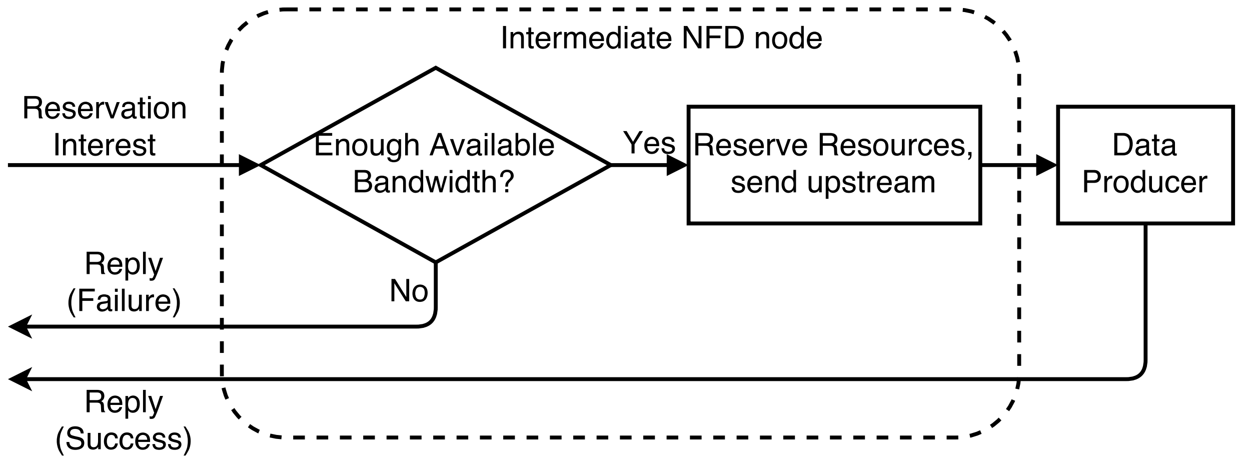


Figure 7.12: Reservation with NDN.

This section presents a high-level overview of an NDN-based Bandwidth Reservation Protocol. In addition to this protocol, the deadline based transfer protocol uses the NDN-based circuit-creation protocol and the multipath and delay based forwarding protocols. Figure. 7.13 shows a high-level overview of the deadline based data transfer protocols along with the building blocks. Descriptions of these constructs are deliberately high-level at this point, but later sections describe them in detail.

Namespace

Hierarchical NDN names align well with existing scientific namespaces; as chapter 5 shows. This work uses xrootd [33] namespace as an example. Data is published under a root prefix, e.g., **</xrootd>**. Different sub-namespaces under the root prefix

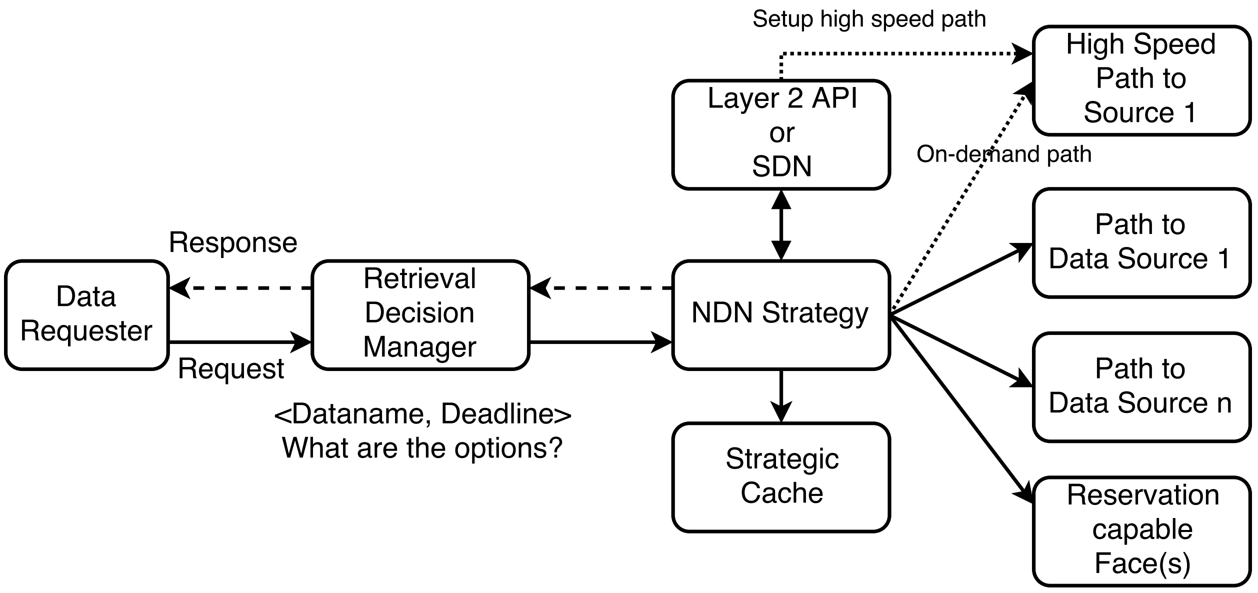


Figure 7.13: Overview of a deadline-based data transfer protocol.

identify data and various services. Figure. 7.14 shows such an example; actual data is served under **</xrootd/data>** while two other services are advertised under **</xrootd/query>** and **</xrootd/reservation>**. The first service allows applications to query the current network state for the **</xrootd>** prefix. The second provides a reservation service that a strategy can use to set up a reserved path for **</xrootd>**.

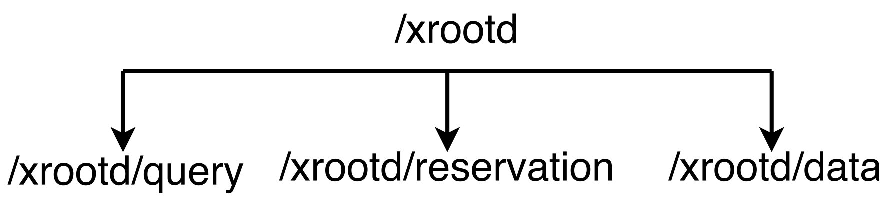


Figure 7.14: Namespace Design.

7.2.2 SCARI - A Strategic Caching and Reservation protocol for NDN

This section describes SCARI - a protocol that brings all the components described above together to create a complete bandwidth reservation and caching protocol for science data.

This work on SCARI draws inspiration from the RSVP protocol [197] in IP that provides receiver-initiated reservations. Since NDN natively supports multicast, in-network smart forwarding strategy, and caching, an efficient reservation protocol is easier to implement with NDN. While SCARI is similar to RSVP , there are two crucial differences between it and RSVP: (a) SCARI is per name prefix. Per-prefix reservation means transfers can share the reservation as long as they share some of the network paths, and (b) in SCARI, while the end-user (or the applications) uses a reservation, it is not in charge of creating or maintaining it. The second point is an important distinction. In the IP network, the receiver decides what resources it needs and requests a reservation. This work argues that the network, and not the user, should be in charge of reservations. In SCARI, the receiver only describes its requirements to the network, e.g., data transfer deadline in this work, but the network decides when to create reservations, how to maintain them, and how to optimize reservations to utilize in-network resources better.

Several works have built solutions on top of RSVP that can reserve bandwidth for end users or clients. Such works range from Energy efficient bandwidth reservation [134] to layer 3 protocols such as ESnet's OSCARS [81], from cooperative bandwidth scheduling [143] to time-shifted advanced bandwidth reservations [150]. However, all these protocols are either theoretical since it is hard to implement and deploy them for IP networks, or very complex since they need to implement end-user authentication, point-to-point reservation, data transfer scheduling. As later sections show, it can be much simpler to create and support reservations in NDN networks and at the same time, reduce network load, and more optimally use the available resources.

Table 7.1: Reservation Table in SCARI

Req Num	Prefix	Data Size	Avail. Band-width	Resv. Request	Avail. Cache	Start Time (s)	Deadline (s)
1	/xrootd/data1	1GB	10Gbps	1Gbps	1TB	1	10
2	/xrootd/data1	1GB	10Gbps	1Gbps	1TB	2	100
3	/xrootd/data3	1GB	10Gbps	1Gbps	1TB	1	10
4	/xrootd/data3	1GB	10Gbps	1Gbps	1TB	10	20

7.2.3 Protocol Design

This section discusses the protocol details of SCARI and explains the mechanisms for creating a hop-by-hop reservation in an NDN network, the protocol components, and message exchanges.

Protocol Overview

Figure. 7.15 provides a high-level overview of our protocol. In SCARI, the receiver expresses its requirements to the network. This protocol does not define these requirements, and the requirements can be specific requirements for bandwidth, delay, cache space, or something else. Appropriate strategies are required to interpret the requests and reserve these resources. This study uses bandwidth and in-network storage as reservable resources.

In this design, each router has a reservation manager (RM) . The job of this manager is to track available resources, schedule reservation according to local policies, aggregate reservations if possible, and strategically cache contents for later use. The reservation manager can be part of the forwarding strategy or a stand-alone daemon. Two types of reservation managers exist in this work (a) reservation managers located on end nodes (ERM), and (b) reservation managers located on router nodes (RRM).

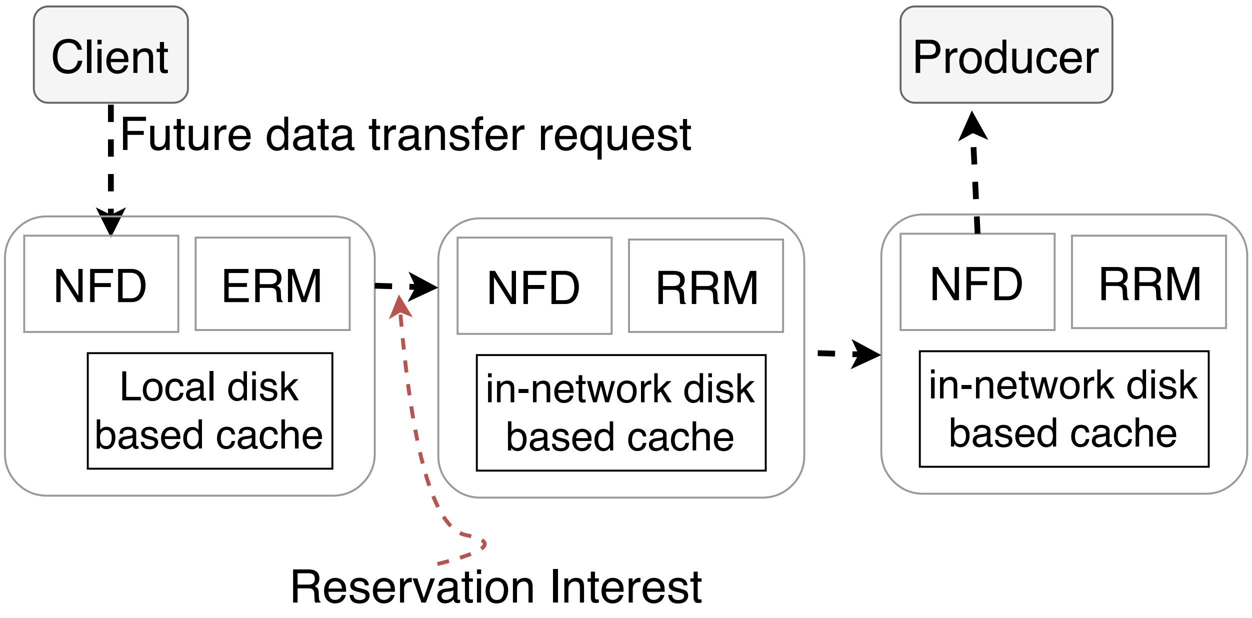

Client
Future data transfer request
Producer
NFD
ERM
Local disk based cache
NFD
RRM
in-network disk based cache
NFD
RRM
in-network disk based cache
Reservation Interest

Figure 7.15: Reservation Communication Overview.

The RRMs reserve resources for future reservations, aggregate reservations if they are temporally close, and strategically cache content if the requests are not temporally close (what qualifies as "temporally close" depends on the local policy).

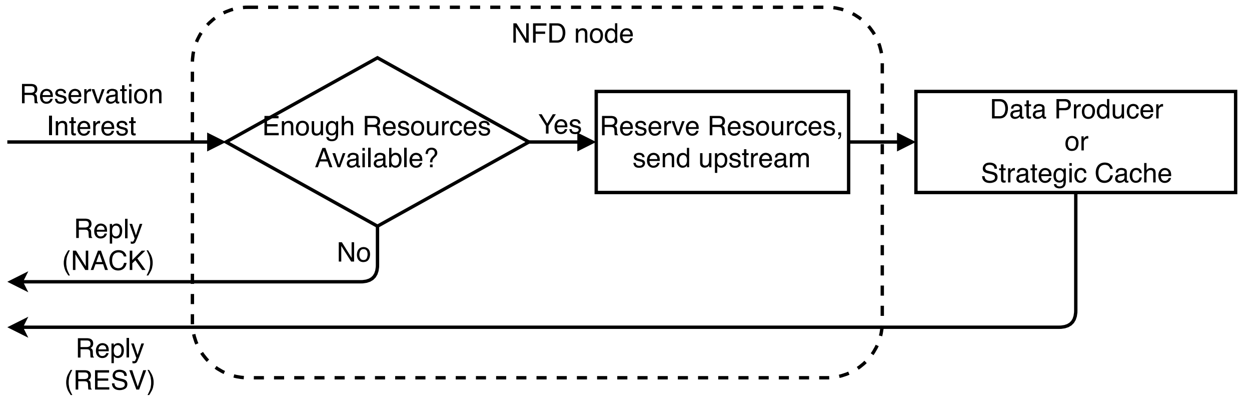

NFD node
Reservation Interest
Enough Resources Available?
Yes
Reserve Resources, send upstream
Data Producer or Strategic Cache
No
Reply (NACK)
Reply (RESV)

Figure 7.16: Reservation Protocol Details.

ERMs perform all services that RRMs perform and also act as a liaison between the network and the applications. In addition to reserving resources, it translates client requests to reservation Interests that it then forwards upstream. For example, an ERM can translate a request from the client indicating <data size, start time, deadline

>into a <data size, requested bandwidth, start time, deadline >tuple understood by RRMs. Besides, the ERMs can act as policy modules enforcing quotas and interact with the clients asking them to resubmit requests when requests fail.

Figure. 7.16 shows a high-level overview of the reservation protocol inside an RRM. To set up a reservation an NDN node sends a particular reservation Interest that is forwarded hop-by-hop upstream. If enough resources are available, the router reserves these resources, and forward the Interest upstream. If enough resources are not available, the network returns a NACK along with the reason for failure. Depending on the network and local policies, there might be several reasons why a router may refuse to reserve resources. For example, a router can refuse reservation if a request exceeds allotted quota, resources are not available, or other higher priority requests must be satisfied first. On receiving the NACK, the application and an ERM decide together what to do next. They might reduce the amount of requested resource, try the reservation at a later time, or abandon the effort.

The Reservation Table

In SCARI, all RMs maintain a reservation table. This table's functionality is similar to NDN's Pending Interest Table (PIT), but instead of keeping forwarding information, it keeps states about reservations. The table contains the prefix, the size of the reserved strategic cache, the reserved bandwidth, available bandwidth, and the start time and the deadline for the reservation. The reservations are based on name prefixes so that they can utilize NDN's request deduplication by caching and PIT and its name based forwarding. The table is extensible and holds any number of parameters dictated by various use cases.

Reservation Prefix Granularity

The granularity of reservation is very important. If a client makes a reservation under a top-level prefix, all other requests under the prefix share that reservation. For example, a reservation for "/xrootd" would lead to millions of datasets under this name prefix to share in-network resources. On the other hand, creating individual file level reservations will lead to an inflated reservation table. However, both of these scenarios can be perfectly fine depending on various use cases. For example, if a large volume of scientific data is being replicated between two LHC site, as it often happens in practice [146], a reservation for an all-encompassing namespace (e.g., /xtootd) might be desirable. This study uses a file-level reservation, but reservation granularity must be carefully considered depending on the use case.

Reservation Interest

The NDN reservation request is an Interest that carries a tuple in following format **<data name, data size, requested bandwidth, start_time, deadline >**. This implementation sends a reservation Interest using a special namespace, **</namespace/ reservation/>**; so a reservation Interest might look like **</xrootd/reservation/</xrootd/**
data1/data_size/start_time/deadline/bandwidth».

On receiving the reservation Interest the forwarding strategy checks if the request can be merged with another existing request. If it can be aggregated, the strategy sends a response indicating this. For example, if a strategy sees two overlapping requests, such as request 1 and 2 in Table. 7.1, it combines them since the same data flow can satisfy both.

SCARI uses three types of messages to communicate with the client. The strategy sends out a "Duplicate Reservation" ACK to the client when reservations are combined, and the client then proceeds to request data at the start time. If a reservation request reaches the producer node, it simply answers with a “RESV” message that signifies a successful reservation up to the data producer, and the client starts the retrieval at the start time. Finally, if enough resources are not available at a node, it sends out a NACK back. A NACK clears the pending reservation Interests at all downstream hops. On receiving this NACK, the client and reservation manager can decide how to proceed. While not implement it in this work, the intermediate nodes can also decide how to satisfy the requests on receiving a NACK. For example, it can create two smaller reservations to two different sources that satisfy the resource requirements.

Bandwidth Reservation

On receiving a reservation Interest, a node reserves the appropriate amount of downstream bandwidth for future incoming Data Packets. This work assumes that Interest packets are small and the path can support them without requiring bandwidth reservation. If the reservation is successful, the node forwards the reservation Interest upstream. If the node is the publisher, it returns a RESV message.

SCARI enforces bandwidth quota for each prefix by controlling the Interest forwarding rate. Assuming NFD can forward a finite number of packets per seconds, the protocol considers this number as a sharable resource and divide it appropriately among reservations. However, NDN Data packet sizes can be variable [19] and hard to predict. In the cases where data sizes are known, such as for science data, forwarding rates are easy to calculate. When returning data sizes are unknown, the forwarding strategy can predict how much bandwidth is used on the return path based on observed Data packet sizes. When the return data rate goes above the quota, the strategy asks the client to slow down Interest sending rate, either using a NACK or other congestion control mechanisms.

Cache Reservation and Strategic Caching

Unlike IP networks where bandwidth is the only reservable resource, NDN nodes can also reserve in-network caches. As a result, NDN can reduce bandwidth consumption at the expense of in-network storage. In SCARI, contents are strategically cached on a long-term, disk-based, in-network storage. This work envisions the caches to be disk-based and significantly larger than the in-memory CS, in the order of several TB or more. Which content to strategically cache depends on how many reservation requests arrive at a particular node.

This cache is different from a CS in the sense that it has more capacity and content are cached longer. However, knowing which content to cache for future requests requires foreknowledge and therefore, is not practical. However, for scientific workflows, this work anticipates that end users/applications would submit their requests before the actual data transfers, a common practice in scientific computing [132], allowing the network some time to optimize the transfers.

SCARI reserves cache space along with bandwidth for the reservations. If another request for the same data with an extended deadline comes in, the strategy is to simply cache the content until the new deadline. Request 3 and 4 in Table. 7.1 demonstrates this; without strategic caching, SCARI will need to create two separate reservations, one from time 1-10 Sec and another from time 10-20 Sec. However, if the path of these two requests intersects, with the help of strategic caching, the network can reserve bandwidth from t_1 to t_{10} and caches the content until t_{20} at the intersection. There is no need to create two reservations from the clients to the data producer, two reservation up to the first cache and one reservation from the cache to the data producer is sufficient, freeing up upstream bandwidth (from intermediate cache to content producer) that can serve other requests.

Strategic caching also can automatically optimize content placement through the network. For example, a node may decide to cache data for **</xrootd>** until $t_{deadline}$

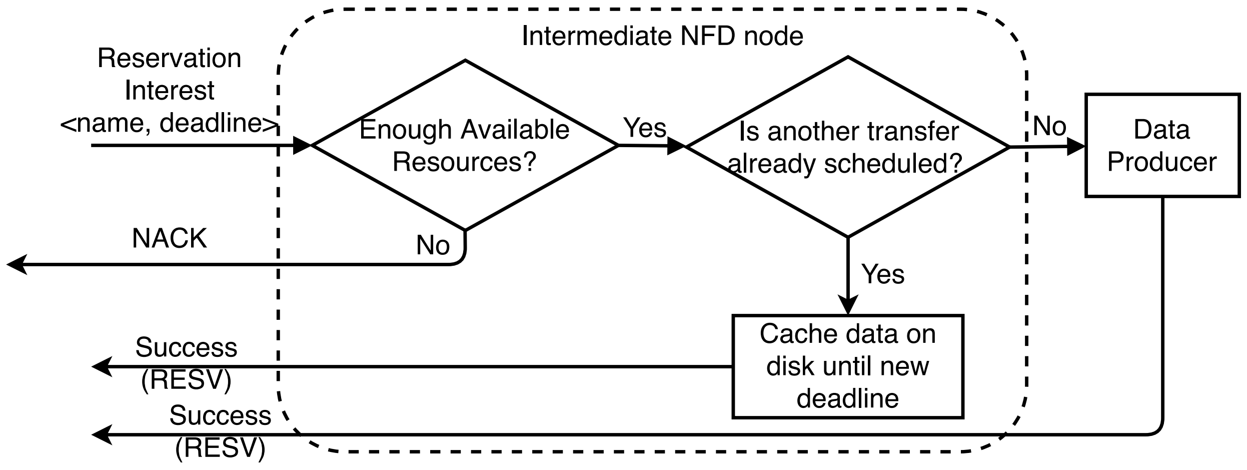

Intermediate NFD node
Reservation Interest <name, deadline>
Enough Available Resources?
Yes
Is another transfer already scheduled?
No
Data Producer
NACK
No
Yes
Cache data on disk until new deadline
Success (RESV)
Success (RESV)

Figure 7.17: Using reservation with strategic caching.

if it receives n requests through m different faces. This simple strategy places the content only at the nodes (e.g., R1 in Figure. 7.18 that lies at the intersection of future content paths. This method also frees up resources in the downstream router that does not lie at any intersection and does not need to cache this particular content strategically.

Since scientific datasets show a high degree of temporal and spatial locality [161], this work anticipates that in-network strategic caching is helpful for these data flows. To summarize, SCARI has two main benefits: unlike today, an end-to-end per-client path reservation is not required which frees up network resources. Second, SCARI can dynamically create in-network strategic caches without requiring prior planning and operator intervention. In-network strategic caches can trade bandwidth for in-network caches, thereby freeing up available resources.

7.2.4 Simulation Setup

Our topology is deliberately simplistic but should be sufficient to demonstrate the benefits our protocol offers; a more complex topology will result in more in-network strategic caches that will improve our results.

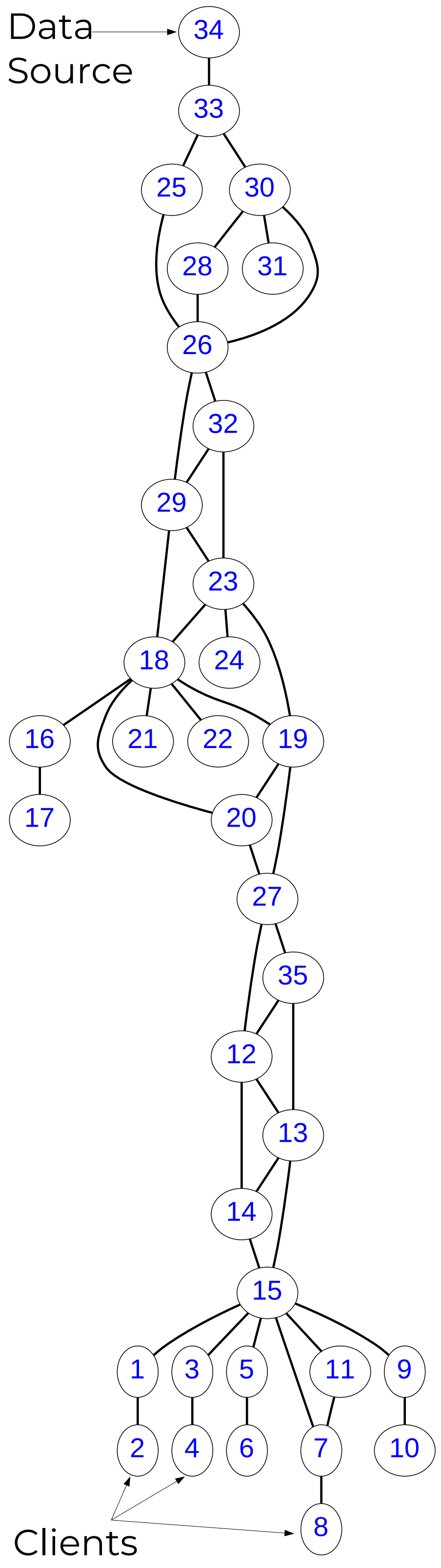


Figure 7.18: Evaluation topology for SCARI

In this work, we use an ndnSIM based simulation to evaluate our protocol from the perspective of the user, the network, and the data producer. To show the benefits of our protocol, we use a topology derived from a real data access log recorded at a single server [161]. Figure. 7.18 shows the topology that we used for our study. Since the original topology contained a large number of nodes, such large topology slows the simulation down, and we pruned the excessive nodes on the paths of the topology without changing the actual connectivity. The new topology contains 35 nodes, one server/data producer, and nine clients.

In our topology, node 34 is the data producer, intermediate nodes are routers with RRMs, and the leaves are the clients with ERMs. For this study, we assume the data source (node 34) and the consumers (leaf nodes in the graph) as part of the same network and we use shortest path routing for our simulation. Using this topology, we conduct our simulations for the three approaches: IP-reservation, NDN-without-caching, and NDN-with-caching. To simplify our simulation, we set the strategic caching size to unlimited, though we found the actual caches could be much smaller, around 500GB, for achieving optimal benefits.

We pick the top 999 requests from each client, a total of 8991 requests. 7965 of these requests contain the unique data names, and the rest are duplicates. The clients create and submit the reservation requests to their local ERMs that forward them upstream. For creating duplicate requests, we choose one client and keep its access log intact, but substitute some of the other clients' requests using randomly chosen requests from the first client's request set. Each client then sends the reservation requests using the times in the original log, with data sizes ranging between 1 MB and 49.4 GB.

In our scenario, all links had 1Gbps bandwidth. To allow for realistic deadlines, we calculate the deadlines based on TCP transfer times over a 100Mbps link. Before the actual data retrieval happens, the client must successfully reserve the bandwidth on the path towards the server. The actual data can be cached but the ACK or RESV messages are uncacheable. In actual use cases, we expect the clients to allow some additional time in their deadlines than is required for actual data transfers. To ensure timely completion of our simulations, we used large chunk sizes (20MB) . For NDN-without-caching, the reservation does not end until it times out, so the large chunks will not impact the results. For NDN-with-caching, we limit the outstanding Interest pipeline number as the ratio of chunk numbers to the reserved time, trying to minimize the effect the large chunks size.

7.2.5 SCARI Evaluation

This section evaluates SCARI from the perspective of the user or client, the network, and the data producer. The evaluation starts with the scenario where all reservations are stand-alone, labeled as IP reservation. However, if the requests are temporally close, SCARI can merge them in an effective multicast group. If the clients start retrieving the content approximately at the same time, only one request reaches the server, and the rest of the requests are satisfied by content flowing through the network. This section compares SCARI with and without reservation aggregation.

Figure. 7.19 shows the number of successful reservation requests. Currently, each request must reserve bandwidth on all the nodes along the path towards the server. The number of reservation remains constant around 70 (out of the 200 requests) even when many reservation requests are duplicates.

In contrast, with the same resources, SCARI can support more reservations - aggregating temporally close reservation requests shows an improvement in the number of accommodated reservations. Figure. 7.19 shows that when there is no duplicate request, number of successful reservations is equal to IP. However, as the number of duplicate reservation requests increases; with 60 percent duplicates, the number of successful reservations grows to 110 with SCARI.

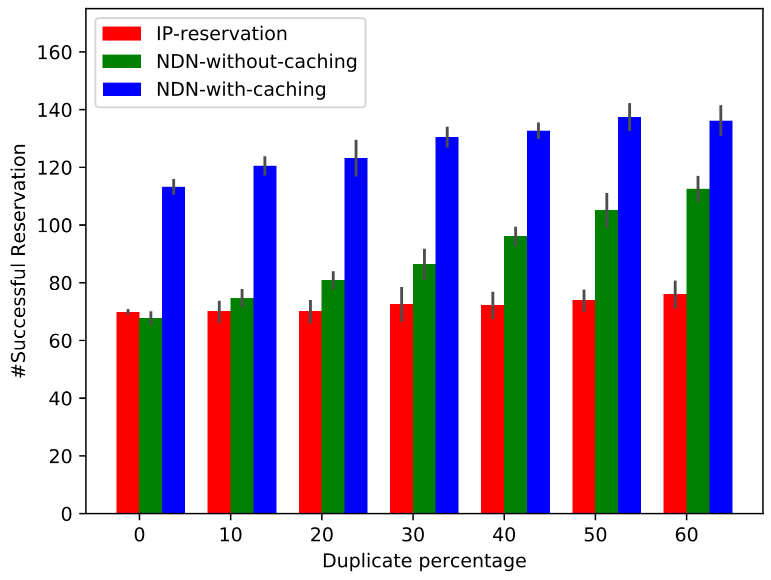


Figure 7.19: Number of successful reservation requests

With strategic caching, the results are even better. With strategic caching, 140 out of 200 reservation requests could be satisfied, effectively doubling the number of reservations that the network can support with the same bandwidth. This improvement is the direct result of strategic caching where the intermediate nodes cache data locally to free up the reserved bandwidth. Note that as the network grows in size and complexity and more nodes create strategic caches in the network and free up upstream bandwidth, the number of reservation that can be supported with the same amount of bandwidth should increase proportionally.

SCARI not only benefits the clients but also benefits the data producers. In Figure. 7.20 plots the number of reservation requests that reached the data producer. Figure. 7.20 demonstrate the number of reservation requests that arrive at the data producer. The simulation found a similar number of requests reach the data produce both for IP reservation and NDN-without-caching. When all requests are unique, the number for NDN-with-caching is much larger than IP-reservation and NDN-without-caching cases. This is an interesting observation; in IP and NDN-without-caching, the

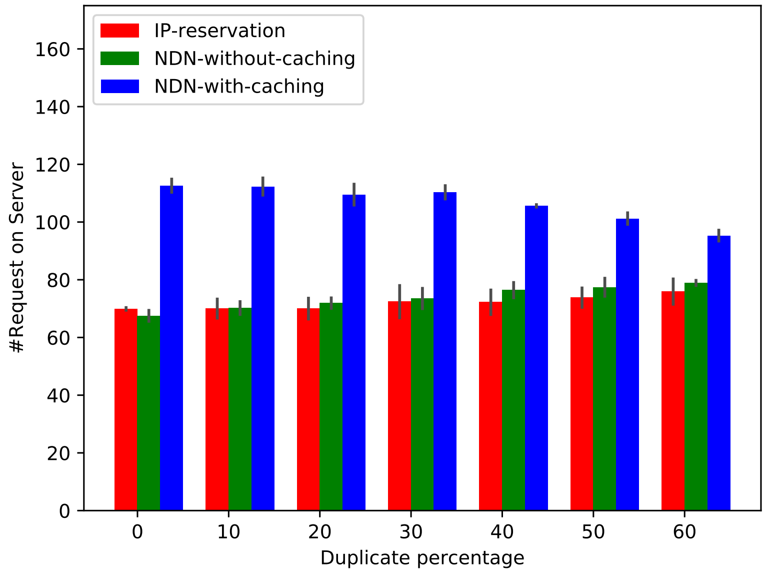


Figure 7.20: Number of reservation requests that arrived on the data producer

intermediate router does not have enough bandwidth and therefore rejects most of the reservation requests before they reach the data producer. As discussed before, with strategic caching, the intermediate node can release the previously reserved bandwidth when it can cache the content; trading storage for bandwidth frees up resources so that other reservation requests could go through. As duplicate reservation requests increase, the number of requests that reach the data producer decreases linearly for NDN-with-caching scenario since the in-network strategic cache serves some of these requests instead of the data producer, freeing up available bandwidth at the data producer.

Besides accommodating an increased number of reservations with the same amount of resources and freeing up bandwidth at the data producer, our protocol also benefits the network by reducing bandwidth consumption at the intermediate nodes. Figure. 7.21 shows the amount of available bandwidth over time at the bottleneck (node 33) with 70 percent duplicate reservation requests. The x-axis is the timeline of the whole simulation, and the y-axis shows the amount of bandwidth available at the in-

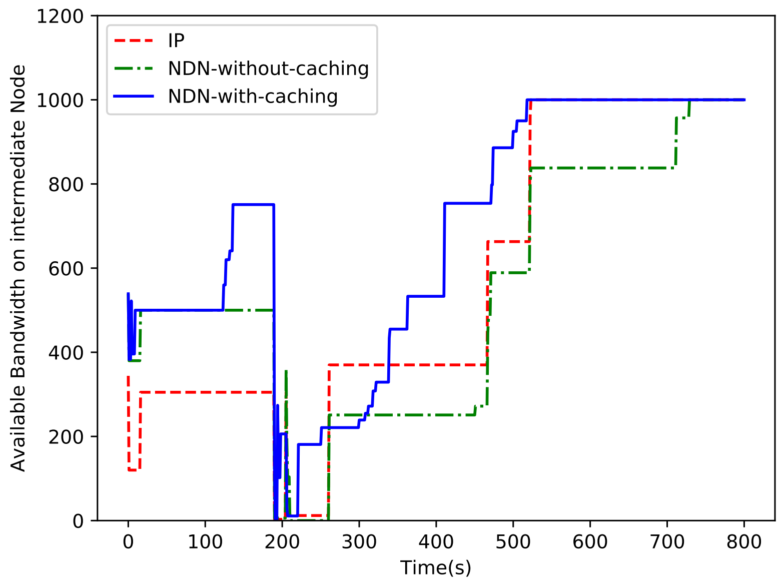


Figure 7.21: Resource utilization at the intermediate node

termediate node. Both NDN scenarios, with and without strategic caching, use less bandwidth compared to the IP scenario. Less resource usage means the network can support more reservations and data flows using the same amount of bandwidth.

While this section shows that SCARI can reduce resource consumption for all stakeholder, the users, data producers, and the network, more sophisticated cache optimization, data transfer scheduling, and advanced strategies such as multipath reservation may be able to reduce resource consumption even further.

7.2.6 A Deadline-Based Data Transfer Protocol: Design and Implementation

This section uses the NDN-based primitives discussed above to design a deadline-based data transfer protocol for both best-effort and reserved bandwidth data transfers.

The reference implementation of our protocol has three main components; a data requester/client, a per-node retrieval decision manager and custom NDN strategies.

The forwarding strategy controls intelligent Interest forwarding decisions and when necessary, reserves bandwidth, creates strategically placed in-network caches and interacts with the upper/lower layer protocols for dynamic path creation. The retrieval manager acts as an intermediary between the client and the strategy. In addition to communicating with strategies and the clients, the retrieval manager works as a policy module to enforce retrieval or reservation quotas. Policies are needed to ensure that applications do not force the network to use dedicated paths for all transfers by setting impossible deadlines.

The Reservation Table

The reservation table keeps track of the current and future reservation requests. Besides, the table also keeps track of available resources such as bandwidth. The table is extensible, and as the section 7.2.10 will show, can be used to reserve other resources such as in-network caches.

Table 7.2:
RESERVATION SCHEDULING TABLE

ReqID	Prefix	Requested StartTime	Deadline	BW
1	/xrootd	1463330393	1463355592	1Gbps
2	/xrootd	1463330519	1463355623	1Gbps

7.2.7 Component Interaction

In this protocol, the client notifies the network of its requirements by sending an Interest packet to the retrieval manager. The Interest takes the following form: <**data name, deadline, dataset size, hard/soft deadline flag**>. The name of the dataset defines the requested dataset; the retrieval deadline denotes the latest acceptable time for data retrieval; a “hard deadline” flag means the deadline is non-negotiable while a “soft deadline” flag denotes best effort traffic. While transfer time for requests with soft deadlines is not guaranteed, the strategy still may use intelligent forwarding, e.g., multipath forwarding, to fulfill the request within a reasonable time.

The Interest also tells the retrieval manager the size of the data. While estimating dataset sizes is not easy for general Internet traffic, scientific data sizes are usually recorded in a catalog, and therefore relatively easy to estimate [72].

If the retrieval manager sees two or more requests with overlapping deadlines, it simply aggregates them and suggests a start time to the clients. Otherwise, the retrieval manager talks to the local NFD about possible retrieval options using a special query namespace. For querying retrieval options for **</xrootd>**, the retrieval manager sends an Interest to the NFD with the following structure: **</xrootd/query/ </xrootd/data1/start_time/deadline/deadline_type»**. On receiving this Interest, NFD compiles a list of options and returns it to the retrieval manager.

If none of the existing options can meet the deadline, the network must create a new high-speed path. This work built an NDN strategy that works with Layer 2 protocols (or SDN) to create such a path, as section 4 discussed. Once the strategy on a node decides that a new path is needed, it calls the OSCARS API which then creates a new VLAN between the node and a data producer. Once the high-speed path is set up, the strategy uses it as simply another available link.

This protocol is generic and should work in both NDN-only networks, and NDN overlays over IP. No additional mechanism is required for fulfilling requests with soft deadlines; however, meeting hard deadlines will require QoS guarantees not only from the NDN entities but also from the underlying IP routers.

7.2.8 Fulfilling Requests with Soft Deadlines

Figure. 7.22 shows the decision path for Interests with soft deadlines. If the Interest is for **</xrootd/data1>**, the custom NDN strategy looks up the list of faces in the FIB that can be used for retrieving **</xrootd/data1>**. If the Interest is under a namespace that was not previously used for data retrieval, the strategy fetches a few chunks using each matching face and records the following information for each face: **<FaceID, RTT, Max Bandwidth>**. The strategy then compiles retrieval options and sends it to the retrieval manager, which in turn notifies the client. Instead of sending binary yes/no response to the client, the protocol replies with more detailed return values along with a suggested start time. For this implementation they are (a) the request can be satisfied, and the client starts retrieval immediately; (b) the retrieval can be satisfied only with an extended deadline; if the new deadline is acceptable, the client adjusts the deadline and requests again; (c) the retrieval is aggregated and the client starts retrieval at the suggested time; and (d) the request can not be satisfied. Such fine-grained information may enable the clients to make intelligent decisions, a feature that is not available today. However, since the network condition may change after the initial reply, there is no guarantee that the network will be able to satisfy the soft deadline. However, in the case of soft deadlines, this is implicitly understood. If data must be retrieved by a strict deadline, scientists may use the "hard headline", as the following section discusses.

7.2.9 Fulfilling Requests with Hard Deadlines

While strategies such as multi-path may work well for best-effort traffic, the only way to guarantee timely completion of large data transfers is to create a reserved bandwidth path [180]. In case the deadline is "hard", the strategy must create a reserved path to a publisher or a cache. In this implementation, the application sends a reservation Interest using a special namespace, **</xrootd/reservation/>**. A reservation Interest looks like **</xrootd/reservation/**
</xrootd/data1/start_time/deadline/bandwidth>>.

On receiving this Interest, a node reserves the appropriate amount of downstream bandwidth for future incoming Data Packets. This work assumes that Interest packets

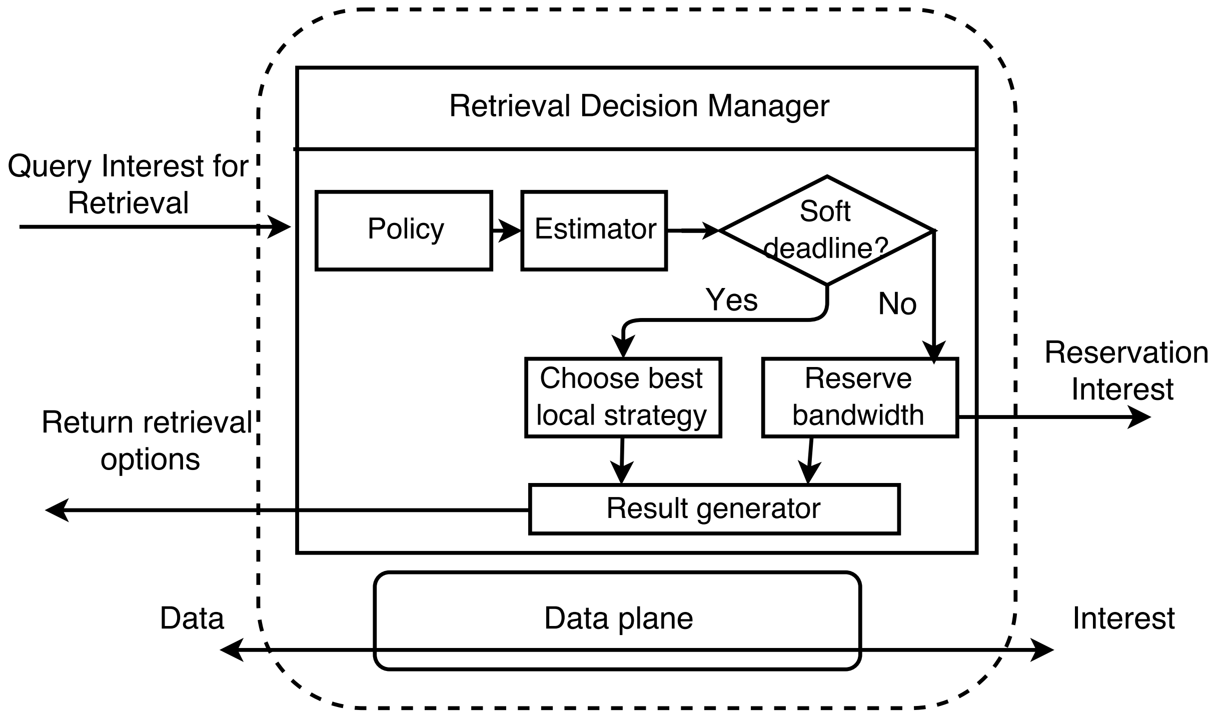


Figure 7.22: Strategy decisions on client node's NFD.

are small and the path can support them without requiring bandwidth reservation. If the reservation is successful, the node forwards the reservation Interest upstream. If the node is the publisher, it returns a success message.

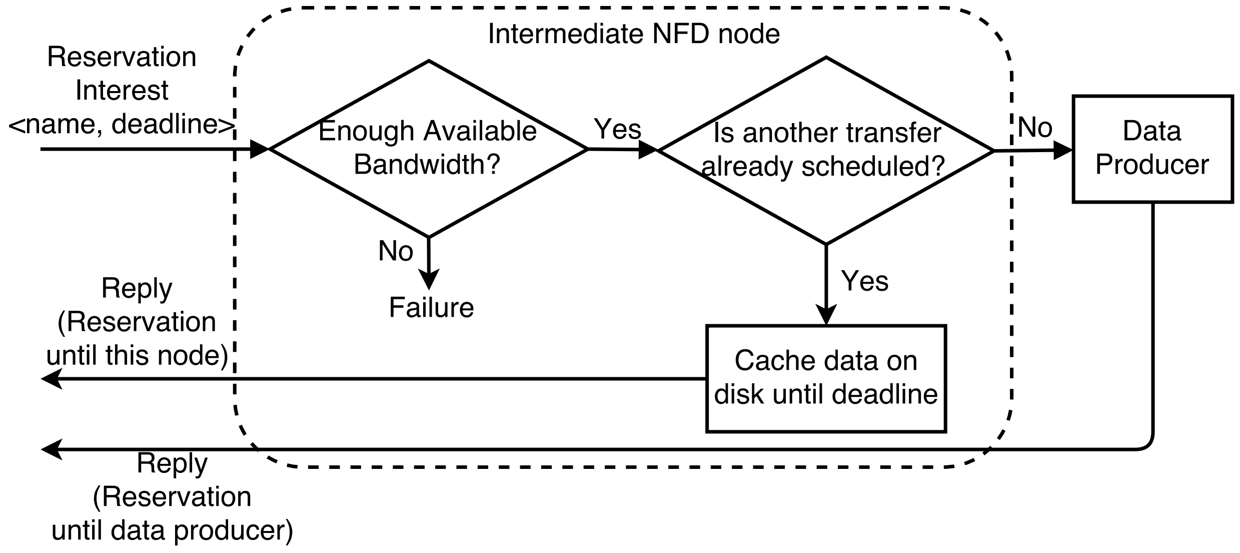

Intermediate NFD node
Reservation Interest <name, deadline>
Enough Available Bandwidth?
Yes
Is another transfer already scheduled?
No
Data Producer
No
Failure
Yes
Reply (Reservation until this node)
Cache data on disk until deadline
Reply (Reservation until data producer)

Figure 7.23: Using reservation with strategic caching.

The reservation protocol can be tweaked to create intelligent data dissemination strategies; Figure. 7.23 shows an example. If a new request overlaps with an existing one, our strategy merges the requests and sends back a reply indicating success and the time of the reservation. Note that in this case the reserved path is created only between the client and the replying node. In addition to performing this simple aggregation, an intelligent strategy may create a temporary cache in the intermediate nodes. For example, a node may decide to cache data for **</xrootd>** until $t_{deadline}$ if there are n requests scheduled between now and time $t\_deadline$. Since scientific datasets show a high degree of temporal locality [162], in-network strategic caching is helpful for these data flows.

Our method has two benefits for scientific datasets: unlike today, an end-to-end per-client path reservation is not required which frees up network resources. Second, our strategy can dynamically create in-network caches without requiring prior planning and operator intervention.

7.2.10 Evaluation

The deadline-based data transfer protocol described in this chapter optimizes network usage by aggregating requests, caching, and intelligent request scheduling. This work analytically analyzes an xrootd [33] access log recorded from Apr 23th, 2016 to Apr 30, 2016. The logs had 114K unique requests from 267 users, recorded at a ten-minute interval. The access logs showed a high degree of duplicate requests; users requested only 1871 unique datasets over 114K requests; so on average, each dataset was requested sixty times. Duplicate requests can occur in xrootd for popular datasets or if transfers fail, which then automatically triggers another request for the same data. NDN can optimize HEP data flow by de-duplicating requests using our deadline based protocol. For combining the requests, this work introduces a "scheduling window"; requests falling within this window can potentially be combined.

To investigate temporal locality within our scheduling window, this work first separated the requests in (arbitrary) six-hour bins. The actual window will depend on network capacity, storage, and individual workflows. Figure. 7.24 shows the number of duplicate requests over time; each dot represents the number of requests for a specific dataset in a six-hour window. The evaluation found that many datasets were requested several thousand of times and over 50% of the requests have one or more follow-up request(s) within 10 minutes. Having so many duplicate requests in the log is good news for our strategy since they can be combined efficiently.

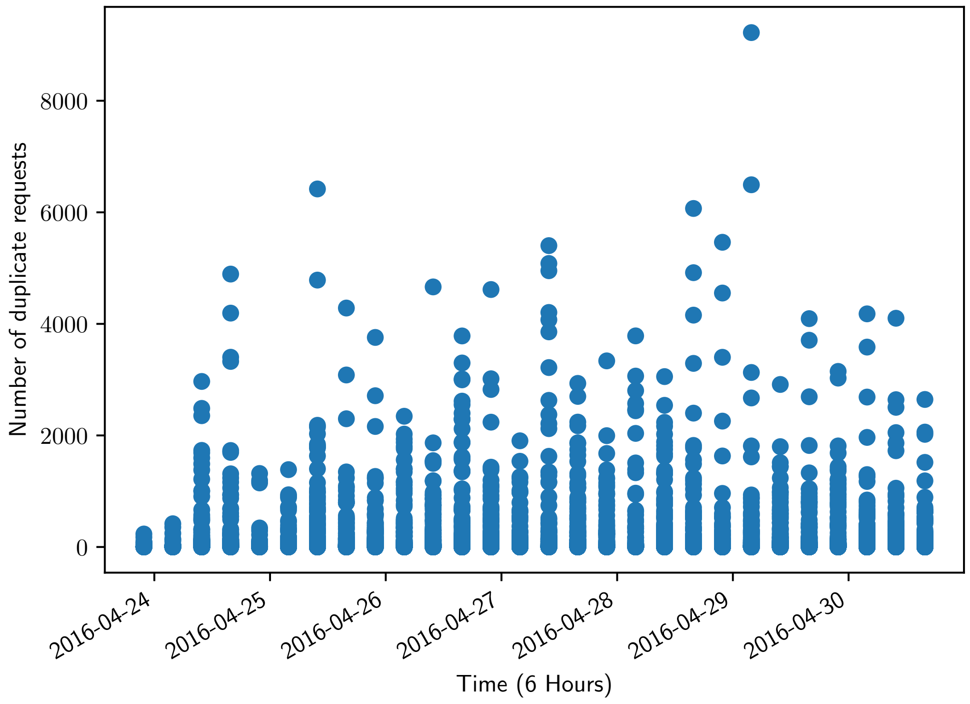


Figure 7.24: Duplicate requests for individual datasets over time.

The actual file sizes were not available in this log, as a result this work assumes each request was for a 2GB file, the average file size found in the log used for these experiments [166]. Depending on the file size and the request pattern, the bandwidth savings in Figure. 7.25 will increase or decrease but this figure represents a rough estimate of how much bandwidth savings can be expected in an NDN network.

Intelligent request scheduling allows NDN to retrieve only one copy of the data that satisfies all requests for the same data. To compare IP's bandwidth consumption with NDN, this work first calculates the amount of bandwidth needed for each request and then calculate the total aggregate bandwidth required to serve all requests over six-hour periods. Figure. 7.25 shows that the total bandwidth requirement for xrootd is very high, with peaks at approximately 64 Gbps. This work also calculated how much bandwidth NDN could save compared to IP when requests are aggregated; it takes the same requests over 6 hours, de-duplicate the requests and calculate the total bandwidth required to serve them.

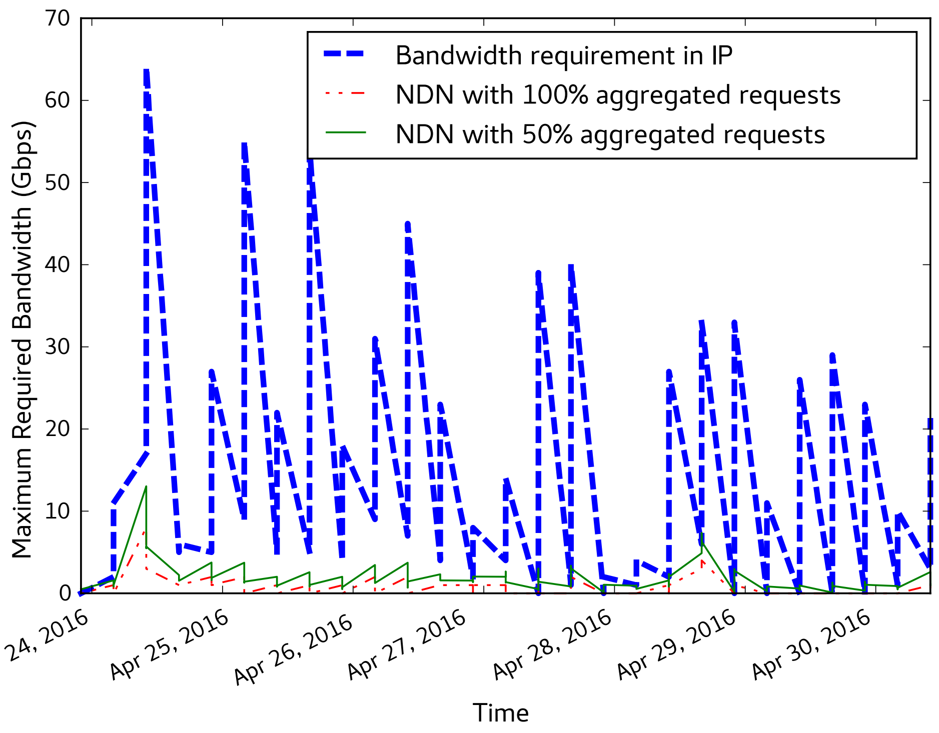


Figure 7.25: Reduced bandwidth consumption with NDN.

Looking from the servers' point of view, Figure. 7.25 shows overall bandwidth demand of the system in two scenarios. First one is the best case; if the protocol can aggregate all duplicate requests over 6 hours, the max bandwidth requirement at the servers drops from 64 Gbps to around 8.2 Gbps, an 85% reduction. However, this work acknowledges that not all requests can be aggregated; some requests might have very tight deadlines and need to be served immediately. Even if the protocol can aggregate 50% of the duplicate requests, the bandwidth requirement comes down to 13.2 Gbps, a 79% reduction. This result shows that even some degree of de-duplication can go a long way in reducing resource consumption for scientific data flows.

7.3 Usability Challenges

In addition to high-performance and deadline based transfers, NDN can provide better usability. As explained by the following sections, NDN can push computation to the source of data. A significant amount of research [181] [195] [44] exists on pushing computation to data using NDN that reduces the amount of data transferred, enable distributed computations at the network's edge, and pushing lambdas to the data source. The goal of these work is not to discuss all of them - but merely to show that a name based network can support operations necessary for large science data communities. Smarter operations can also be implemented using NDN, as required by individual workflows. The following sections discuss two such operations, remote subsetting and remote data staging.

7.3.1 Remote subsetting

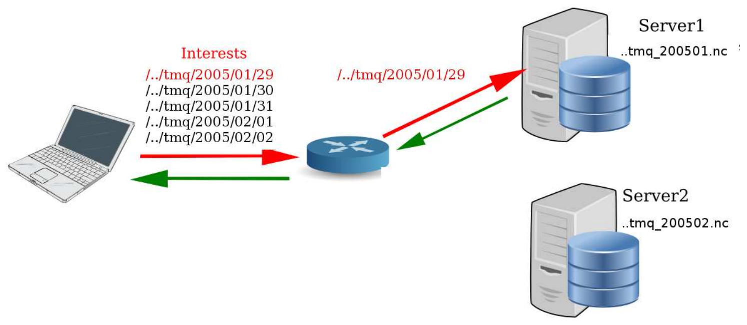


Figure 7.26: NDN remote subsetting

Remote subsetting can be useful for scientific communities. Instead of downloading the full dataset and running computations on them, remote subsetting allows scientists to fetch only a portion of the data that is required for a particular computation. This approach not only speeds up the total data transfer needed for a particular computation, but it also reduces the amount of data flowing over the network.

Existing applications and protocols such as OpenDAP [61], GridFTP [27], and DataCutter [36] provide remote subsetting capability. However, these applications also have several drawbacks. OpenDAP requires the data to be converted into a specific format suitable for processing. This requires more processing overhead. OpenDAP query execution model is sequential, and it does not support parallel query execution. When data is distributed, this poses a problem. Moreover, OpenDAP only supports specific types of queries [61]. GridFTP can provide a portion of a file but can not run complex subsetting tasks on data. More importantly, these applications do not provide generic functionality and either domain specific or need to be configured manually.

NDN, on the other hand, does not need any additional steps for supporting remote subsetting. Any query can be embedded in the Interest name, and NDN transports the Interest to a publisher. Embedding the query in the name allows various applications to merely add their domain-specific queries to the Interests. Each application does not need to implement their domain-specific functionality separately. When a name reaches a data publisher, it is up to the publisher to interpret the name and perform necessary actions.

Embedding the query in the Interest, therefore, enables NDN to support various applications and all possible queries. Additionally, just like parallel data retrieval discussed before, NDN enables parallel remote subsetting over multiple sources by distributing the subsetting Interests among the available replicas. The resulting subsets are transported back assembled by the application.

7.3.2 Staging Data

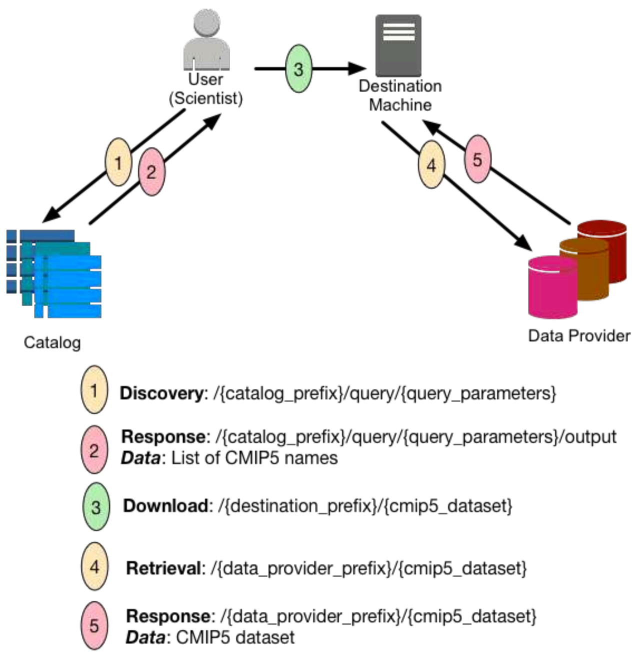

User
(Scientist)
3
Destination
Machine
1
2
4
5
Catalog
Data Provider
1 **Discovery**: /{catalog_prefix}/query/{query_parameters}
2 **Response**: /{catalog_prefix}/query/{query_parameters}/output
Data: List of CMIP5 names
3 **Download**: /{destination_prefix}/{cmip5_dataset}
4 **Retrieval**: /{data_provider_prefix}/{cmip5_dataset}
5 **Response**: /{data_provider_prefix}/{cmip5_dataset}
Data: CMIP5 dataset

Figure 7.27: Discovery and Retrieval Protocols

Another example of NDN's support for improved usability is staging data. Users often want to download data at the supercomputer where the computation will be performed. Right now, this process is cumbersome - a scientist often log-in to the account on a supercomputer, run a script to download data and continue to monitor it until the process finishes. Intelligent applications such as GridFTP [27] provides a more simplified process - the users identify a source and a destination, authenticates to both endpoints and asks GridFTP to perform the retrieval. While this approach works better than manually downloading datasets, several challenges remain. First, scientists need to know where data is located, set up the logistics of the transfer, and start the transfer manually. If underlying network condition changes or the data source fails, there is no way to react to these events other than selecting a new data source and starting the retrieval again.

NDN can simplify data staging - a user can stage data to a remote site by simply communicating the desired content name to a retrieval agent at the remote site. This operation is nuanced from the standard NDN use case as scientists may retrieve to a remote supercomputer. There is no need for synchronization between the remote site and the user - the retrieval agent retrieves data according to local policy and resource availability. Additionally, the network can react to changing conditions and data source failure. In addition to simple retrieval, the retrieval agent can also take advantage of deadline-based and high-performance data transfer options mentioned above. The user needs to specify the name and the deadline to the retrieval agent, and the data is downloaded to the remote (computing) site without any user intervention.

7.3.3 Common Interface in the Network

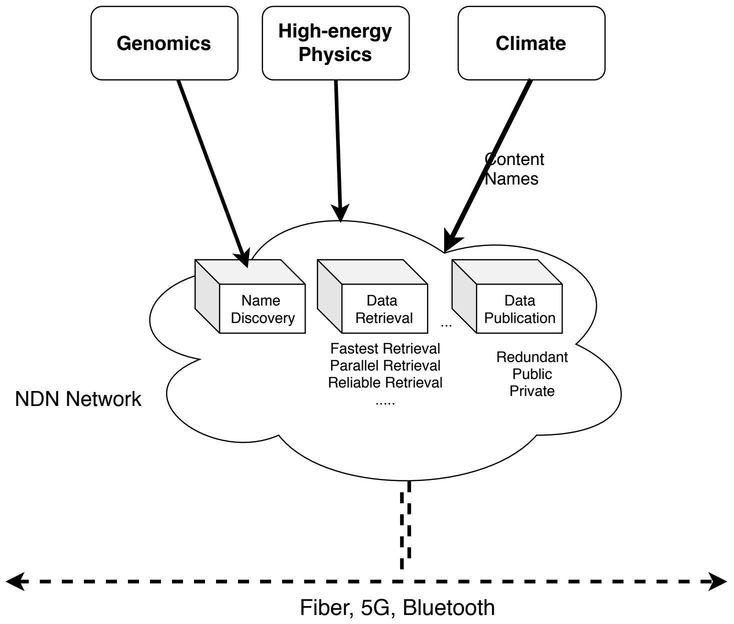


Figure 7.28: NDN common interface

In addition to the intelligent functions in the network, NDN can provide a standard interface in the network that provides common functionality across various scientific domains. For example, all scientific communities need to publish data, discover data, retrieve data, and run computations. Using names in the network allows the creation of domain-independent, reusable implementations for these functions.

The current method of implementing these functionalities at the application layer is not scalable. First, each community must develop, deploy, and maintain each of these software pieces. Second, since these applications typically make domain specific assumptions, these applications are very tightly bound to a class of domain-specific use-cases and are not reusable by other communities.

Even when applications are generic; for example, GridFTP can enable fast data transfers, tuning and maintaining the tools for each use cases is fairly time-consuming. NDN simplifies such tools by providing common strategies at the network layer. For example, any application trying to do parallel retrieval can use the parallel retrieval NDN strategy. In case more sophisticated strategies are required, they can be implemented and installed at the network by the network operator. Offloading the standard functionality in the network not only simplifies the applications but also improve maintainability of such applications.

While the application complexity is reduced, the network complexity increases. As discussed above, this study found that a few common strategies would be sufficient for most applications in the scientific domain, reducing the problem space. For the broader Internet, however, the interaction between diverse strategies in the network is not very well understood and requires more careful studies.

This section demonstrates how a name based network can provide intelligent functionality in the network. This chapter complements Chapter 5 and Chapter 6 that discuss naming and name discovery. The contribution of this chapter is to show how a name based network better support scientific workflow with new functionality in the network. Sought after features such as deadline-based data transfers, high-performance data transfers, and others are not difficult to implement with NDN.

More importantly, a name based network not only helps to simplify the applications but also improves resource utilization inside the network by aggregating requests and streamlining the use of in-network resources.

Chapter 8

Conclusions

We have entered an era when the next round of scientific breakthroughs are going to be supported by big data, new algorithms, purpose-built instruments, and powerful supercomputers. To accommodate this new paradigm Computing technologies and infrastructures must evolve very quickly to keep up with the oncoming deluge of data, to support novel algorithms and new types of computations, and to support increasing global collaborations. The network has always played a vital role in connecting global scientific communities and supporting scientific ecosystems, starting from the development of the World Wide Web at CERN. However, the primary design considerations of the current Internet was to connect resources(e.g., hosts).

The host centric model of the Internet does not support the increasingly content-centric requirements of the contemporary scientific workflows very well. For example, an application retrieving data from a storage host must know the data's host, host's location (IP address), connect to it, and monitor the retrieval. Other artifacts of the Internet, such as end-to-end paradigms are indispensable for ensuring proper function of the current application ecosystems over the Internet, but can adversely affect transfers when the data volume is enormous, in the order of Terabytes or Petabytes. This section summarizes the thesis which discusses these problems at length and shows that new Internet protocols help to alleviate these problems. This section also makes recommendations for the big-data and the networking communities to facilitate big-data.

8.1 Summary

Chapter 2 discusses the difficulties that current scientific workflows face. Scientific data requests are often temporally and spatially close. The current network does not allow easy reuse of data at the network layer, forcing the network to carry duplicate traffic and thus wasting resources. Security is in the applications - making it harder to cache content in the network and reuse them. The ability to retrieve content from multiple sources, pushing computation to the edge, network monitoring and management are some of the elements that are crucial for scientific communities and yet, not well supported by the network. As a result, scientific communities must invest time and resources to implement each functionality they need from scratch. The result is fragmented ecosystems of software solutions that painstakingly reimplement same functionality - basic operations for these communities such as fast data transfers, moving data closer to the users, and others must be implemented, deployed, and maintained separately. Additionally, the amount of data might be so significant in the near future, merely improving the network protocols, transfer speeds, and available bandwidth might not be enough to alleviate significant data problems. New features such as pushing computations to the edge, pre-caching content at the location of requests, and minimizing data transfer over the network will be necessary.

Chapter 2 also discusses Named Data Networking (NDN) . NDN repositions the network architecture to be content-centric, instead of host-centric. The hosts no longer have a host identifier, and neither do the packets. Network operations such as routing, forwarding, content retrieval, caching, and others are accomplished using content names. An application can express an “Interest” into the network, the routers forward the requests based on content names, and the data producers return data following the same path. Focusing on the content name better aligns the application requirements with the network.

Additionally, being a clean-slate architecture, NDN can utilize the lessons learned from IP's massive success. For example, a measurement and monitoring framework is likely to be integrated with the architecture, allowing applications to communicate with the network effectively. The in-network strategy layer allows the scientific communities to decide what type of intelligence they would need in the network and deploy them easily. Several other functionalities such as caching content in the network near the request hotspots, pushing content to the edge, utilizing the network storage to optimally fulfilling requests, and reducing resource consumptions in the network are easier to implement using NDN.

This thesis extensively studies three contemporary scientific workflows in three different domains - Climate, High Energy Particle Physics, and Genomics. Using these three big-data use cases, this thesis tries to understand the deficiencies in these domains and how the network contributes to these problems. This work finds that several critical operations in scientific domains are common across scientific domains. Operations such as fast data retrieval, the discovery of dispersed datasets, automatic failover to a replica data source can be useful to these communities. Additionally, advanced features such as data transfers from multiple sources, data transfer by a specific deadline, caching content for localized access are essential but hard to implement with the current technology. As a result, each of these communities implements their own software solutions at the application layer, leading to duplicate implementations of the same functionality. This thesis first hypothesizes that each of these functionalities (and more) are straightforward to implement using NDN and using a content-centric network aligns the functionality of the network with the application requirements.

To examine this hypothesis, Chapter 3 studies a representative dataset over three years that was obtained from a node serving climate data. This work shows that NDN can indeed benefit scientific applications by caching content in the network, automatically routing requests to the closest replica of the data, using multiple data sources, and enabling intelligent strategies in the network. These observations also have a profound impact on NDN's deployment - at the time of this study, very few researchers have looked into NDN's feasibility using real-world datasets - this study shows NDN can provide not only novel methods in the network but also maintain the appropriate level of performance. Validating NDN's properties using real data access logs and access patterns demonstrates the deployability of NDN in real networks.

After showing that NDN can support data-intensive science, Chapters 4, 5, 6, and 7 show how this might be accomplished. The naming of content is critical in a name-based network - Chapter 5 discusses different naming conventions in the scientific communities. This chapter notes that though scientific names are naturally hierarchical, aligning them with a name based networking paradigm is not trivial. Since naming of content affects how content is retrieved, accessed, and requests for these content are handled, trade-offs of naming conventions (such as placement of name components) should be carefully considered. In some cases, existing names are not sufficient, and the scientists must consult the metadata and actual data for creating names that are sufficient to utilize all the benefits that NDN offers. This chapter discusses how careless naming would create problems in the network and also hamper data discovery and retrieval. Using domain knowledge and name translation schema, this section demonstrates how a generic name translation framework can be built that can transform existing names into NDN compatible names. Finally, the chapter provides NDN naming guidelines for data-intensive science.

Naming data is only the first piece of the puzzle. Since names are bound to content, names must be discoverable - easily, quickly, and consistently. Chapter 6 presents a framework for publishing and discovering names in an NDN network. This framework still maintains a catalog of names, just like contemporary scientific applications do. However, an NDN based catalog need to maintain only the names (and not the locations) of the data, making the name publication and updation of the catalog lightweight. The lightweight nature of this catalog also beneficial for created synchronized instances of the catalog. On name publication (or updation) only the diffs need to be exchanged - since the catalog only hold names, the operation is lightweight, even when many names are included in the update. In this study, publishing 2.7 million names took less than 200 milliseconds. A MySQL database supports the backend of this system. The system can provide automatic failover for the queries, caching of results (so that same query is not executed repeatedly), and NDN provides routing of a query to the nearest catalog replica. This chapter also provides a front-end that helps scientists to query the published named using various methods such as auto-completion assisted typing, selecting name components, and browsing a name tree. Once the names are known, data can be retrieved by simply expressing Interests into the network. This work also created a survey and asked a small set of climate scientists to test the system and provide feedback, and the responses were overwhelmingly positive.

Chapter 7 describes the in-network protocols for supporting big-science. Once the naming and name discovery part are in place, the network can support data-intensive science through in-network intelligent protocols. What operations an NDN network should support is still under active research. However, looking at the various scientific communities, a clear pattern emerged. All scientific communities perform operations such as data publication, data discovery, fast retrieval, pushing computations to the edge.

Additionally, all these workflows exhibit specific properties, such as temporal locality of data access, data popularity following a Zipf distribution, and others. It became clear, at least for the scientific communities, migrating which operations in the network would be beneficial. In network caching, request aggregation, using names to push computations to the edge, high-speed and deadline based transfers are some of the properties that can help data-intensive science, and this chapter presents NDN based protocols for each of these. Using in-network caching and an intelligent strategy layer, this chapter shows how data distribution and access can be optimized using operations such as strategic caching, trading disk-space for bandwidth, and reserving resources in the network. Finally, this chapter shows how these domain-agonistic common operations can be packaged in an in-network framework that can reduce the need for domain-specific software stacks.

8.2 Findings

This section summarizes critical recommendations learned from developing network protocols and data management frameworks for big-science communities.

- This work finds that functional commonalities exist across scientific domains. Most science communities perform a set of basic operations such as data publication, discovery, and retrieval. Identifying and implementing these functionalities at a common layer would reduce the time and effort currently required to design domain-specific solutions.
- The network is the common platform that all these communities must use. Therefore, the network is a natural place to implement most of these common functionalities. The recent trend of programmable networks is making it increasingly feasible to implement these functions in the network.
- Content-centric network paradigms such as NDN can support big-science communities better. NDN's in-network protocols can support novel features such as parallel retrieval, low-latency retrieval, and at the same time, optimize in-network resource usage via caching, and intelligent request routing.

- In an NDN world, applications should pay extra attention to how they name their data. Data naming will have a profound impact on location independence, load balancing, and the amount of in-network state.
- In NDN, names can act as a common interface across various services. Scientific communities should follow the “name-once” principle, where the community names data appropriately once and use it consistently across various services and APIs.
- Future networks (such as NDN) must expose enough information to the applications. Applications should be able to query information such as the capability of the network, available resources, usage pattern, and cache capacities to make intelligent decisions.
- NDN uses in-network strategies for providing application specific in-network capabilities. How these in-network strategies will interact is an open research question and needs to be studied carefully.

8.3 Recommendations

This thesis clearly shows that the current way of handing big-data is hardly optimal. Chapter 3 presents an example where upon encountering failures applications continues to retry retrieval in an ad-hoc manner, making the problems worse. The end-to-end network principle does not help either since applications must work *around* its limitations. The next-generation science applications and an intelligent network layer must work together to ensure proper utilization of available resources.

The exercise of applying NDN for big-science clearly shows that several *aspects* of NDN such as caching, intelligent request routing are beneficial for big-science.

Adopting these aspects in a programmable network environment can alleviate big-data problems in the near future.

Finally, this thesis also shows that NDN's caching paradigm and in-network intelligence helps big-data. However, it is hard to generalize the caching and in-network protocols for every science community. Understanding individual workflows is the key to bringing synergy between applications and in-network network protocols. For example, physics data access pattern exhibits an extremely long request tail. NDN's copy-everywhere caching does not work very well with this pattern, and novel caching mechanisms should be investigated for this workflow. Similarly, in-network protocols should also continue to evolve based on the communities' requirements.

This thesis advances the state of both networking as well as data-intensive science research. It presents a compelling use case for NDN and demonstrates the value it can provide at the network layer. Utilizing NDN for a significant class of applications and showing the benefits, the author hopes would encourage further research. The protocols and architectures developed during this work have also extensively contributed to the NDN ecosystem through code contribution, testing, and protocol development. On the other hand, this work shows that the data-intensive science community can benefit from a content-centric network - the operations supported by NDN can simplify the workflows (and dataflows) used by these communities. The lessons learned from this exercise not only benefits the next generation applications but can also be applied to current workflows immediately.

Bibliography

[1] A. Hanushevsky, Potential Data Access Architectures using xrootd. http://xrootd.org/presentations/OSGAHM_1103.
Plenary.pptx.

[2] CDN Pricing,https://www.maxcdn.com/blog/cdn-framework-step-2/.

[3] Climate Science's Globally Distributed Infrastructure. https://nci.org.au/wp-content/uploads/2016/11/Williams-Trenham-2016-AGU-Fall-Meeting-ESGF.pdf.

[4] ESNet. https://fasterdata.es.net.

[5] Globus, www.globus.org.

[6] Microsoft Azure CDN Pricing, https://azure.microsoft.com/en-us/pricing/details/cdn/.

[7] NDN Strategy Callbacks. http://ndnsim.net/2.1/fw.html.

[8] openDAP, www.opendap.org.

[9] Source code for simulation scenarios. https://github.com/susmit85/icn17-simulation-scenario.

[10] TCP Tuning at ESNet. https://fasterdata.es.net/assets/fasterdata/JT-201010.pdf.

[11] Worldwide LHC Computing Grid. http://wlcg.web.cern.ch/.

[12] 4th annual Earth System Grid Federation and Ultrascale Visualization Climate Data Analysis Tools face-to-face conference report. Technical Report LLNL-TR-666753, Lawrence Livermore National Laboratory, Livermore, CA, 2014.

[13] Academic Torrents, May 2019. [Online; accessed 11. May 2019].

[14] CMIP5 Data Search | CMIP5 | ESGF-CoG, Mar 2019. [Online; accessed 17. Mar. 2019].

[15] ESnet's Network, Software Help SLAC Researchers in Record-Setting Transfer of 1 Petabyte of Data, Jan 2019. [Online; accessed 31. Jan. 2019].

[16] Global Server Load Balancing (GSLB) for Enterprise: Part 1. Concept and Service Logic, Feb 2019. [Online; accessed 1. Feb. 2019].

[17] Home | Pacific Wave, Feb 2019. [Online; accessed 15. Feb. 2019].

[18] NDN Testbed - Named Data Networking (NDN), Apr 2019. [Online; accessed 24. Apr. 2019].

[19] NFD Developer's Guide - Named Data Networking (NDN), Feb 2019. [Online; accessed 1. Feb. 2019].

[20] Open Science Torrents - the Datahub, May 2019. [Online; accessed 11. May 2019].

[21] Overview : Main : Sequence Read Archive : NCBI/NLM/NIH, Mar 2019. [Online; accessed 12. Mar. 2019].

[22] Processing: What to record? | CERN, May 2019. [Online; accessed 10. May 2019].

[23] Whole Genome Sequencing Cost, Apr 2019. [Online; accessed 13. Apr. 2019].

[24] Ibrahim Abdullahi, Suki Arif, and Suhaidi Hassan. Survey on caching approaches in information centric networking. *Journal of Network and Computer Applications*, 56:48–59, 2015.

[25] Mohamed Abouelela and Mohamed El-Darieby. Scheduling big data applications within advance reservation framework in optical grids. *Applied Soft Computing*, 38:1049–1059, 2016.

[26] T Åkesson. The atlas experiment at the cern large hadron collider. 1999.

[27] William Allcock, John Bresnahan, Rajkumar Kettimuthu, Michael Link, Catalin Dumitrescu, Ioan Raicu, and Ian Foster. The globus striped gridftp framework and server. In *Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 54. IEEE Computer Society, 2005.

[28] Ethernet Alliance and Blaine Kohl. Ethernet jumbo frames, 2009.

[29] Venkatramani Balaji, Karl E Taylor, Martin Juckes, Bryan N Lawrence, Paul J Durack, Michael Lautenschlager, Chris Blanton, Luca Cinquini, Sébastien Denvil, Mark Elkington, et al. Requirements for a global data infrastructure in support of cmip6. *Geoscientific Model Development*, 11(9):3659–3680, 2018.

[30] Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[31] Artur Barczyk. World-wide networking for lhc data processing. In *National Fiber Optic Engineers Conference*, pages NTu1E–1. Optical Society of America, 2012.

[32] Tanya Barrett, Stephen E Wilhite, Pierre Ledoux, Carlos Evangelista, Irene F Kim, Maxim Tomashevsky, Kimberly A Marshall, Katherine H Phillippy, Patti M Sherman, Michelle Holko, et al. Ncbi geo: archive for functional genomics data sets—update. *Nucleic acids research*, 41(D1):D991–D995, 2012.

[33] LAT Bauerdick, K Bloom, B Bockelman, DC Bradley, S Dasu, I Sfiligoi, A Tadel, M Tadel, F Wuerthwein, and A Yagil. Xrootd monitoring for the cms experiment. In *Journal of Physics: Conference Series*, volume 396, page 042058. IOP Publishing, 2012.

[34] C Glenn Begley and John PA Ioannidis. Reproducibility in science: improving the standard for basic and preclinical research. *Circulation research*, 116(1):116–126, 2015.

[35] Theophilus Benson, Aditya Akella, and David A Maltz. Unraveling the complexity of network management. In *NSDI*, pages 335–348, 2009.

[36] Michael D Beynon, Tahsin Kurc, Umit Catalyurek, Chialin Chang, Alan Sussman, and Joel Saltz. Distributed processing of very large datasets with datacutter. *Parallel Computing*, 27(11):1457–1478, 2001.

[37] Ashwin R Bharambe, Cormac Herley, and Venkata N Padmanabhan. Analyzing and improving bittorrent performance. *Microsoft Research, Microsoft Corporation One Microsoft Way Redmond*, WA, 98052:2005–03, 2005.

[38] Andrew D Birrell, Roy Levin, Michael D Schroeder, and Roger M Needham. Grapevine: An exercise in distributed computing. *Communications of the* ACM, 25(4):260–274, 1982.

[39] Timm Böttger, Felix Cuadrado, Gareth Tyson, Ignacio Castro, and Steve Uhlig. Open connect everywhere: A glimpse at the internet ecosystem through the lens of the netflix cdn. ACM SIGCOMM *Computer Communication Review*, 48(1):28–34, 2018.

[40] Lee Breslau, Pei Cao, Li Fan, Graham Phillips, Scott Shenker, et al. Web caching and zipf-like distributions: Evidence and implications. In *Ieee Infocom*, volume 1, pages 126–134. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEEE), 1999.

[41] Brian Tierney and Joe Metzger, ESnet. High Performance Bulk Data Transfer. https://fasterdata.es.net/assets/fasterdata/JT-201010.pdf.

[42] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. 1998. In *Proceedings of the Seventh World Wide Web Conference*, 2017.

[43] Rene Brun and Fons Rademakers. Root—an object oriented data analysis framework. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):81–86, 1997.

[44] Jeff Burke, Paolo Gasti, Naveen Nathan, and Gene Tsudik. Securing instrumented environments over content-centric networking: the case of lighting control and ndn. In *2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 394–398. IEEE, 2013.

[45] Konrad Büssow, Claudia Quedenau, Volker Sievert, Janett Tischer, Christoph Scheich, Harald Seitz, Brigitte Hieke, Frank H Niesen, Frank Götz, Ulrich Harttig, et al. A catalog of human cdna expression clones and its application to structural genomics. *Genome biology*, 5(9):R71, 2004.

[46] Matt Calder, Ashley Flavel, Ethan Katz-Bassett, Ratul Mahajan, and Jitendra Padhye. Analyzing the performance of an anycast cdn. In *Proceedings of the 2015 Internet Measurement Conference*, pages 531–537. ACM, 2015.

[47] S Campbell, J Calderazzo, Cmmap Education Changing Climates, et al. Changing climates@ colorado state: 100 (multidisciplinary) views of climate change. In *AGU Fall Meeting Abstracts*, 2011.

[48] Neal Cardwell, Yuchung Cheng, Soheil Yeganeh, and Van Jacobson. Bbr congestion control. *Working Draft, IETF Secretariat, Internet-Draft draft-cardwell-iccrg-bbr-congestion-control-00*, 2017.

[49] Neal Cardwell, Stefan Savage, and Thomas Anderson. Modeling tcp latency. In *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No. 00CH37064)*, volume 3, pages 1742–1751. IEEE, 2000.

[50] CL Philip Chen and Chun-Yang Zhang. Data-intensive applications, challenges, techniques and technologies: A survey on big data. *Information sciences*, 275:314–347, 2014.

[51] Min Chen, Shiwen Mao, and Yunhao Liu. Big data: A survey. *Mobile networks and applications*, 19(2):171–209, 2014.

[52] Shuo Chen, Junwei Cao, and Lipeng Zhu. Ndss: A named data storage system. In *2015 International Conference on Cloud and Autonomic Computing*, pages 196–199. IEEE, 2015.

[53] Shuo Chen, Weiqi Shi, Junwei Cao, Alexander Afanasyev, and Lixia Zhang. Ndn repo: an ndn persistent storage model. 2014.

[54] Ann Chervenak, Ian Foster, Carl Kesselman, Charles Salisbury, and Steven Tuecke. The data grid: Towards an architecture for the distributed management and analysis of large scientific datasets. *Journal of network and computer applications*, 23(3):187–200, 2000.

[55] Brian Cho and Indranil Gupta. Budget-constrained bulk data transfer via internet and shipping networks. In *Proceedings of the 8th ACM international conference on Autonomic computing*, pages 71–80. ACM, 2011.

[56] Deanna M Church, Valerie A Schneider, Tina Graves, Katherine Auger, Fiona Cunningham, Nathan Bouk, Hsiu-Chuan Chen, Richa Agarwala, William M McLaren, Graham RS Ritchie, et al. Modernizing reference genome assemblies. *PLoS biology*, 9(7):e1001091, 2011.

[57] Luca Cinquini, Daniel Crichton, Chris Mattmann, John Harney, Galen Shipman, Feiyi Wang, Rachana Ananthakrishnan, Neill Miller, Sebastian Denvil, Mark Morgan, et al. The earth system grid federation: An open infrastructure for access to distributed geospatial data. *Future Generation Computer Systems*, 36:400–417, 2014.

[58] Ian Clarke, Oskar Sandberg, Brandon Wiley, and Theodore W Hong. Freenet: A distributed anonymous information storage and retrieval system. In *Designing privacy enhancing technologies*, pages 46–66. Springer, 2001.

[59] LSST Dark Energy Science Collaboration et al. Large synoptic survey telescope: dark energy science collaboration. *arXiv preprint arXiv:1211.0310*, 2012.

[60] Mauro Conti, Paolo Gasti, and Marco Teoli. A lightweight mechanism for detection of cache pollution attacks in named data networking. *Computer Networks*, 57(16):3178–3191, 2013.

[61] Peter Cornillon, James Gallagher, and Tom Sgouros. Opendap: Accessing data in a distributed, heterogeneous environment. *Data Science Journal*, 2:164–174, 2003.

[62] S Cotter. Sdn-based innovation in new zealand. In 2014 *International Science and Technology Conference (Modern Networking Technologies)(MoNeTeC)*, pages 1–2. IEEE, 2014.

[63] Ali Dabirmoghaddam, Mostafa Dehghan, and JJ Garcia-Luna-Aceves. Characterizing interest aggregation in content-centric networks. In *IFIP Networking Conference (IFIP Networking) and Workshops, 2016*, pages 449–457. IEEE, 2016.

[64] Yuri Demchenko, Paola Grosso, Cees De Laat, and Peter Membrey. Addressing big data issues in scientific data infrastructure. In 2013 *International Conference on Collaboration Technologies and Systems (CTS)*, pages 48–55. IEEE, 2013.

[65] Yuri Demchenko, Zhiming Zhao, Paola Grosso, Adianto Wibisono, and Cees De Laat. Addressing big data challenges for scientific data infrastructure. In *4th IEEE International Conference on Cloud Computing Technology and Science Proceedings*, pages 614–617. IEEE, 2012.

[66] Peter Dewdney, Peter Hall, R Schillizzi, and J Lazio. The square kilometre array. *Proceedings of the Institute of Electrical and Electronics Engineers IEEE*, 97(8):1482–1496, 2009.

[67] Shyamala Doraimani and Adriana Iamnitchi. Revisiting locality of reference in scientific grid workloads.

[68] Alvise Dorigo, Peter Elmer, Fabrizio Furano, and Andrew Hanushevsky. Xrootd-a highly scalable architecture for data access. *WSEAS Transactions on Computers*, 1(4.3):348–353, 2005.

[69] Bonnie J Dorr, Craig S Greenberg, Peter Fontana, Mark Przybocki, Marion Le Bras, Cathryn Ploehn, Oleg Aulov, Martial Michel, E Jim Golden, and Wo Chang. The nist data science initiative. In *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 1–10. IEEE, 2015.

[70] Ricky Egeland, Tony Wildish, and Simon Metson. Data transfer infrastructure for cms data taking. In *XII Advanced Computing and Analysis Techniques in Physics Research*, volume 70, page 033. SISSA Medialab, 2009.

[71] Veronika Eyring, Sandrine Bony, Gerald A Meehl, Catherine A Senior, Bjorn Stevens, Ronald J Stouffer, and Karl E Taylor. Overview of the coupled model intercomparison project phase 6 (cmip6) experimental design and organization. *Geoscientific Model Development (Online)*, 9(LLNL-JRNL-736881), 2016.

[72] Chengyu Fan, Susmit Shannigrahi, Steve DiBenedetto, Catherine Olschanowsky, Christos Papadopoulos, and Harvey Newman. Managing

scientific data with named data networking. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, page 1. ACM, 2015.

[73] Darleen Fisher. Us national science foundation and the future internet design. *ACM SIGCOMM Computer Communication Review*, 37(3):85–87, 2007.

[74] Darleen Fisher. A look behind the future internet architectures efforts. *ACM SIGCOMM Computer Communication Review*, 44(3):45–49, 2014.

[75] David Gil and Il-Yeol Song. Modeling and management of big data: challenges and opportunities, 2016.

[76] Yolanda Gil, Ewa Deelman, Mark Ellisman, Thomas Fahringer, Geoffrey Fox, Dennis Gannon, Carole Goble, Miron Livny, Luc Moreau, and Jim Myers. Examining the challenges of scientific workflows. *Computer*, 40(12):24–32, 2007.

[77] Krzysztof Ginalski, Jakub Pas, Lucjan S Wyrwicz, Marcin von Grotthuss, Janusz M Bujnicki, and Leszek Rychlewski. Orfeus: detection of distant homology using sequence profiles and predicted secondary structure. *Nucleic acids research*, 31(13):3804–3807, 2003.

[78] Matthew J Graham, S George Djorgovski, Ashish Mahabal, Ciro Donalek, Andrew Drake, and Giuseppe Longo. Data challenges of time domain astronomy. *Distributed and Parallel Databases*, 30(5-6):371–384, 2012.

[79] MA Grigorieva, EA Ryabinkin, MV Golosova, MY Gubin, VV Osipova, and AA Klimentov. Evaluating non-relational storage technology for hep metadata and meta-data catalog. In *J. Phys. Conf. Ser.*, volume 762, page 012017, 2016.

[80] Chin Guok. A user driven dynamic circuit network implementation. *Lawrence Berkeley National Laboratory*, 2009.

[81] Chin Guok, ESnet Network Engineer, and David Robertson. Esnet on-demand secure circuits and advance reservation system (oscars). *Internet2 Joint*, 92, 2006.

[82] Chin Guok, David Robertson, Mary Thompson, Jason Lee, Brian Tierney, and William Johnston. Intra and interdomain circuit provisioning using the oscars reservation system. In *Broadband Communications, Networks and Systems, 2006. BROADNETS 2006. 3rd International Conference on*, pages 1–8. IEEE, 2006.

[83] Arpit Gupta, Laurent Vanbever, Muhammad Shahbaz, Sean P Donovan, Brandon Schlinker, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, and Ethan Katz-Bassett. Sdx: A software defined internet exchange. *ACM SIGCOMM Computer Communication Review*, 44(4):551–562, 2015.

[84] Thomas J Hacker, Brian D Athey, and Brian Noble. The end-to-end performance effects of parallel tcp sockets on a lossy wide-area network. In *Proceedings 16th International Parallel and Distributed Processing Symposium*, pages 10–pp. IEEE, 2001.

[85] Mohamed Ahmed Hail, Marica Amadeo, Antonella Molinaro, and Stefan Fischer. Caching in named data networking for the wireless internet of things. In *2015 international conference on recent advances in internet of things (RIoT)*, pages 1–6. IEEE, 2015.

[86] Akram Hakiri, Pascal Berthou, Aniruddha Gokhale, and Slim Abdellatif. Publish/subscribe-enabled software defined networking for efficient and scalable iot communications. *arXiv preprint arXiv:1711.05036*, 2017.

[87] Keith Hamilton. Secure distributed publish/subscribe system, October 16 2018. US Patent App. 10/104,049.

[88] Bing Han, Xiaofei Wang, Nakjung Choi, Ted Kwon, and Yanghee Choi. Amvs-ndn: Adaptive mobile video streaming and sharing in wireless named data networking. In 2013 *IEEE Conference on Computer Communications Workshops* (INFOCOM WKSHPS), pages 375–380. IEEE, 2014.

[89] Andreas Hanemann, Jeff W Boote, Eric L Boyd, Jérôme Durand, Loukik Kudarimoti, Roman Łapacz, D Martin Swany, Szymon Trocha, and Jason Zurawski. Perfsonar: A service oriented architecture for multi-domain network monitoring. In *International conference on service-oriented computing*, pages 241–254. Springer, 2005.

[90] Jarmo Harju and Perttu Kivimaki. Co-operation and comparison of diffserv and intserv: performance measurements. In *Proceedings 25th Annual IEEE Conference on Local Computer Networks*. LCN 2000, pages 177–186. IEEE, 2000.

[91] David Harrison. bep_0000.rst_post, Mar 2019. [Online; accessed 11. May 2019].

[92] Thomas Hauser, Patrick J Burns, Thomas E Cheatham, HJ Siegel, and James Williams. Cyberinfrastructure facilities at colorado state university.

[93] Thomas Hauser, Patrick J Burns, Thomas E Cheatham, HJ Siegel, and James Williams. Rocky mountain advanced computing consortium cyberinfrastructure plan.

[94] Felix Heine, Matthias Hovestadt, and Odej Kao. Towards ontology-driven p2p grid resource discovery. In *Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 76–83. IEEE Computer Society, 2004.

[95] AKM Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. Nlsr: named-data link state routing protocol. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking*, pages 15–20. ACM, 2013.

[96] Ivy Pei-Shan Hsu, David Chun-Ying Cheung, and Rajkumar Ramniranjan Jalan. Global server load balancing, October 25 2016. US Patent 9,479,574.

[97] Bert Hubert, Thomas Graf, Greg Maxwell, Remco van Mook, Martijn van Oosterhout, P Schroeder, Jasper Spaans, and Pedro Larroy. Linux advanced routing & traffic control. In *Ottawa Linux Symposium*, volume 213, 2002.

[98] Qingmin Jia, Renchao Xie, Tao Huang, Jiang Liu, and Yunjie Liu. The collaboration for content delivery and network infrastructures: A survey. *IEEE Access*, 5:18088–18106, 2017.

[99] Xiaolong Jin, Benjamin W Wah, Xueqi Cheng, and Yuanzhuo Wang. Significance and challenges of big data research. *Big Data Research*, 2(2):59–64, 2015.

[100] Mario Jurić, Jeffrey Kantor, KT Lim, Robert H Lupton, Gregory Dubois-Felsmann, Tim Jenness, Tim S Axelrod, Jovan Aleksić, Roberta A Allsman, Yusra AlSayyad, et al. The lsst data management system. *arXiv preprint arXiv:1512.07914*, 2015.

[101] Stephen Kaisler, Frank Armour, J Alberto Espinosa, and William Money. Big data: Issues and challenges moving forward. In *2013 46th Hawaii International Conference on System Sciences*, pages 995–1004. IEEE, 2013.

[102] Ethan Katz-Bassett, Harsha V Madhyastha, Vijay Kumar Adhikari, Colin Scott, Justine Sherry, Peter Van Wesep, Thomas E Anderson, and Arvind Krishnamurthy. Reverse traceroute. In *NSDI*, volume 10, pages 219–234, 2010.

[103] JE Kay, C Deser, A Phillips, A Mai, C Hannay, G Strand, JM Arblaster, SC Bates, G Danabasoglu, J Edwards, et al. The community earth system model (cesm) large ensemble project: A community resource for studying climate change in the presence of internal climate variability. *Bulletin of the American Meteorological Society*, 96(8):1333–1349, 2015.

[104] Daehwan Kim, B Langmead, and S Salzberg. Hisat2: graph-based alignment of next-generation sequencing reads to a population of genomes, 2017.

[105] Mariam Kiran, Peter Murphy, Inder Monga, Jon Dugan, and Sartaj Singh Baveja. Lambda architecture for cost-effective batch and speed big data processing. In *2015 IEEE International Conference on Big Data (Big Data)*, pages 2785–2792. IEEE, 2015.

[106] Tevfik Kosar, Engin Arslan, Brandon Ross, and Bing Zhang. Storkcloud: Data transfer scheduling and optimization as a service. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 29–36. ACM, 2013.

[107] Diego Kreutz, Fernando MV Ramos, Paulo Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[108] Kevin Lahey. Tcp problems with path mtu discovery. Technical report, 2000.

[109] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–565, 1978.

[110] Yun-Young Lee and Richard Grotjahn. Evidence of specific mjo phase occurrence with summertime california central valley extreme hot weather. *Advances in Atmospheric Sciences*, 36(6):589–602, Jun 2019.

[111] Hui Li. Workload dynamics on clusters and grids. *The Journal of Supercomputing*, 47(1):1–20, 2009.

[112] Jun Li, Hao Wu, Bin Liu, Jianyuan Lu, Yi Wang, Xin Wang, YanYong Zhang, and Lijun Dong. Popularity-driven coordinated caching in named data networking. In *Proceedings of the eighth ACM/IEEE symposium on Architectures for networking and communications systems*, pages 15–26. ACM, 2012.

[113] Huhnkuk Lim, Alexander Ni, Dabin Kim, Young-Bae Ko, Susmit Shannigrahi, and Christos Papadopoulos. Ndn construction for big science: Lessons learned from establishing a testbed. *IEEE Network*, 32(6):124–136, 2018.

[114] Jared Lindblom, M Huang, Jeff Burke, and Lixia Zhang. Filesync/ndn: Peer-to-peer file sync over named data networking. *NDN Technical Report NDN-0012*, 2013.

[115] Qiming Lu, Liang Zhang, Sajith Sasidharan, Wenji Wu, Phil DeMar, Chin Guok, John Macauley, Inder Monga, Se-young Yu, Jim Hao Chen, et al. Bigdata express: Toward schedulable, predictable, and high-performance data transfer. In *2018 IEEE/ACM Innovating the Network for Data-Intensive Science (INDIS)*, pages 75–84. IEEE, 2018.

[116] Eng Keong Lua, Jon Crowcroft, Marcelo Pias, Ravi Sharma, Steven Lim, et al. A survey and comparison of peer-to-peer overlay network schemes.

[117] Jake Luo, Min Wu, Deepika Gopukumar, and Yiqing Zhao. Big data application in biomedical research and health care: a literature review. *Biomedical informatics insights*, 8:BII–S31559, 2016.

[118] Gary Scott Malkin. Traceroute using an ip option. 1993.

[119] Edoardo Martelli and S Stancu. Lhcopn and lhcone: status and future evolution. In *Journal of Physics: Conference Series*, volume 664, page 052025. IOP Publishing, 2015.

[120] Vivien Marx. Biology: The big challenges of big data, 2013.

[121] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnsim 2.0: A new version of the ndn simulator for ns-3. *NDN, Technical Report NDN-0028*, 2015.

[122] Spyridon Mastorakis, Alexander Afanasyev, Yingdi Yu, and Lixia Zhang. ntorrent: Peer-to-peer file sharing in named data networking. In *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, pages 1–10. IEEE, 2017.

[123] LLC MaxMind. Geoip, 2006.

[124] Péter Megyesi, Zsolt Krämer, and Sándor Molnár. How quick is quic? In *2016 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2016.

[125] Inder Monga, Eric Pouyoul, and Chin Guok. Software-defined networking for big-data science-architectural models from campus to the wan. In *2012 SC Companion: High Performance Computing, Networking Storage and Analysis*, pages 1629–1635. IEEE, 2012.

[126] morrison. BiSON Network, Mar 2005. [Online; accessed 14. Feb. 2019].

[127] Richard P Mount et al. The office of science data-management challenge. Technical report, Stanford Linear Accelerator Center (SLAC), 2005.

[128] Marcus R Munafò, Brian A Nosek, Dorothy VM Bishop, Katherine S Button, Christopher D Chambers, Nathalie Percie Du Sert, Uri Simonsohn, Eric-Jan Wagenmakers, Jennifer J Ware, and John PA Ioannidis. A manifesto for reproducible science. *Nature human behaviour*, 1(1):0021, 2017.

[129] Stefano Nativi, Max Craglia, and Jay Pearlman. Earth science infrastructures interoperability: the brokering approach. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 6(3):1118–1129, 2013.

[130] Stefano Nativi, Paolo Mazzetti, Mattia Santoro, Fabrizio Papeschi, Max Craglia, and Osamu Ochiai. Big data challenges in building the global earth observation system of systems. *Environmental Modelling & Software*, 68:1–26, 2015.

[131] Harvey Newman, Richard Cavanaugh, Julian James Bunn, Iosif Legrand, Steven H Low, Dan Nae, Sylvain Ravot, Conrad D Steenberg, Xun Su, Michael Thomas, et al. The ultralight project: The network as an integrated and managed resource for data-intensive. *Computing in science & engineering*, 7(6):38, 2005.

[132] Harvey Newman, Azher Mughal, Dorian Kcira, Iosif Legrand, Ramiro Voicu, and Julian Bunn. High speed scientific data transfers using software defined networking. In *Proceedings of the Second Workshop on Innovating the Network for Data-Intensive Science*, page 2. ACM, 2015.

[133] Catherine Olschanowsky, Susmit Shannigrahi, and Christos Papadopoulos. Supporting climate research using named data networking. In *2014 IEEE 20th International Workshop on Local & Metropolitan Area Networks* (LANMAN), pages 1–6. IEEE, 2014.

[134] Anne-Cécile Orgerie, Laurent Lefèvre, and Isabelle Guérin-Lassous. Energy-efficient bandwidth reservation for bulk data transfers in dedicated wired networks. *The Journal of Supercomputing*, 62(3):1139–1166, 2012.

[135] Jonathan T Overpeck, Gerald A Meehl, Sandrine Bony, and David R Easterling. Climate data challenges in the 21st century. *science*, 331(6018):700–702, 2011.

[136] H Cenk Ozmutlu, Amanda Spink, and Seda Ozmutlu. Analysis of large data logs: an application of poisson sampling on excite web queries. *Information processing & management*, 38(4):473–490, 2002.

[137] Robert B O'hara and D Johan Kotze. Do not log-transform count data. *Methods in Ecology and Evolution*, 1(2):118–122, 2010.

[138] Ping P Pan, Ellen L Hahne, and Henning Schulzrinne. Bgrp: Sink-tree-based aggregation for inter-domain reservations. *Journal of Communications and Networks*, 2(2):157–167, 2000.

[139] David J Patterson, J Cooper, Paul M Kirk, RL Pyle, and David P Remsen. Names are key to the big new biology. *Trends in ecology & evolution*, 25(12):686–691, 2010.

[140] Qiuyu Peng, Anwar Walid, Jaehyun Hwang, and Steven H Low. Multipath tcp: Analysis, design, and implementation. *IEEE/ACM Transactions on networking*, 24(1):596–609, 2016.

[141] Andreas J Peters and Lukasz Janyst. Exabyte scale storage at cern. In *Journal of Physics: Conference Series*, volume 331, page 052015. IOP Publishing, 2011.

[142] Costin Raiciu, Sebastien Barre, Christopher Pluntke, Adam Greenhalgh, Damon Wischik, and Mark Handley. Improving datacenter performance and robustness with multipath tcp. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 266–277. ACM, 2011.

[143] Kannan Rajah, Sanjay Ranka, and Ye Xia. Advance reservations and scheduling for bulk transfers in research networks. *IEEE Transactions on Parallel and Distributed Systems*, 20(11):1682–1697, 2009.

[144] Arcot Rajasekar, Reagan Moore, Chien-yi Hou, Christopher A Lee, Richard Marciano, Antoine de Torcy, Michael Wan, Wayne Schroeder, Sheau-Yen Chen, Lucas Gilbert, et al. irods primer: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 2(1):1–143, 2010.

[145] Arcot K Rajasekar and Reagan W Moore. Data and metadata collections for scientific applications. In *International Conference on High-Performance Computing and Networking*, pages 72–80. Springer, 2001.

[146] J Rehn, T Barrass, D Bonacorsi, J Hernandez, I Semeniouk, L Tuura, and Y Wu. Phedex high-throughput data transfer management system. In *Computing in High Energy and Nuclear Physics (CHEP)*, volume 2006, 2006.

[147] Yongmao Ren, Jun Li, Shanshan Shi, Lingling Li, Guodong Wang, and Beichuan Zhang. Congestion control in named data networking–a survey. *Computer Communications*, 86:1–11, 2016.

[148] Russ Rew and Glenn Davis. Netcdf: an interface for scientific data access. *IEEE computer graphics and applications*, 10(4):76–82, 1990.

[149] Injong Rhee, Lisong Xu, Sangtae Ha, Alexander Zimmermann, Lars Eggert, and Richard Scheffenegger. Cubic for fast long-distance networks. Technical report, 2018.

[150] David R Richardson, John Cormie, Imran S Patel, Benjamin WS Redman, and Richard Sheehan. Request routing utilizing client location information, September 27 2011. US Patent 8,028,090.

[151] Matei Ripeanu, Adriana Iamnitchi, and Ian Foster. Mapping the gnutella network. *IEEE Internet Computing*, (1):50–57, 2002.

[152] Charles P Rodi, Roderick T Bunch, Sandra W Curtiss, Larry D Kier, Marc A Cabonce, Julio C Davila, Michael D Mitchell, Carl L Alden, and Dale L Morris. Revolution through genomics in investigative and discovery toxicology. *Toxicologic pathology*, 27(1):107–110, 1999.

[153] Antony Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *IFIP/ACM International Conference on Distributed Systems Platforms and Open Distributed Processing*, pages 329–350. Springer, 2001.

[154] Seref Sagiroglu and Duygu Sinanc. Big data: A review. In 2013 *International Conference on Collaboration Technologies and Systems* (CTS), pages 42–47. IEEE, 2013.

[155] Jagruti Sahoo, Mohammad A Salahuddin, Roch Glitho, Halima Elbiaze, and Wessam Ajib. A survey on replica server placement algorithms for content delivery networks. *IEEE Communications Surveys & Tutorials*, 19(2):1002–1026, 2017.

[156] Bertil Schmidt and Andreas Hildebrandt. Next-generation sequencing: big data meets high performance computing. *Drug discovery today*, 22(4):712–717, 2017.

[157] Klaus Schneider, Cheng Yi, Beichuan Zhang, and Lixia Zhang. A practical congestion control scheme for named data networking. In *Proceedings of the 2016 conference on 3rd ACM Conference on Information-Centric Networking*, pages 21–30. ACM, 2016.

[158] Wentao Shang, Alexander Afanasyev, and Lixia Zhang. Vectorsync: distributed dataset synchronization over named data networking. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, pages 192–193. ACM, 2017.

[159] Wentao Shang, Zhe Wen, Qiuhan Ding, Alexander Afanasyev, and Lixia Zhang. Ndnfs: An ndn-friendly file system. *NDN Technical Report NDN-0027, Revision 1*, 2014.

[160] S. Shannigrahi, C. Fan, and G. White. Bridging the icn deployment gap with ipoc: An ip-over-icn protocol for 5g networks. In *ACM SIGCOMM 2018 Workshop on Networking for Emerging Applications and Technologies*(NEAT). ACM, 2018.

[161] 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*, pages 54–65. ACM, 2017.

[162] 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*, pages 54–65. ACM, 2017.

[163] Susmit Shannigrahi, Chengyu Fan, and Christos Papadopoulos. Named data networking strategies for improving large scientific data transfers. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops): Information Centric Networking Solutions for Real World Applications (ICN-SRA) (ICC 2018 Workshop - ICN-SRA)*, Kansas City, USA, May 2018.

[164] Susmit Shannigrahi, Chengyu Fan, and Christos Papadopoulos. Named data networking strategies for improving large scientific data transfers. In *2018 IEEE International Conference on Communications Workshops (ICC Workshops): Information Centric Networking Solutions for Real World Applications (ICN-SRA) (ICC 2018 Workshop - ICN-SRA)*. IEEE, 2018.

[165] Susmit Shannigrahi, Chengyu Fan, and Christos Papadopoulos. Scari: A strategic caching and reservation protocol for icn. In *Proceedings of the Asian Internet Engineering Conference*, pages 1–8. ACM, 2018.

[166] Susmit Shannigrahi, Christos Papadopoulos, Edmund Yeh, Harvey Newman, Artur Jerzy Barczyk, Ran Liu, Alex Sim, Azher Mughal, Inder Monga, Jean-Roch Vlimant, et al. Named data networking in climate research and hep applications. In *Journal of Physics: Conference Series*, volume 664, page 052033. IOP Publishing, 2015.

[167] Arie Shoshani and Doron Rotem. Scientific data management. challenges, technology, and development. *Scientific Data Management: Challenges, Technology, and Deployment*, 01 2009.

[168] Michael Slocombe, Matthew Miller, Casey Ajalat, and Vincent A Fuller III. Content request routing and load balancing for content distribution networks, June 8 2017. US Patent App. 15/433,942.

[169] Robin Snader and Nikita Borisov. A tune-up for tor: Improving security and performance in the tor network. In *ndss*, volume 8, page 127, 2008.

[170] Won So, Ashok Narayanan, David Oran, and Mark Stapp. Named data networking on a router: forwarding at 20gbps and beyond. In *ACM SIGCOMM Computer Communication Review*, volume 43, pages 495–496. ACM, 2013.

[171] Daniele Spiga, Stefano Lacaprara, W Bacchi, Mattia Cinquilli, Giuseppe Codispoti, Marco Corvo, A Dorigo, Alessandra Fanfani, Federica Fanzago, Fabio Farina, et al. The cms remote analysis builder (crab). In *International Conference on High-Performance Computing*, pages 580–586. Springer, Berlin, Heidelberg, 2007.

[172] Zachary D Stephens, Skylar Y Lee, Faraz Faghri, Roy H Campbell, Chengxiang Zhai, Miles J Efron, Ravishankar Iyer, Michael C Schatz, Saurabh Sinha, and Gene E Robinson. Big data: astronomical or genomical? *PLoS biology*, 13(7):e1002195, 2015.

[173] Ion Stoica, Robert Morris, David Karger, M Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM Computer Communication Review*, 31(4):149–160, 2001.

[174] Gary Strand. Community earth system model data management: Policies and challenges. *Procedia Computer Science*, 4:558–566, 2011.

[175] Rick Summerhill. The new internet2 network. In *6th GLIF Meeting*, 2006.

[176] Alexander Szalay and Jim Gray. 2020 computing: Science in an exponential world. *Nature*, 440(7083):413, 2006.

[177] Tarik Taleb, Konstantinos Samdanis, Badr Mada, Hannu Flinck, Sunny Dutta, and Dario Sabella. On multi-access edge computing: A survey of the emerging 5g network edge cloud architecture and orchestration. *IEEE Communications Surveys & Tutorials*, 19(3):1657–1681, 2017.

[178] Karl E Taylor, V Balaji, Steve Hankin, Martin Juckes, Bryan Lawrence, and Stephen Pascoe. Cmip5 data reference syntax (drs) and controlled vocabularies. In *PCMDI*, 2011.

[179] Karl E Taylor, Ronald J Stouffer, and Gerald A Meehl. An overview of cmip5 and the experiment design. *Bulletin of the American Meteorological Society*, 93(4):485–498, 2012.

[180] Brian Tierney, Ezra Kissel, Martin Swany, and Eric Pouyoul. Efficient data transfer protocols for big data. In *2012 IEEE 8th International Conference on E-Science*, pages 1–9. IEEE, 2012.

[181] Christian Tschudin and Manolis Sifalakis. Named function networking.

[182] Satoshi Tsuchiya, Yoshinori Sakamoto, Yuichi Tsuchimoto, and Vivian Lee. Big data processing in cloud environments. *Fujitsu Sci. Tech. J*, 48(2):159–168, 2012.

[183] Prem Uppuluri, Narendranadh Jabisetti, Uday Joshi, and Yugyung Lee. P2p grid: service oriented framework for distributed resource management. In *2005 IEEE International Conference on Services Computing (SCC'05) Vol-1*, volume 1, pages 347–350. IEEE, 2005.

[184] Marco Viceconti, Peter Hunter, and Rod Hose. Big data, big knowledge: big data for personalized healthcare. *IEEE journal of biomedical and health informatics*, 19(4):1209–1215, 2015.

[185] Sebastian Wandelt, Astrid Rheinländer, Marc Bux, Lisa Thalheim, Berit Haldemann, and Ulf Leser. Data management challenges in next generation sequencing. *Datenbank-Spektrum*, 12(3):161–171, 2012.

[186] Meisong Wang, Prem Prakash Jayaraman, Rajiv Ranjan, Karan Mitra, Miranda Zhang, Eddie Li, Samee Khan, Mukkaddim Pathan, and Dimitrios Georgeakopoulos. An overview of cloud based content delivery networks: research dimensions and state-of-the-art. In *Transactions on Large-Scale Data- and Knowledge-Centered Systems* XX, pages 131–158. Springer, 2015.

[187] Joel M Wein, John Josef Kloninger, Mark C Nottingham, David R Karger, and Philip A Lisiecki. Content delivery network (cdn) content server request handling mechanism with metadata framework support, April 19 2018. US Patent App. 15/846,526.

[188] DN Williams. 2015 esgf progress report. Technical report, Lawrence Livermore National Lab.(LLNL), Livermore, CA (United States), 2015.

[189] Hao Xu, Terrell Russell, Jason Coposky, Arcot Rajasekar, Reagan Moore, Antoine de Torcy, Michael Wan, Wayne Shroeder, and Sheau-Yen Chen. irods primer 2: integrated rule-oriented data system. *Synthesis Lectures on Information Concepts, Retrieval, and Services*, 9(3):1–131, 2017.

[190] George Xylomenos, Christopher N Ververidis, Vasilios A Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V Katsaros, and George C Polyzos. A survey of information-centric networking research. *IEEE Communications Surveys & Tutorials*, 16(2):1024–1049, 2014.

[191] Chao-Tung Yang, Sung-Yi Chen, and Tsui-Ting Chen. A grid resource broker with network bandwidth-aware job scheduling for computational grids. In *In-*

ternational Conference on Grid and Pervasive Computing, pages 1–12. Springer, 2007.

[192] Shui Yu, Meng Liu, Wanchun Dou, Xiting Liu, and Sanming Zhou. Networking for big data: A survey. *IEEE Communications Surveys & Tutorials*, 19(1):531–549, 2017.

[193] 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)*, pages 1–9. IEEE, 2012.

[194] Roberta H Yuhas, Alexander FH Goetz, and Joe W Boardman. Discrimination among semi-arid landscape endmembers using the spectral angle mapper (sam) algorithm. 1992.

[195] Haitao Zhang, Zhehao Wang, Christopher Scherb, Claudio Marxer, Jeff Burke, Lixia Zhang, and Christian Tschudin. Sharing mhealth data via named data networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 142–147. ACM, 2016.

[196] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named data networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[197] Lixia Zhang, Stephen Deering, Deborah Estrin, Scott Shenker, and Daniel Zappala. Rsvp: A new resource reservation protocol. *Network, IEEE*, 7(5):8–18, 1993.

[198] 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*, 157:158, 2010.

[199] Meng Zhang, Hongbin Luo, and Hongke Zhang. A survey of caching mechanisms in information-centric networking. *IEEE Communications Surveys & Tutorials*, 17(3):1473–1499, 2015.

[200] Minsheng Zhang, Vince Lehman, and Lan Wang. Partialsync: Efficient synchronization of a partial namespace in ndn. *Technical report, Technical Report NDN-0039, NDN*, 2016.

[201] Yanxia Zhang and Yongheng Zhao. Astronomy in the big data era. *Data Science Journal*, 14, 2015.

[202] Ben Y Zhao, Ling Huang, Jeremy Stribling, Sean C Rhea, Anthony D Joseph, and John D Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on selected areas in communications*, 22(1):41–53, 2004.

[203] Zhenkai Zhu and Alexander Afanasyev. Let's chronosync: Decentralized dataset state synchronization in named data networking. In *Network Protocols (ICNP), 2013 21st IEEE International Conference on*, pages 1–10. IEEE, 2013.

[204] Liudong Zuo, Michelle Mengxia Zhu, and Chase Qishi Wu. Fast and efficient bandwidth reservation algorithms for dynamic network provisioning. *Journal of Network and Systems Management*, 23(3):420–444, 2015.

Glossary

Anycast Network addressing scheme where multiple hosts have single IP address. 27

Big Data Extremely Large Datasets; in scientific context, ranges from hundreds of Terabytes to Exabytes. 4, 9, 91

BitTorrent A peer-to-peer file sharing protocol. 30, 31, 54, 137

CDN A Content Delivery Network. vi, 84, 85, 163, 167

CENIC The Corporation for Education Network Initiatives in California. 94

Chronosync A protocol for synchronizing datasets among multiple parties in NDN. 137–139, 141

CLUI Command Line User Interface. 129

CS NFD Content Store; a data structure in Named Data Networking. 54, 55, 115, 176

CSU Colorado State University. 93

Data Management Tools and algorithms for managing datasets that are often very large and dispersed. 8

DKRZ The German Climate Computing Centre. 59

DNS Domain Name System. 26–28, 48, 49, 105, 108

DOE Department of Energy. 36

DRS Data reference syntax; a naming guide used in ESGF. 37, 106

ERM End node reservation manager. 171–173, 179

ESGF The Earth System Grid Federation; a distributed system for sharing climate data. v, vi, viii–x, 3, 24, 28, 36–41, 43, 57–60, 62, 70, 76, 84, 87, 129, 155, 157, 230

ESnet The Energy Sciences Network; high-speed computer network serving United States Department of Energy scientists and their collaborators. 7, 21, 94, 95, 99, 153, 165–167, 170

FIA NSF Future Internet Architecture Project. 45

FIB Forwarding Information Base; a data structure in Named Data Networking. 47, 50, 53, 105, 185

FRGP Front Range GigaPop; a consortium that cooperate to share wide area networking service. 94

FTP File Transfer Protocol. x, 12, 21, 155, 157

Globus A high-performance data transfer application. 39

GridFTP GridFTP is an extension of the File Transfer Protocol for grid computing. 37, 192, 193, 195

GSLB Global server load balancing; intelligent distribution of traffic across servers. 26, 27

HEP High Energy Particle Physics. ii, 1, 7, 9, 12, 24, 35, 36, 57, 79, 92, 103, 110, 111, 121, 152, 153, 188

HTTP The Hypertext Transfer Protocol. x, 12, 18, 20, 23, 26, 27, 39, 59–61, 155, 157

ICN Information Centric Network. 6, 45, 88

IP Internet Protocol. 19–24, 26–30, 32, 34–36, 40, 41, 44, 46, 47, 51, 53, 55, 58–61, 93, 95, 99, 153, 154, 166, 167, 170, 176, 179–181, 183, 185, 190, 197, 199

iRODS A Genomics Data Management Software. ix, 24, 41–44, 112

LFU Least Frequently Used Policy; A caching policy that evicts the least frequently used entry. 157

LHC The Large Hadron Collider; world's largest and most powerful particle collider. 3, 5, 21, 57, 165, 167, 174

LHCOPN LHC Optical Private Network. 21, 165

LLNL Lawrence Livermore National Laboratory. 59, 63, 70, 73, 85, 161

LSST The Large Synoptic Survey Telescope; a wide-field survey reflecting telescope under construction. 3, 16, 57

NCBI The National Center for Biotechnology Information. 112

NDN Named Data Networking; A future Internet architecture. ii, iii, v–xi, 1, 6–8, 24, 30, 31, 44–55, 57, 58, 61, 70, 72, 74–76, 78, 79, 81, 84, 85, 87–93, 95–102, 104–113, 115–117, 119–124, 126, 127, 130–133, 135, 137, 138, 143–146, 149–155, 157–161, 163, 164, 166–171, 173–176, 179–183, 185, 188–195, 198–202, 204

NDN-SCI NDN-based scientific data management framework. vi, 7, 8, 92, 144, 145, 151

NDNSim NDN simulator for NS3. 70, 72, 73, 91, 151

NFD Named Data Networking Forwarding Daemon. x, xi, 90, 95, 97–99, 133, 175, 185, 187

NSG Next Generation Genome Sequencing. 2

OpenDAP OPeNDAP is a framework that aims to simplifie all aspects of scientific data networking. 37, 192

OpenID OpenID is an open standard and decentralized authentication protocol. 37, 38, 40, 60

OSCARS ESnet's On-Demand Secure Circuits and Advance Reservation System; an advanced software system for booking time and resources on high-speed science networks. x, 7, 94, 99, 163–166, 170, 185

P2P Peer-to-Peer. 29, 30, 36, 37

PB Petabyte. 9, 15, 16, 59, 165

PIT Pending Interest Table; a data structure in Named Data Networking. 48, 74, 76, 78, 173

PKI A public key infrastructure creates,manages, and distributes digital certificates and manage public-key encryption. 38, 135

QoS Quality of service. 21

REANNZ The Kiwi Advanced Research and Education Network. 94

RM Reservation Manager. 171, 173

RRM Router Reservation Manager. 171–173, 179

RSVP Resource Reservation Protocol. 22, 166, 170

SDN Software Defined Networking. 7, 23, 100, 154, 163, 185

SKA The Square Kilometre Array; a radio telescope project proposed to be built in Australia and South Africa. 57

strategy NDN in-network intelligent mechanisms for steering traffic. 25, 54, 84, 85, 88, 138, 144, 199, 202

TB Terabyte. 15, 16, 28, 67, 79, 92, 93, 171, 176

TCP Transmission Control Protocol. viii, x, 21–23, 29, 30, 32–36, 84, 95–97, 99, 154, 161, 179

UDP User Datagram Protocol. 21, 95, 96, 99

UI User Interface. x, 111, 124, 129, 147, 148

VLAN virtual LAN. 94, 99, 185

wget GNU Wget is an application that downloads web content. 37, 39, 40, 84

Xrootd A hierarchical storage system for Physics Data. v, ix, 31–35