EPJ Web of Conferences **245**, 04018 (2020) https://doi.org/10.1051/epjconf/202024504018
CHEP 2019

Named Data Networking based File Access for XRootD

Cătălin Iordache[1,*], *Ran* Liu[2], *Justas* Balcas[1,**], *Raimondas* Širvinskas, *Yuanhao* Wu[2], *Chengyu* Fan[3], *Susmit* Shannigrahi[4], *Harvey* Newman[1], and *Edmund* Yeh[2]

[1]California Institute of Technology, 1200 E California Blvd, Pasadena, CA 91125, United States
[2]Northeastern University, 360 Huntington Ave, Boston, MA 02115, United States
[3]Colorado State University, Fort Collins, CO 80523, United States
[4]Tennessee Tech University, 1 William L Jones Dr, Cookeville, TN 38505, United States

Abstract. We present the design and implementation of a Named Data Networking (NDN) based Open Storage System plug-in for XRootD. This is an important step towards integrating NDN, a leading future internet architecture, with the existing data management systems in CMS. This work outlines the first results of data transfer tests using internal as well as external 100 Gbps testbeds, and compares the NDN-based implementation with existing solutions.

1 Introduction

This paper describes a significant effort in the SDN-Assisted NDN for Data Intensive Experiments (SANDIE) [1] project, that brings together researchers from Caltech, Northeastern University, Colorado State University, Tennessee Tech and other institutes working towards integrating a Named Data Networking (NDN)-based data distribution system for the CMS experiment at the LHC [2]. The team has created NDN-based data naming schemes, and an NDN-based data access architecture that is supported by advanced Software Defined Network (SDN) services to meet the challenges faced by the LHC high energy physics program as well as other data intensive science use cases such as Genomics and Astrophysics. SANDIE also integrates NDN's Virtual Interest Packet (VIP) [3] paradigm, a joint forwarding and caching set of algorithms, which is applied to a set of the most frequently requested data files in CMS in order to achieve a desirable trade-off between performance and operating complexity.

While the SANDIE project aims to integrate NDN with the mainstream data distribution systems of the LHC experiments, starting with CMS, the focus of this paper is to discuss the development and deployment of a major subcomponent, an NDN-based XRootD Open Storage System (OSS) [4][5][6] plug-in instrumented with an accelerated packet forwarder.

The paper is structured as follows. In Section 2, we provide a very brief overview of NDN. In Section 3, we discuss the NDN-based XRootD OSS plug-in design and implementation details. Specifically, we present the producer-consumer applications and their integration with XRootD in the context of CMS workflow, and then discuss our experience in deploying and performance testing the plug-in on a local testbed. Section 4 demonstrates the performance analysis of our initial testbed deployment, and Section 5 shows the increasing maturity of NDN, together with the performance gains achieved by using the NDN-DPDK forwarder

*e-mail: catalinn.iordache@gmail.com
**e-mail: justas.balcas@cern.ch

© The Authors, published by EDP Sciences. This is an open access article distributed under the terms of the Creative Commons Attribution License 4.0 (http://creativecommons.org/licenses/by/4.0/).

developed by NIST. We finally discuss our plans for continued development and integration efforts and conclusions.

2 Named Data Networking

Named Data Networking (NDN) [7][8] is a leading Future Internet Architecture where users and applications can fetch data directly using the content name instead of connecting to the host where the data resides. NDN forwards packets using the content names directly, which greatly simplifies the routing and forwarding infrastructure. Additionally, NDN supports in-network caching, native multicast, multipath-forwarding, and *a strategy layer* where users can define (dynamic) per-prefix request forwarding policies. All these properties allow users to utilize resources intelligently, e.g., through the use of caching for popular datasets, utilizing multiple paths for superior latency and failover performance. NDN Data packets carry publicly verifiable built-in signatures that guarantee data provenance and integrity. This property decouples data security from its original publisher or communication channels, enabling data retrieval from any repository or router cache, as well as the original publisher. These properties also allow NDN to fully exploit rapidly declining storage costs by making caching an integral part of the network architecture, which also enhances network performance by decreasing congestion and delays. The NDN architecture has been the subject of active research around the world over the last ten years [9][10], and versions of the architecture have been experimentally implemented in several international testbeds [8].

3 The XRootD NDN based Open Storage System plug-in

To integrate an end-to-end solution for data distribution using NDN as the underlying architecture, we implemented NDN-based consumer and producer applications, the two fundamental entities in any data transfer process. The consumer acts as a client that composes NDN Interest packets (analogous to a query) to request data from the network. The producer is the server that has access to data on file systems and publishes it in the form of NDN Data packets (analogous to a response).

3.1 The consumer application

The architecture of the consumer application takes into consideration the integration with the XRootD framework, and its capability to interpret file system calls includes requests for byte-ranges within the events in HEP data files. Currently, both the consumer and producer support only the file system calls needed for copying files, namely: *open*, *fstat*, *read* and *close*. The *open* file system call provides access to files in the file system, *fstat* returns information about files such as name or size, while *read* is able to retrieve bytes from files. The consumer translates file system calls on file paths to Interest Names following the naming format "/ndn/xrootd/file_system_call/filename/segment_no?<additional_info>". All the Interest packets expressed by the consumer are under the same prefix - "/ndn/xrootd". The rest of the Name components [11], except the last one in case of *read* file system call, represent strings delimited by a "/" character in the file path.

An Interest packet that addresses the *open* file system call on file: "/path/to/foo.txt" has the NDN Name: "/ndn/xrootd/open/path/to/foo.txt<additional_info>". The consumer expects a Data packet that contains the encoded NDN Name, which is the same as the Interest that addresses it, the Content, and additional information such as the signature [12], all of which adds up to a maximum packet size of 8800 Bytes, where the Data packet protocol

between consumer and producer can support a maximum Content size of 7168 Bytes. This means that when expecting to request or publish content larger than the maximum size that can be accommodated within a single packet, one has to keep protocol consistency between the consumer and the producer. In the case of the *read* file system call, the SANDIE consumer accomplishes this by adding another Name component that represents a segment number. For example, the Interest: "/ndn/xrootd/read/path/to/foo.txt/3<additional_info>" requests the third segment in the file and expects a Data packet with bytes in the offset range [21504, 28671) from "/path/to/foo.txt".

The consumer offers multi-threaded support, and the number of concurrent threads to copy a file over the network is configurable from the command line [6]. Each thread reads a chunk of 256 KB (also configurable) from an offset, by composing all Interest packets needed and placing them in a thread-safe fixed window size pipeline. The size of the pipeline is also configurable and dictates how many Interests are expressed at any one time. For all expressed Interests in the pipeline, NACK, duplicate, congestion or timeout cases are handled with retransmission.

On receiving the Data packets, the consumer validates their signature, retrieves the Content, and passes it to a higher level of the application. When all the Interests in the pipeline are satisfied, the execution continues until the desired data from the entire file is retrieved. When a processing thread receives all Data packets for its Interests, it saves the content to a thread-safe heap object, which will be used in the final step of the application to store it on disk (i.e., in case of network performance analysis one may skip this step to save time and prolong disk life).

3.2 The producer application

The producer application runs as a *systemd* service on servers that have access to CMS data. Its primary purpose is to respond to incoming Interest packets from the forwarders in the network, interpret them and respond with Data packets. To keep track of all opened files and not run into problems that may occur in Linux based operating systems, the producer uses a thread-safe map object of all file descriptors (abstract indicators used for accessing files). Upon receiving an Interest packet, the first step of execution is to identify the file system call and the file path. Once these are determined, it checks if the file is already opened by consulting the map of file descriptors and takes action accordingly.

If the Producer has access to the file, it will perform the desired system call on it, compose a Data packet and sign it. For example, in the case of an *open* file system call, the Content of the packet is an integer, which can be zero in case of success or an error number (linux *errno*) on failure. If the call fails, the Data packet is marked as an Application Level NACK [12], so that the consumer can identify it as an error. For the *fstat* file system call the Content is the actual POSIX

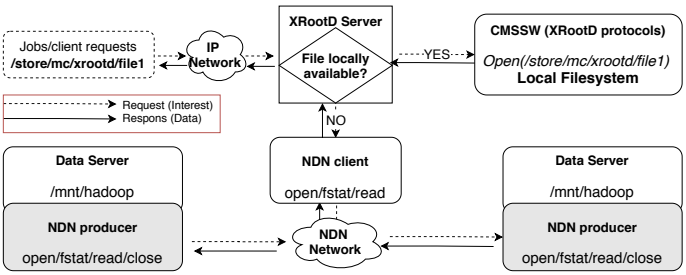

Figure 1: XRootD NDN based Open Storage System plug-in architecture

struct stat of 144 Bytes. On *read*, the producer uses the segment number in the Interest Name to locate the offset in the file and then read the maximum Content size.

The map of file descriptors is periodically checked (the frequency is configurable) and files that have not been accessed for a long time are closed. The access time is updated at each Interest packet and checked asynchronously. Other configuration parameters of the producer application are - the number of threads to be used for packet processing, enable or disable signing (i.e., use a fake signature), or the freshness period of Data packets [6].

3.3 Integration with CMS and deliverables

To have a transparent transition to the end-user from the current CMS architecture for distributing data to an NDN-based one, we perform this integration through an Open Storage System plug-in for the XRootD framework. The OSS plug-in provides the particular implementation of a logical file system. Here, logical file system operations are translated into specific actions optimized for the underlying storage. The plug-in is developed following a C++ programming language interface proposed by XRootD, where each function mirrors one of the POSIX file system calls. It represents a shared library that exports the *XrdOssGetStorageSystem* symbol used by the framework to load it. The NDN based XRootd plug-in, and its flow in an NDN network with multiple producers working with the CMS software base (CMSSW), is presented in Figure 1.

The default operating system used in CMS is CentOS 7. For better integration with the system, we packaged our software as RPMs and published them on a proprietary repository developed by SANDIE, where the entire software stack is kept updated. We consider the NDN software stack to be composed of: ndn-cxx [13], the NDN Forwarding Daemon (NFD) [14], ndn-tools [15], Boost C++ libraries[1] and XRootD. The repository is available at [16] and is currently used by the NDN team for their continuous integration system and puppet[2] scripts for configuring the SANDIE testbed.

4 Performance Analysis

The consumer and producer were built using the ndn-cxx API [13], which is the foundation for the NFD [14] forwarder as well as the applications that depend on it. NFD creates in-network caches and forwarding services in the network that the applications can utilize. Both ndn-cxx and NFD are research prototypes that bring a considerable performance overhead and a low throughput. The current OSS plug-in utilizes both, and as a result, suffers from low performance. More recently, NIST has created a high-performance NDN-DPDK [17] forwarder that provides much-improved application performance, and the SANDIE team is currently working on integrating it into the XRootD plug-in.

The performance testing of the consumer and producer was done on the local SANDIE testbed at Caltech. Three dual processor Intel Xeon E5 and one AMD EPYC uniprocessor servers, with 32 compute threads per processor and more than 100 GB of RAM were used. The forwarders were configured with specific routing rules and without caching policy. The throughput analysis of the consumer and producer, used as stand-alone applications, is presented in Figure 2. The first plot shows the variation with the different number of threads and with different interface delays. In this case, the fixed window size was 64 packets, the number of threads was 16, and the maximum interface delay was 100ms. Because the congestion control algorithm uses a fixed window, the performance decreases with added delay

[1]Boost C++ Libraries: https://www.boost.org/
[2]puppet: https://puppet.com/

on the interface. For a 2.4 GB CMS data file, the maximum throughput achieved was close to 800 Mbps, when the consumer was using four processing threads. When varying the pipeline size, we observed that the overall throughput is higher than in the first test, but it still fell with increasing interface delay, as shown in Figure 2. Compared with standard NDN tools for copying files (i.e., ndnput/cat chunks [15]), the consumer-producer applications achieved more than an 8x increase in throughput performance when no delay was applied on the interface.

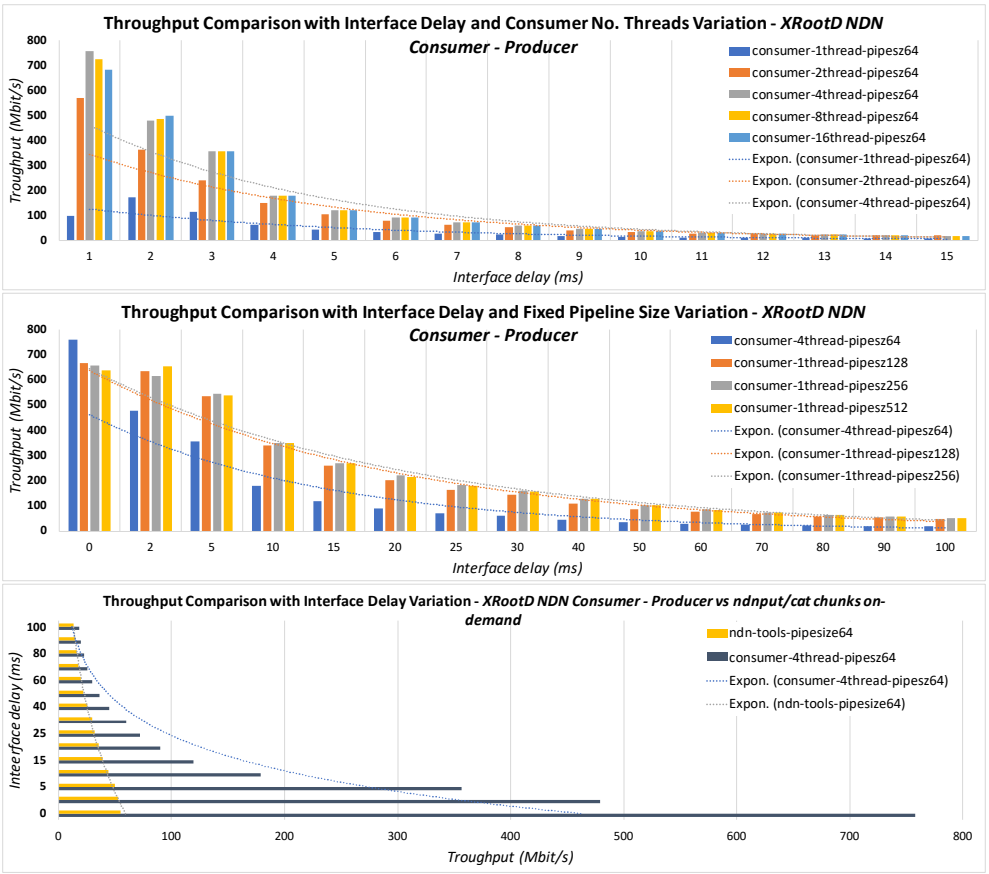

Figure 2: Performance analysis of the SANDIE consumer and producer applications

The XRootD NDN based OSS plug-in performance was also compared against standard CMS tools: XRootD with a TCP based POSIX OSS plug-in and the Fast Data Transfer (FDT) [18] application. The same hardware configuration as in the previous analysis was used, with the same file sizes. The Xrdcp client was used in order to copy files using XRootD, which also offers the possibility of varying the number of streams, a similar functionality to that found in FDT. The results can be observed in Figure 3, where the XRootD POSIX based plug-in has the best performance when interface delay is 2ms. It can also be observed that FDT performance is more stable with increased interface delay.

This first series of tests and analysis allowed us to understand how we can integrate NDN-based solutions with the CMS (and other) physics workflows and also that ndn-cxx and NFD are not the ideal choices for production-grade integration because of the low upper bound on throughput performance. Following the NIST team's work, that moved away from NFD

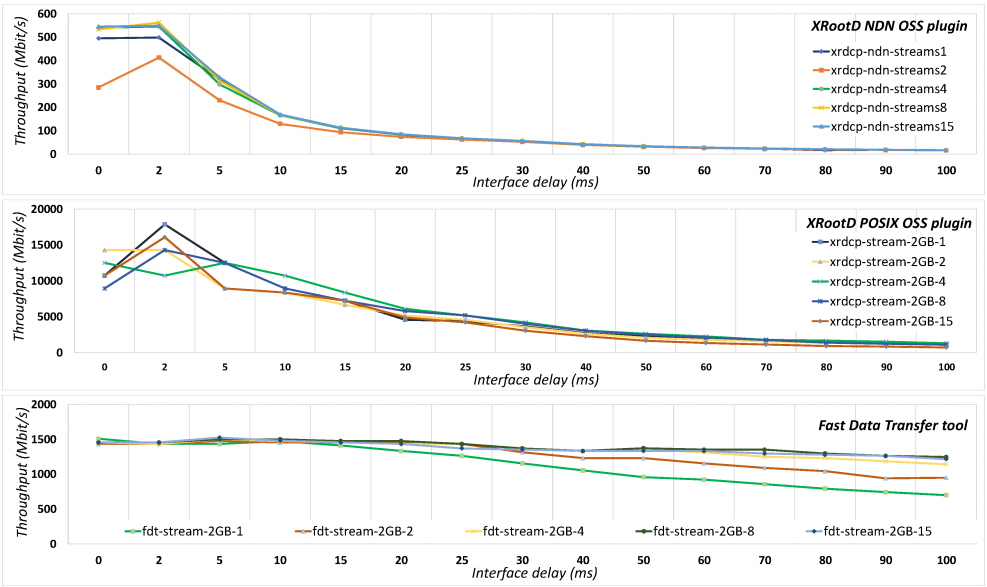

Figure 3: Performance comparison of consumer and producer and standard CMS tools

in order to develop a new DPDK-based forwarding solution (NDN-DPDK) to achieve high throughput, SANDIE also made the paradigm shift in its development process in order to deliver a solution that CMS could consider for distributing data. The performance of the new forwarder with LHC data is discussed in Section 5.

5 XRootD NDN-DPDK Open Storage System plug-in

The SANDIE team has applied the recently-developed high-speed NDN-DPDK forwarder on a testbed environment. NDN-DPDK [19] is a high-speed NDN forwarder developed with the Data Plane Development Kit (DPDK)[3] by NIST. NDN-DPDK achieves superior throughput performance by enabling direct access to the server's network interface cards (NICs), thus eliminating context switching time by bypassing Linux kernels, and exploiting a multi-threaded structure to greatly accelerate lookup and packet processing. Early benchmarking results, presented by NIST, demonstrated up to 62.8 Gbps of throughput on a local testbed, with six forwarding threads and optimized data payload size and packet name length.

At the Supercomputing 2019 conference (SC19) [4], the SANDIE team gave the first demonstration [20] that the NDN-DPDK can achieve comparably excellent throughput performance (per thread) with real LHC data on a layer-2 transcontinental network testbed. CMS data from the producer at Caltech in Pasadena was transferred to the consumer located at the Caltech SC19 booth in Denver. As shown in Figure 4, the demo achieved an impressive mean throughput of 6.7 Gbps over the wide area path between the consumer at the SC19 booth and the Data producer at Caltech in Pasadena. The performance was achieved using two single threaded applications running the CentOS 8 operating system. The applications used for this demo were completely rewritten using NDN-DPDK. They follow the same architecture as presented in Figure 1, however they were not so advanced in the development. Features such

[3]NDN Packet Format Specification 0.3 documentation: http://named-data.net/doc/NDN-packet-spec/current/
[4]SC19 Conference Website: https://sc19.supercomputing.org/

as multi-threaded support, congestion control and retransmission logic which were missing, are currently under development, and so the performance may be further improved in the future.

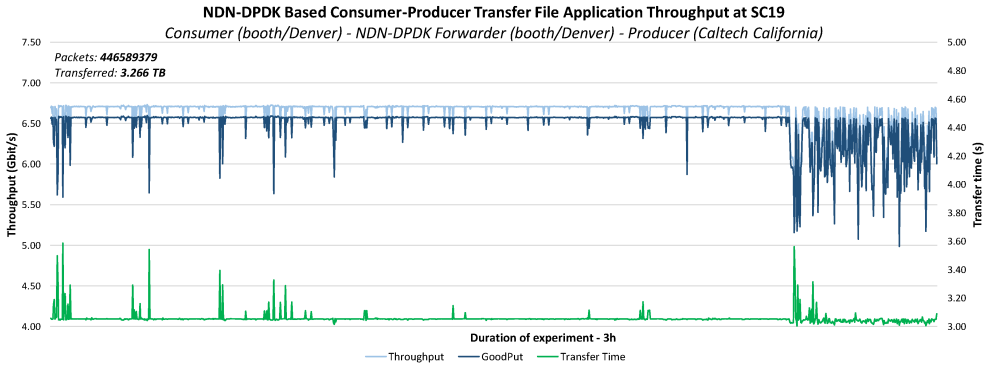

Figure 4: SC 19 Demo: Real-time Performance Plot

The demo points to the great promise of NDN for accelerating data transport and management for the LHC and other data-intensive science applications. The throughput achieved at SC19 over a transcontinental wide area network closely matches the throughput observed by NIST [19] with a single thread over a local testbed. Since NIST has shown that the forwarding performance of the NDN-DPDK forwarder increases linearly with the number of threads, we expect to achieve throughputs up to the 100 Gbps range over the wide area by using multi-threaded consumer and producer applications.

6 Conclusions

This work presents a successful integration of the legacy XRootD framework with NDN primitives, by implementing a new OSS plug-in. The new architecture can provide location transparent data access and simplify XRootD's data location services that are currently implemented through a set of "redirectors" [21]. At this point, the integration has been achieved only with the consumer object developed based on ndn-cxx API. The performance results as well as a deep analysis of the entire code flow showed that the main bottleneck is the NFD forwarder, which limited the application to a maximum throughput of 1 Gbps.

Following further development of new consumer and producer applications using the NDN-DPDK forwarder, higher throughput results were shown at the SC19 Network Research Exhibition. The new applications have a completely different logic from the ones based on the ndn-cxx API, but the architecture remains the same. The new developments show a mean throughput of 6.7 Gbps over a transcontinental testbed. These are promising results and we are continuing to work towards the integration of NDN-DPDK with XRootD, as well as further throughput improvements by:

- Implementing multi-thread support on both consumer and producer applications. The consumer object will be used by XRootD, which for large files is distributing the work among a pull of threads, while the producer will be able to supply multiple concurrent requests from different clients with high performance.
- Implementing a common interface for different file systems supported by CMS (i.e., HDFS[5] and CephFS[6]) on the producer side. Currently, all file requests are handled us-

[5]The Hadoop Distributed File System: https://storageconference.us/2010/Papers/MSST/Shvachko.pdf
[6]Ceph File System: https://docs.ceph.com/docs/master/cephfs/

ing POSIX file system calls, a factor that can become the main bottleneck of the entire flow if not taking advantage of the high performance file access of distributed file systems.

- Containerizing NDN-DPDK consumer and producer applications using Docker containers. DPDK requires many dependencies (e.g., kernel version, network cards, drivers) that are hard to configure at a larger scale. By packaging all requirements inside containers, it's easier for CMS to adopt the proposed NDN solution.
- Deploying and testing of prototypical CMS workflows using the new NDN-DPDK forwarder in the context of the XRootD plug-in, to demonstrate its performance and scalability.

In the longer term we plan to explore an alternate scheme of data location and gathering using NDN together with the CEPH file system, and to further accelerate the packet forwarding functions through the use of smart NICs with FPGAs.

7 Acknowledgements

This work is supported in part by grants from DOE office of Advanced Scientific Computing Research (SDN-NGenIA - DOE/ASCR, DE-SC0015527 and SENSE, DOE/ASCR, DE-SC0015528) and the National Science Foundation (NSF-1246133, NSF-1341024, NSF-1120138).

References

[1] E. Yeh, H. Newman, and C. Papadopoulos, *Sandie: SDN Assisted NDN for Data Intensive Experiments*, 2017, NSF Award 1659403

[2] L. Evans and P. Bryant, *LHC Machine*, Journal of Instrumentation, **vol. 3**, p. S08001, 2008

[3] E. M. Yeh, T. Ho, Y. Cui, M. Burd, R. Liu and D. Leon, *VIP: A Framework for Joint Dynamic Forwarding and Caching in Named Data Network*, Proceedings of ACM Conference on Information-Centric Networking, pp. 117-126, 2014

[4] A. Dorigo, P. Elmer, F. Furano, and A. Hanushevsky, *XROOTD - A highly scalable architecture for data access*, WSEAS Transactions on Computers, **vol. 1**, no. 4.3, pp. 348–353, 2005

[5] *XRootD: Open File System & Open Storage System Configuration Reference*, [Accessed 2020-1-27], [Online], Available: https://xrootd.slac.stanford.edu/doc/dev50/ofs_config.pdf

[6] *The NDN based File System XRootD plugin component for Open Storage System and a suitable NDN producer*, [Accessed 2020-1-27], [Online], Available: https://github.com/cmscaltech/sandie-ndn/tree/master/xrootd-ndn-oss-plugin

[7] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, *Named Data Networking* (ACM SIGCOMM Computer Communication Review), **vol. 44**, no. 3, pp. 66–73, 2014

[8] S. Shannigrahi, C. Papadopoulos, E. Yeh, H. Newman, A. J. Barczyk, R. Liu, A. Sim, A. Mughal, I. Monga, J.-R. Vlimant et al., *Named Data Networking in Climate Research and HEP Applications* in Journal of Physics: Conference Series, **vol. 664**, no. 5. IOP Publishing, 2015, p. 052033

[9] *NDN Annual Progress Summaries* , [Accessed 2020-1-27], [Online], Available: https://named-data.net/project/annual-progress-summaries/

[10] *NDN Publications* , [Accessed 2020-1-27], [Online], Available: https://named-data.net/publications/

[11] *NDN Name Format*, [Accessed 2020-1-27], [Online], Available: http://named-data.net/doc/NDN-packet-spec/current/name.html

[12] *NDN Packet Format Specification*, [Accessed 2020-1-27], [Online], Available: https://named-data.net/doc/NDN-packet-spec/current/

[13] *ndn-cxx: NDN C++ library with eXperimental eXtensions*, [Accessed 2020-1-27], [Online], Available: https://github.com/named-data/ndn-cxx

[14] *NFD - Named Data Networking Forwarding Daemon*, [Accessed 2020-1-27], [Online], Available: http://named-data.net/doc/NFD/current/

[15] *NDN Tools and Applications*, [Accessed 2020-1-27], [Online], Available: https://named-data.net/codebase/applications/

[16] *The SANDIE NDN software stack repository*, [Accessed 2020-1-27], [Online], Available: https://github.com/cmscaltech/sandie-ndn/tree/master/packaging/RPMS/x86_64

[17] *NDN-DPDK: High-Speed Named Data Networking Forwarder*, [Accessed 2020-1-27], [Online], Available: https://github.com/usnistgov/ndn-dpdk

[18] *Fast Data Transfer*, [Accessed 2020-1-27], [Online], Available: http://monalisa.cern.ch/FDT/

[19] J. Shi, *Ndn-dpdk: High-speed named data networking forwarder*, National Institute of Standards and Technology, 2020, [Accessed 2020-1-7]. [Online]. Available: https://doi.org/10.18434/M32111

[20] *NRE-19: SC19 Network Research Exhibition: Caltech Booth 543 Demonstrations*, [Accessed 2020-1-27], [Online], Available: https://sc19.supercomputing.org/app/uploads/2019/11/SC19-NRE-024.pdf

[21] *Using Xrootd Service (AAA) for Remote Data Access*, [Accessed 2020-6-06], [Online], Available: https://twiki.cern.ch/twiki/bin/view/CMSPublic/WorkBookXrootdService#ReDirector