N-DISE: NDN-based Data Distribution for Large-Scale Data-Intensive Science

Yuanhao Wu
Faruk Volkan Mutlu
Yuezhou Liu
Edmund Yeh
wu.yuanh@northeastern.edu
mutlu.f@northeastern.edu
liu.yuez@northeastern.edu
eyeh@ece.neu.edu
Northeastern University
Boston, MA, USA

Ran Liu
liuranapply@gmail.com
Google
Madison, WI, USA

Catalin Iordache
Justas Balcas
Harvey Newman
Raimondas Sirvinskas
catalinn.iordache@gmail.com
jbalcas@caltech.edu
newman@hep.caltech.edu
raimis.sirvis@gmail.com
California Institute of Technology
Pasadena, CA, USA

Michael Lo
Sichen Song
Jason Cong
Lixia Zhang
milo168@g.ucla.edu
songsichen123@gmail.com
cong@cs.ucla.edu
lixia@cs.ucla.edu
UCLA
Los Angeles, CA, USA

Sankalpa Timilsina
Susmit Shannigrahi
stimilsin43@tntech.edu
sshannigrahi@tntech.edu
Tennessee Technological University
Cookeville, TN, USA

Chengyu Fan
chengy.fan@gmail.com
Pure Storage
Mountain View, CA, USA

Davide Pesavento
Junxiao Shi
Lotfi Benmohamed
davide.pesavento@nist.gov
junxiao.shi@nist.gov
lotfi.benmohamed@nist.gov
NIST
Gaithersburg, MD, USA

ABSTRACT

To meet unprecedented challenges faced by the world's largest data- and network-intensive science programs, we design and implement a new, highly efficient and field-tested data distribution, caching, access and analysis system for the Large Hadron Collider (LHC) high energy physics (HEP) network and other major science programs. We develop a hierarchical Named Data Networking (NDN) naming scheme for HEP data, implement new consumer and producer applications to interface with the high-performance NDN-DPDK forwarder, and build on recently developed high-throughput NDN caching and forwarding methods. We integrate NDN systems concepts and algorithms with the mainstream data distribution, processing, and management system of the Compact Muon Solenoid (CMS) experiment. We design and prototype stable, high-performance virtual LANs (VLANs) over a continental-scale wide area network testbed. In extensive experiments, our proposed integrated system, named NDN for Data-Intensive Science Experiments (N-DISE), is shown to deliver LHC data over the wide area network (WAN) testbed at throughputs exceeding 31 Gbps between Caltech and StarLight, with dramatically reduced download time.

ICN '22, September 19–21, 2022, Osaka, Japan
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9257-0/22/09…$15.00
https://doi.org/10.1145/3517212.3558087

CCS CONCEPTS

• **Networks** → **Network architectures**; **Network protocols**; **Network algorithms**; **Network performance evaluation**.

KEYWORDS

named data networking, information centric networking, naming, caching, forwarding, high energy physics, large hadron collider.

ACM Reference Format:
Yuanhao Wu, Faruk Volkan Mutlu, Yuezhou Liu, Edmund Yeh, Ran Liu, Catalin Iordache, Justas Balcas, Harvey Newman, Raimondas Sirvinskas, Michael Lo, Sichen Song, Jason Cong, Lixia Zhang, Sankalpa Timilsina, Susmit Shannigrahi, Chengyu Fan, Davide Pesavento, Junxiao Shi, and Lotfi Benmohamed. 2022. N-DISE: NDN-based Data Distribution for Large-Scale Data-Intensive Science. In *9th ACM Conference on Information-Centric Networking (ICN '22), September 19–21, 2022, Osaka, Japan.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3517212.3558087

1 INTRODUCTION

In spite of network technology advances, the largest data- and network-intensive science programs including the Large Hadron Collider (LHC) [24] program, the Joint Genome Institute applications and the BioGenome project [7, 11], face 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 operation and collaboration within global scientific enterprises each comprised of hundreds to thousands of scientists.

One of the most data-intensive programs is the LHC high energy physics (HEP) program, which has an estimated 900 petabytes under management at more than 170 sites in the US and around the world. The corresponding network traffic, which already exceeded 1600 petabytes (1.6 exabytes) in 2019, is projected to grow by another order of magnitude or more by the start of the upgraded High Luminosity LHC (HL LHC) program beginning in 2027 [2].

The HEP community is not the only one facing these challenges. Take, for example, human genome sequencing in the biology community. It is anticipated that around 1 exabyte of human genome data exists [22] in institutional repositories, research labs, and in the genome sequencing industry. Furthermore, these numbers are only for human genome data—similar amounts of data exist for plants [32], animals [31], viruses [13], and bacteria [33]. Finally, the Earth Biogenome project [8] aims to sequence all known species in the tree-of-life. The data volume for this project alone is expected to exceed an exabyte [7].

To meet the sustained and peak needs of these data- and network-intensive science fields within feasible resource constraints, more efficient and less resource-intensive data access and distribution methods are required. An effective approach for this is Named Data Networking (NDN), a leading data-centric network architecture with intuitive and efficient naming, security and provenance, data access, caching, and forwarding methods and structures.

In this paper, we describe an innovative effort to meet unprecedented challenges faced by the world's largest data- and network-intensive science programs, by designing and implementing a new, highly efficient and field-tested data distribution, caching, access and analysis system for the LHC HEP network and other major science programs. We name this system *NDN for Data Intensive Science Experiments* (N-DISE). Our major contributions include the following:

- We develop and prototype the first high-performance NDN-based integrated data delivery system for large-scale data-intensive science applications.
- We develop an NDN naming scheme for CMS data. To address the incongruity where the names of the CMS data at different granularity do not share the same prefix, we introduce an additional name mapping scheme to associate data objects used by workflows (e.g., blocks of CMS data) with data units directly used in network requests (e.g., files of CMS data).
- To achieve exceptional packet forwarding performance, we develop consumer and producer applications that interface with the multi-threaded high-throughput NDN-DPDK forwarder [29]. We also develop the NDNc library which offers APIs for these applications to efficiently communicate with the NDN-DPDK forwarder using memif, a shared memory packet interface providing high performance packet transmission.
- To achieve outstanding caching and forwarding performance, we integrate the VIP joint caching and forwarding algorithm [34], which has leading throughput, delay, and cache hit ratio performance, with the NDN-DPDK forwarder and the new NDNc-based consumer and producer applications.
- We establish a high-bandwidth, long-range wide area network testbed, with nodes at Northeastern (MGHPCC), UCLA, Tennesee Tech, and StarLight Chicago, with 100 Gbps connections running from MGHPCC through StarLight to Caltech.
- Working with many partners (Internet2, ESnet, CENIC), we provision dedicated VLANs over which we can directly experiment with NDN protocols and algorithms at layer 2.
- We extensively experiment with the NDNc-based consumer and producer applications, the NDN-DPDK forwarder, and VIP joint caching and forwarding. We carry out extensive testing using LHC data files on the fully operational long-range WAN testbed. We show that the deployed system can reach a maximum throughput of 31 Gbps, and VIP joint caching and forwarding can simultaneously increase throughput and decrease download delay, relative to multiple baseline algorithms.

The use of NDN-based data delivery systems for data-intensive science applications has seen limited investigation thus far, in the context of climate science [18, 25], high energy physics [9, 12], and genomics [26]. In contrast to the cited works, the N-DISE system presented in this paper aims to achieve significantly higher throughput by effectively integrating a number of essential high-performance system components, and to demonstrate performance gains over WAN testbeds operating at link capacities exceeding 100 Gbps.

The rest of this paper is organized as follows. In Section 2, we discuss the NDN naming scheme for CMS data. In Section 3, we present the details of our software implementation for the NDNc-based consumer and producer applications, as well as the VIP joint caching and forwarding algorithm. In Section 4, we give an overview of our WAN testbed. In Section 5, we report the results of our experiments. In Section 6, we discuss some of the major lessons we derived from the development and experimentation of N-DISE. Finally, in Section 7, we conclude the paper with a brief summary of our efforts and provide future directions we will take with this work.

2 NAMING OF LHC DATA

The LHC CMS generates petabytes of data from experiments, analysis, and simulations each year and stores them in special data formats. For example, CMS generates a large amount of physics analysis data that gets stored in various Analysis Object Data (AOD) formats such as MiniAOD and NanoAOD [19, 23]. This large volume of data is grouped into a hierarchical structure consisting of three levels in increasing order of granularity: *dataset*, *block* and *file*. A dataset contains several data blocks and a data block contains several files.

The content names at each of these granularity levels are hierarchical. These names can be converted naturally into hierarchical NDN names [28, 27] by removing the unnecessary components (e.g., the first two components of the file name that describes the filesystem location where the file is stored), reorganizing components, and adding new components where necessary. Such an example file name can be: "`/store/data/Run2017F/SingleMuon/MINIAOD/17Nov2017-v1/60000/C47DBE62-67E7-E711-B896-5065F37D51B2.root`".

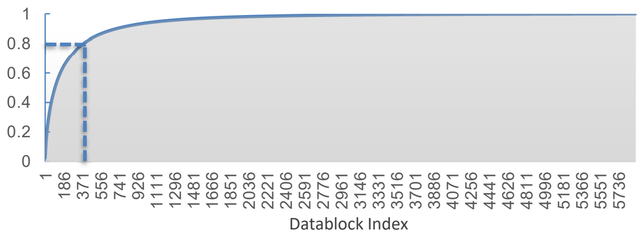


Figure 1: Percentage of requests covered by data blocks (February–March 2019, Caltech).

Although naming schemes at each of the granularity levels are hierarchical, the file, dataset, and block names might follow different naming hierarchies. For example, the file above may belong to a block with the name "`/SingleMuon/Run2017F-17Nov2017-v1/MINIAOD#13f6ce2c-e734-11e7-af75-02163e01b396`" and a dataset with the name "`/SingleMuon/Run2017F-17Nov2017-v1/MINIAOD`". Note that these names do not share the same prefix with the actual file name, which is used by the NDN network for in-network forwarding and caching.

The CMS users can use these names to request desired data at the file granularity. They can also request part of the files by specifying dynamic functions that can filter and create smaller data from large files. For example, "`/store/data/Run2017F/SingleMuon/MINIAOD/17Nov2017-v1/60000/C47DBE62-67E7-E711-B896-5065F37D51B2.root/<byte-range>`" can be used to extract the specified byte-range at the data source and transfer the extracted data. However, in most cases, the user requires all the files in a whole block or dataset, since the files contained in a block or a dataset are always strongly correlated.

To address the incongruity where users want to utilize data blocks but request files by their names, we have built a name mapping hash table to record the relationships between files and blocks. In our current design, each forwarder maintains a name mapping hash table to allow caching and forwarding algorithms to operate at the appropriate granularity (e.g. block level) while users still utilize file names for content retrieval. The name mapping table need not change frequently due to the relatively static nature of the CMS data structure, and are assumed to be obtained and updated through the control plane, as are routing updates.[1]

To further motivate the concept of name mapping between different data granularities, we analyzed CMS workflow logs of February and March 2019 and found that while the request frequency for files is relatively flat, the request frequency for data blocks has sharper decay. At the Caltech site, roughly 6000 blocks with unique names were requested, but the 400 most requested blocks covered roughly 80% of all requests, as Figure 1 shows. Therefore, in the VIP caching and forwarding algorithm we describe in Section 3, in order to achieve scalability implementation, we maintain control state only over these most popular 400 data blocks.

[1]In order to quantify the effect of the name mapping table in terms of delay, we measured the delay incurred by the name mapping as a function of the table size. In our test results, the additional delay is less than 0.2 milliseconds for table sizes up to 6000 entries and the total number of requested file names up to 60000. This indicates that the delay effect of name mapping is negligible in terms of system performance.

3 SOFTWARE IMPLEMENTATION

Software makes up a large part of the development of N-DISE. In particular, to achieve exceptional packet forwarding performance, we build consumer and producer applications that interface effectively with the multi-threaded high-throughput NDN-DPDK forwarder. To achieve outstanding caching and forwarding performance, we implement the VIP joint forwarding and caching algorithm [34] aimed at optimizing usage of storage and bandwidth resources, and integrate it with the NDN-DPDK forwarder. In this section, we provide a detailed discussion of these software components.

3.1 NDNc Consumer and Producer

To deploy an end-to-end solution for data distribution using NDN as the underlying architecture, we implement both a consumer and a producer application, the two fundamental entities in any NDN data transfer process. The consumer acts as a client that composes NDN Interest Packets (analogous to a query) to request data from the network, while the producer is the server with access to the data on a file system and publishes it in the form of NDN Data Packets (analogous to a response).

To achieve exceptional packet forwarding performance, we design our applications to communicate with the high-performance NDN-DPDK forwarder developed by NIST [29], which has demonstrated leading throughput performance using multi-threaded forwarding. One challenge with this approach is the lack of any NDN C++ library compatible with this new forwarder. We therefore develop our own lightweight C++ library called NDNc [16] that bridges the ndn-cxx library [21] with the NDN-DPDK forwarder, while adding a small number of common features needed by the consumer and producer applications. The NDNc library offers PIT token support (required for interoperability with NDN-DPDK), a *memif*-based face [1] capable of providing efficient packet transmit and receive functions to and from a locally running forwarder, an NDN packet encoder and decoder, Interest pipelining controlled by either a fixed-window or a congestion-aware AIMD algorithm, and a GraphQL [10] client that can configure the local forwarder by creating and deleting faces and registering name prefixes.

Using NDNc, we have developed a file transfer consumer and producer for benchmarking the performance characteristics of the library. The consumer and producer are deployed using Docker containers. Both entities follow a predefined naming scheme, and use the same name prefix. There are two types of Data Packets that can be requested: one for retrieving file information (or metadata) and the other for retrieving the file contents. Upon receiving an Interest Packet, the producer parses its name in order to extract the file path and the type of data it requests. For file information, the producer calls the POSIX `stat` system call on the file path and then embeds the answer in a common `Metadata` structure in NDNc for better encoding and decoding. In the case of Interests requesting content from the file, the producer calls the POSIX `read` system call to obtain the desired range of bytes from the file, as specified by the segment number in the Interest name.

The producer application runs in the network indefinitely, constantly waiting for new requests, while the consumer initiates a file transfer and then terminates. On initialization, the consumer uses

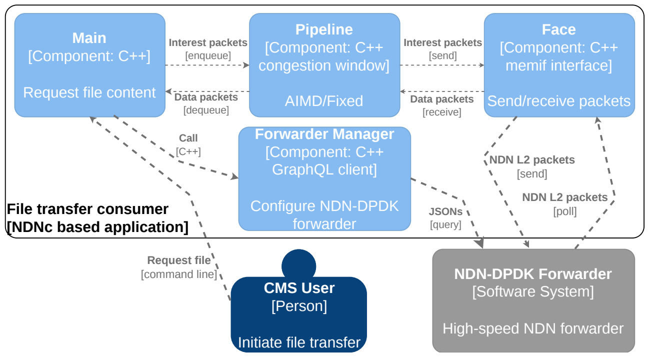


Figure 2: Component diagram for the NDNc consumer application.

the GraphQL client to configure a new interface with the local forwarder and then requests metadata information about a file passed as an input argument via the command line. If the requested file is found, the consumer constructs two worker groups, one for sending Interests and one for receiving Data Packets and assembling the result. The Interest Packets are passed to the *pipeline* that manages the congestion window and then to the local *face* that encodes the Interest to NDN L2 packets and finally sends the packets to the forwarder through the *memif* interface. The data is decoded from L2 and then passed to the receiving worker group. The pipeline takes care of NACK packets and timeouts. Once the file transfer is complete, the consumer destroys its face with the forwarder. The components of the consumer are shown in Figure 2.

3.2 VIP Joint Caching and Forwarding

To achieve outstanding caching and forwarding performance, we build a single-threaded software implementation of the VIP caching and forwarding algorithm, which has theoretically proven and numerically demonstrated leading performance in throughput, delay, and cache hit ratio [34]. We then integrate the VIP implementation with the high-performance NDN-DPDK forwarder [29]. We refer to this integrated caching and forwarding software suite as VIP-NDN-DPDK. Before we present the details of this implementation, we first give an overview of the VIP framework, as well as the distributed joint caching and forwarding algorithm we implement.

The VIP framework employs a virtual control plane, in which user demand rate for data objects in the network is measured through Virtual Interest Packet (VIP) counts. When a request for a data object enters the network at a given node, a corresponding VIP for that object is generated at that node, and the corresponding VIP count for the object is incremented at that node. Each node in the network maintains a separate VIP counter for each object, periodically communicates its VIP counts with its neighbors, and updates the same counters. The evolution of the VIP counters follows the controlled queuing dynamics of the VIPs in the virtual plane. Thus, when a node forwards VIPs for a data object to a neighboring node according to the control algorithm, the sending node decrements the VIP counter for the data object, and the receiving neighboring node increments its VIP counter for the same object. At the source node for a data object, the VIP counter for that object is fixed at 0. If a node is not the source node for a data object but is caching the object, the corresponding VIP counter at that node is decremented over time at a rate proportional to the readout rate of the node, i.e., the rate at which it can produce copies of the object.

At an intuitive level, one can better understand the principle underlying the VIP framework by viewing VIP counts as *potentials*. At the entry points of requests for a data object, the VIP count (potential) for the data object is high, while at the caching node and source node for the data object, the potential is low. In the virtual plane, the VIPs for the data object follow a gradient from the high-potential entry points to the low-potential caching and source nodes, where they are removed from the network. By employing a virtual control plane separate from the actual plane (where Interest and Data Packets are transmitted), we can accurately measure and track demand in the network, even in the presence of mechanisms such as Interest aggregation.

The VIP joint caching and forwarding algorithm operates within the VIP framework, and aims to control the flow of VIPs in the virtual plane to achieve optimal network load balancing, thereby maximizing the demand rate that can be satisfied by the network. Forwarding in the virtual plane applies the backpressure algorithm to VIP counts and allocates VIP transmissions to minimize VIP count differences on each link. Caching in the virtual plane aims to cache the objects with the highest VIP counts, subject to the cache capacities of nodes. In this way, the algorithm efficiently drives VIPs toward data source and caching nodes, while ensuring that demand does not build up in any part of the network. In the actual plane, forwarding and caching strategies observe the flow of VIPs resulting from the virtual plane algorithm and use this information to make forwarding and caching decisions for Interest and Data Packets. The operation of this algorithm, along with the mechanics of the VIP framework, are illustrated in Figure 3. For further details regarding the VIP framework and the VIP algorithm, please see [34].

In VIP-NDN-DPDK, VIP control state is maintained for a selected set of *data objects*, named *registered data objects*, where each data object consists of a set of *chunks* (equivalently, packets). In the CMS application, for example, the data objects can be chosen to be the most frequently requested blocks (determined via e.g. statistical estimation methods). The names of these data objects are stored in a hash table called the *Name Map Table*, which also stores the mapping between chunk names and their corresponding data object names. In the CMS application, for example, the Name Map Table contains the mapping between file chunk names and the corresponding block names (as mentioned in Section 2). The VIP algorithm is only applied to these registered data objects while other data objects are not cached and are forwarded using the default forwarding strategy provided by the NDN-DPDK forwarder.

At each node, VIP-NDN-DPDK updates VIP counts for (registered) data objects in the virtual plane periodically, asynchronous of other nodes. Specifically, starting from the launch of VIP-NDN-DPDK, each node runs its own timer, exchanges VIP control packets with its neighbors every T seconds, and updates VIP related statistics for each data object according to Algorithm 2 in [34]. At the start of each period, each node n pulls a Data Packet containing the VIP counts $V_m^k(t)$ for all data objects k from each neighboring node m, by sending an Interest Packet with the name "/ndn/vip/A" and a

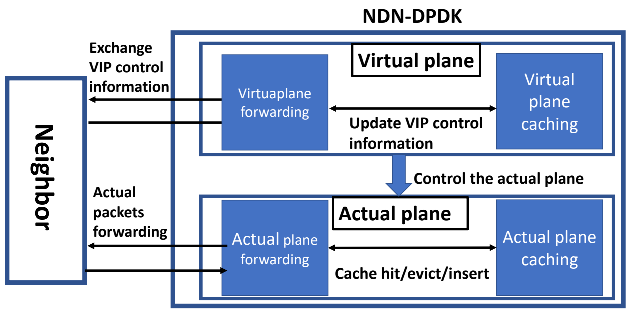


Figure 3: VIP implementation diagram.

nonce. Any neighbor m sending back the VIP count information to node n also sends back an Interest Packet with the name "`/ndn/vip/B`" and a *nonce* to retrieve a Data Packet containing the number of VIPs to transmit (in the virtual plane) $\mu_{nm}^k(t)$ from node n to node m for data object k in this period. All VIP control information is stored in a SipHash-based hash table called *VIP Table*.

In order to quickly populate the virtual plane with VIPs to form nontrivial VIP count gradients, we use an amplification factor α, e.g. $\alpha = 2$, to scale up the consumer generated VIPs of data object k at node n from $A_n^k(t)$ to $\alpha A_n^k(t)$ [34]. That is, for each externally arriving request at node n for data object k, we increment $V_n^k(t)$ by α.

Virtual plane caching decisions are also performed periodically. For simplicity, we assume all registered data objects have the same size in bits. At the start of the ith period, i.e. time t_i, we update the VIP counts as in [34] and decrease $V_n^k(t)$ by r_n for all cached data objects k, where r_n is the DRAM read out rate (in data objects per time period). We then update the cache scores $CS_n^k(t)$ for all data objects. In contrast to Algorithm 2 in [34], the cache score $CS_n^k(t)$ for data object k and node n is implemented as the exponentially weighted cumulative received VIPs: $CS_n^k(t_i) = CS_n^k(t_{i-1})\ exp\,(-\beta * (t_i - t_{i-1})) + R_n^k(t_i)$, where β is a parameter chosen empirically, e.g. $\beta = 0.000015$, and $R_n^k(t_i)$ is the number of VIPs received at node n for data object k from time t_{i-1} to t_i.

In the actual plane, we forward Interest Packets of data object k at node n to the neighboring node m with the maximum exponentially weighted cumulative VIP sending rate for data object k from node n: $\hat{v}_{n,m}^k(t_i) = \hat{v}_{n,m}^k(t_{i-1})\ exp\,(-\beta * (t_i - t_{i-1})) + v_{n,m}^k(t_i)$, where $v_{n,m}^k(t_i)$ is the number of VIPs sent from n to m in the time period from t_{i-1} to t_i.

The VIP algorithm requires all chunks of a data object to be forwarded and cached together. To achieve this, we implement a forwarding direction lock with an associated timer. Initially, the forwarding directions for all data objects are unlocked. For each arriving Interest Packet for data object k arriving to a forwarder n, if the forwarding direction for data object k is unlocked, the forwarder n chooses a new forwarding direction for k corresponding to the maximum exponentially weighted cumulative VIP sending rate for data object k from node n, as described in the previous paragraph. At the same time, forwarder n starts a timer with an expiration sufficiently long to send the entire Interest Packet at the corresponding link rate. Otherwise if the forwarding direction for data object k is locked, the forwarder restarts the corresponding timer with the same expiration, and forwards the Interest in the previously stored direction.

In the *Content Store* (CS) of the actual plane, the VIP algorithm caches only the registered data objects. The objective of the caching policy is to maximize the total cache score of the cached data objects, as described in detail in Algorithm 2 of [34]. While the original NDN-DPDK controls cache space at the chunk level, the VIP algorithm controls cache space at data object level. To achieve the latter without changing the basic CS framework of NDN-DPDK, we implement a hash table called the *CS-VIP* table. When a given forwarder decides to cache a data object, it inserts the name of the data object into the CS-VIP table and maps the name to a list of names for all the data chunks belonging to the data object (which arrives to the forwarder since Interest Packets for all chunks of a given data object are forwarded in the same direction). When the forwarder evicts a data object, it uses the CS-VIP table to evict all data chunks belonging to the data object.

4 WIDE AREA NETWORK TESTBED

In order to evaluate the N-DISE system in a real-world wide area network (WAN) setting, we build a high-bandwidth WAN testbed that includes 7 high-performance servers. The testbed spans 5 sites: Massachusetts Green High Performance Computing Center (MGHPCC), Caltech, StarLight Chicago, UCLA, and Tennessee Tech. The MGHPCC server is owned by the Northeastern University group. The Caltech and StarLight sites host two servers each, while all other sites have one. Each of these servers is equipped with high-end Intel Xeon or AMD EPYC CPUs to take advantage of modern architectural features, large amounts of memory ranging from 96 GB up to 512 GB, large pools of both HDD and NVMe SSD storage as well as Mellanox ConnectX series of network interface cards (NICs) supporting 100GbE. In general, our hardware choices are guided by the list of hardware known to perform well with DPDK and NDN-DPDK [20, 17].

The topology of the testbed and the connectivity among sites are depicted in Figure 4, which shows the maximum theoretical bandwidth and the measured round trip time (RTT) of each link. Note that the servers at the Northeastern (MGHPCC) and the Caltech sites are locally connected via 100GbE while those at the StarLight site are connected via 40GbE.

All servers in the testbed are connected by tagged VLANs with support from CENIC, NOC, Internet2, and individual server sites. For these connections, we attempted to allocate capacities up to the maximum bandwidth limits of the underlying links (as shown in Figure 4). However, the results fell short of this goal due to various issues such as the lack of quality of service (QoS) guarantees, unexpected traffic interruptions, traffic blocking from unknown middleboxes, server tuning problems, and the lack of high-speed switches. In our measurements using *iperf3*, the capacity of the MGHPCC–StarLight link was reported as only 16 Gbps and the capacity of the StarLight1–Caltech1 link was reported as 31 Gbps.

5 EXPERIMENTAL RESULTS

We evaluate the performance of the N-DISE system by running several experiments over the WAN testbed described in Section 4.

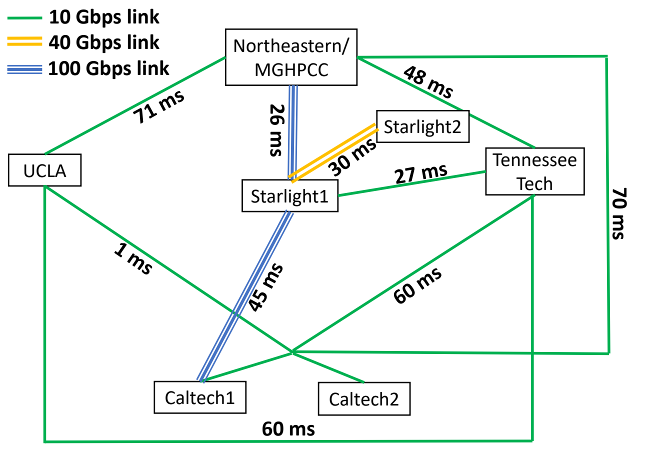


Figure 4: Topology of the WAN testbed. The link capacities are shown in the diagram. The RTTs are shown along the connections.

In this section, we first discuss the results of a throughput test conducted over the testbed. We then discuss the impact of VIP joint caching and forwarding on system performance.

5.1 Throughput Evaluation

We conduct a throughput test between Caltech1 (request node) and StarLight1 (server node) to demonstrate the baseline potential of N-DISE. We report results on six scenarios with different numbers (6, 9, 12) of concurrently running consumer applications at the request node (Caltech1), with and without caching enabled at the server node (StarLight1). For each scenario, the test is run for 10 minutes. Note that caching is disabled at Caltech1 in all scenarios. While we employ 3 forwarding threads for the scenarios with 6 and 9 consumers, we add another forwarding thread for the 12-consumer scenario. Each consumer application requests a particular file repeatedly for the duration of the test and measures the throughput for each file transmission. Only the data payload of transmissions is considered for these measurements, the overhead of the packet header being small compared to the payload. Immediately before this test, the IP throughput over this path was measured to be 31.8 Gbps using *iperf3*, which serves as a reference point for our results.

Figure 5 shows the average throughput (over 10 minutes) achieved in each scenario in our test. Note that for each scenario, the throughput value is aggregated across all consumer applications. We can easily see that by increasing the number of consumers, we extract more throughput from our network: the throughput is increased by 43% going from 6 to 9 consumers, and by 71% going from 6 to 12. This improvement is sub-linear however, as the throughput per consumer application slightly decreases with an increasing number of consumer applications, since we approach the capacity of the path. The effects of enabling caching at the server node are also seen: a 30%, 60% and 68% increase in throughput is achieved with 6, 9 and 12 consumers respectively. Finally, it should be noted that while

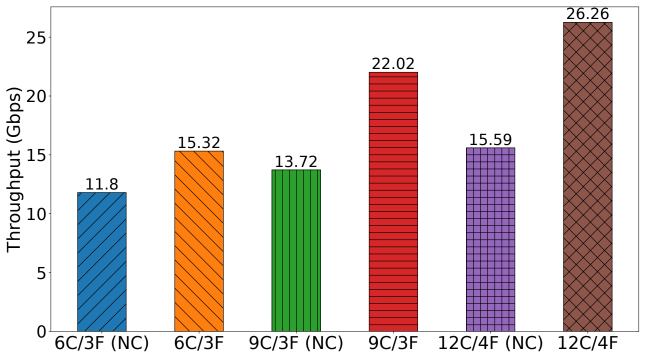


Figure 5: Average throughput from Caltech1 to StarLight1. On the x-axis, "XC/YF" denotes a test with X consumers and Y forwarding threads. "(NC)" indicates that the result is from a no-cache test, otherwise the result is from a test with files cached at the server node (StarLight1).

Figure 5 shows average throughput values, the peak throughput value achieved during the experiment was 31.14 Gbps.

5.2 VIP Caching and Forwarding Evaluation

We evaluate the performance of the VIP algorithm over the service network and multi-path network topologies shown in Figure 6. The service network consists of two consumer nodes, at UCLA and StarLight respectively, two forwarder nodes, at Northeastern University (MGHPCC) and Tennessee Tech respectively, and one server node at Caltech. The multi-path network has two consumer nodes, at Northeastern University (MGHPCC) and Tennessee Tech respectively, one server node at Caltech, and two paths going through the forwarder nodes at UCLA and Tennessee Tech.

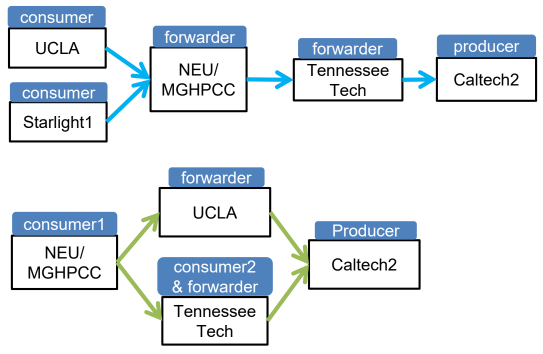


Figure 6: Network topologies used in the tests: service network topology (top), multi-path topology (bottom).

For this performance evaluation, requests are generated for a catalog of 30 registered data objects, each 4 GB in size, which are randomly generated files with CMS file names. In both the service and multipath network topologies, each forwarder is allocated 20 GB of cache space, which enables the caching of 5 data objects.[2]

[2]Due to cache capacity limits, only one block can be cached at each node. We therefore run the VIP caching test at the file level. In this case, each file is considered as a whole

At each request node, we launch one NDNc consumer application. The consumer applications use a fixed window of 8192 packets and a 2-second retransmission timeout (RTO). The window size is chosen according to the bandwidth delay product without caching and is fine tuned to obtain the optimal throughput. The RTO is chosen according to the low-loss characteristics of the wireline testbed. Each consumer application requests data objects sequentially (where the next request is transmitted upon receipt of the previous data response), where each requested data object is randomly and independently chosen from the catalog according to a Zipf distribution with parameter 1. The latter distribution is estimated from the LHC request pattern data presented in Figure 1.

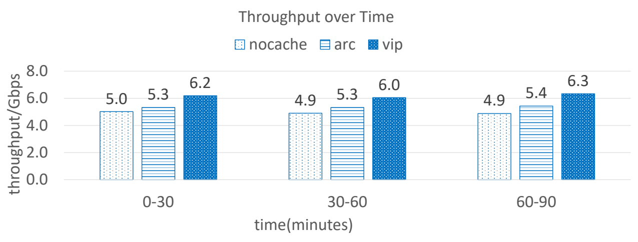
We conduct the test in the service topology for a duration of 1.5 hours and calculate the time-averaged total throughput and average packet delay every 30 minutes. The total throughput is defined as the total number of Data Packet bits retrieved at the consumer nodes (UCLA and StarLight1) from the producer node (Caltech2) or from any of the caching nodes, divided by the length of each test period (i.e. 30×60 = 1800 seconds). The packet delay is defined as the difference between the fulfillment time (i.e., time of arrival of the requested Data Packet) and the creation time of the Interest Packet request. The average delay is defined as the packet delay averaged over all fulfilled Interest Packet requests at the consumer nodes (UCLA and StarLight1) over the testing period (30 minutes).

We investigate the performance of 3 different caching policies (no caching, VIP, and adaptive replacement cache (ARC) [14], which is improved from LRU) and track the cache contents at the forwarder nodes. Note that forwarding is fixed in the service topology.

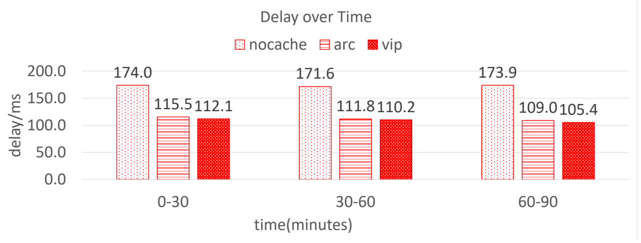
For VIP caching in the service topology, we use a 3-second time slot and a VIP count amplification factor of 1, as described in section 3.2. The VIP time slot length is chosen to be large enough so that each node can build up a reasonably large VIP queue size for every data type in each time slot. The amplification factor potentially allows VIP counts to build up more quickly during the initial transient period. We use small amplification factor and VIP time slot length here since the service topology gathers all request flows into the same path, implying we can readily obtain reasonably large VIP queue sizes. In the case where the network does not generate enough VIPs in each time slot, we have the option of choosing larger amplification factor and VIP time slot length, as we do in the following multi-path topology test.

The results show that the cache contents chosen by VIP at each forwarder node are almost always optimal. The first forwarder (at Northeastern/MGHPCC) always caches the 5 most popular files in the last 30 minutes. The second forwarder (at Tennessee Tech) almost always caches files with popularity ranking 5 through 11. This caching distribution leads to the VIP algorithm outperforming the ARC algorithm, achieving both higher throughput and lower delay per packet, at both the StarLight and UCLA servers. At StarLight, the results during the last 30 minutes (in the format ⟨throughput in Gbps, delay in ms⟩) are as follows: nocache ⟨2.7, 155.4⟩; ARC ⟨3.1, 87.5⟩; VIP ⟨3.8, 83.0⟩. At UCLA, the results are as follows: nocache ⟨2.2, 198.3⟩; ARC ⟨2.2, 139.7⟩; VIP ⟨2.6, 138.4⟩. The performance comparison on throughput and delay is shown in Figure 7.

block, and the file name is mapped to an artificial block name by the name mapping table.



(a) Total throughput of the service network test over time.



(b) Average packet delay of the service network test over time.

Figure 7: Total throughput and average packet delay for the service network test over time.

We next conduct a test over the multipath network topology shown in Figure 6. For the multipath topology, caching and forwarding must be jointly designed for optimal performance. In this test, we have request nodes at Northeastern (MGHPCC) and Tennessee Tech, each running one consumer application.

We test three algorithms: round robin paired with ARC, fast route paired with ARC, and VIP. The round robin forwarding strategy chooses the next forwarding hop by sequentially looping through the interfaces in the Forwarding Information Base (FIB) entry corresponding to the requested data object. Under the fast route forwarding strategy, any given forwarder multicasts the first Interest to all possible next hops, observes which next hop replies first, and forwards subsequent Interests to that hop. The forwarder then periodically probes unused next hops, and switches to another next hop if it has strictly lower delay than all others.

As before, we use a fixed window of 8192 packets and a retransmission timeout of 2 seconds at the consumer applications. The UCLA and Tennessee Tech servers can each cache up to 5 data objects. The request process at each request node is the same as in the service network test. For the multi-path test, the VIP algorithm uses a 1 minute time slot length, and a VIP count amplification factor of 2. Note that we use larger VIP time slot length and VIP count amplification factor for this test as compared with the service topology test. This is because the multi-path topology splits requests between two paths, and we need to use sufficiently large VIP count amplification factor and VIP time slot length to ensure VIP queue sizes are reasonably large so that the VIP algorithm can make accurate forwarding and caching choices. We run the test for 1.5 hours and record performance statistics every 30 minutes.

We run the test twice and report the averaged results for the cache hit ratio, total throughput and average packet delay in Figures 8 and 9. A cache hit for a Data Packet (or equivalently data chunk) is recorded when an Interest Packet reaches a node which is not a data source node but which has the Data Packet in its cache. The cache hit ratio is defined here as the total number of cache hits at cache sites UCLA and Tennessee Tech divided by the total number of Interest Packet requests arriving at those cache sites. The total throughput and average packet delay are defined in the same way as for the service topology test, with the consumer nodes now being Northeastern and Tennessee Tech and the producer node being Caltech2. The cache hit ratio during the last 30 minutes is shown in Figure 8. The cache hit ratio of the VIP algorithm is seen to be 10.1% larger than that for fast route paired with ARC, and 11.9% larger than that for round robin paired with ARC. At Northeastern (MGHPCC), the throughput and delay results during the last 30 minutes are as follows (in the format ⟨throughput in Gbps, delay in ms⟩): round robin paired with ARC ⟨3.4, 93.3⟩; fast route paired with ARC ⟨3.1, 88.3⟩; VIP ⟨3.9, 83.4⟩. At Tennessee Tech, the results are as follows: round robin paired with ARC ⟨3.7, 52.0⟩; fast route paired with ARC ⟨3.9, 48.5⟩; VIP ⟨4.0, 38.7⟩. In Figure 9, we show that VIP simultaneously achieves the highest total throughput and the smallest average packet delay.

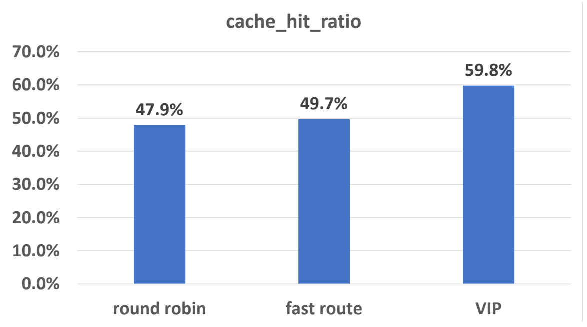


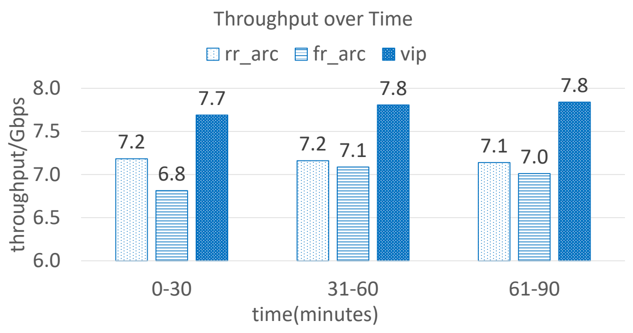
Figure 8: Cache hit ratio in the multi-path test.

6 LESSONS LEARNED

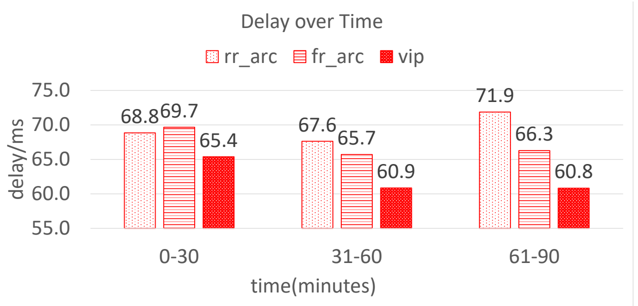
The development and experimentation of N-DISE has pointed to several important lessons, which we detail below.

Challenges in mapping application names to NDN names. As mentioned in Section 2, the naming structure of applications (e.g., CMS) often does not correspond to the underlying hierarchical structure of the data organization. This discrepancy presents difficulties in directly translating application names to NDN names, the latter of which must be designed for efficient network operation. To address this issue, additional mappings must be developed to associate data objects used by workflows (e.g., blocks of CMS data) with data units directly used in network requests (e.g., files of CMS data).

Decoupling control-plane information from forwarding threads. For network control algorithms like VIP, which require exchanging control information with neighboring nodes, transmission and parsing of control packets may interrupt operations of forwarding threads.



(a) Total throughput of the multi-path test over time.



(b) Average packet delay of the multi-path test over time.

Figure 9: Total throughput and average packet delay for the multi-path test over time. "fr" stands for fast route paired with ARC algorithm; "rr" stands for round robin paired with ARC algorithm.

This interruption may increase the queuing delay of Interest and Data Packets in forwarding threads, especially when a node has many neighbors. To avoid this problem, such algorithms can be allocated dedicated threads where control information is generated and parsed.

Impact of middleboxes on NDN-based protocols. Our experimentation over the WAN testbed revealed the disruptive effects of intermediary network devices, i.e. middleboxes [3], on the expected behavior of NDN-based protocols. In particular, we observed several cases where middleboxes were corrupting NDN packets in transit. For instance, we found that some network switches can misidentify the EtherType of an NDN packet as a bare VLAN tag and will thus insert an extra Tag Protocol Identifier before the NDN EtherType. The resulting Ethernet frame is no longer a valid NDN packet, causing the next NDN router on the path to drop all traffic coming from the faulty middlebox. This problem occurred on several paths traversing the StarLight site in Chicago. Checking every switch on these paths to fix the issue at its source was not possible. In order to address the problem, we used the Virtual Extensible LAN (VXLAN) encapsulation protocol to tunnel the NDN packets over UDP/IP. While this method hides NDN packets from the middleboxes and prevents corruption, it also leads to reduced throughput.

System-dependent configuration and tuning. To reduce deployment complexity in our WAN setting, we used Docker containers to package the forwarder, consumer and producer applications. While this is helpful to an extent, such deployments inevitably interact with diverse server equipment, operating systems and network configurations. Therefore, substantial manual effort is still required in configuring drivers, ensuring proper usage, and integrating these software stacks into workflows. Through our experiments, we learned that not all components operate well with each other, so we needed to understand their interactions well and tune them manually, which entailed a large amount of trial-and-error experience.

7 CONCLUSION AND FUTURE WORK

This paper describes the development and prototyping of N-DISE, the first high-performance NDN-based integrated data delivery system for some of the world's largest data- and network-intensive science programs. We have shown that the system successfully combines a NDN naming scheme, new consumer and producer applications, jointly optimal caching and forwarding algorithms, integrated with the high-performance NDN-DPDK forwarder, to achieve leading throughput (around 31 Gbps) and delay performance in real-world WAN settings.

In spite of the impressive performance landmarks established by N-DISE thus far, we believe that significant potential performance gains remain, and can be realized by pursuing important innovations on multiple fronts. We summarize some of these future research directions below.

Integration of NDN with the mainstream data distribution systems at the LHC experiments. Our goal is to integrate NDN with the mainstream data distribution systems at the LHC experiments, starting with CMS. One major component of this project is an NDN-based XRootD Open Storage System (OSS) plugin [6, 5], instrumented with an accelerated packet forwarder, able to deliver data at high speeds. Learning from our previous experience of integrating NDN at the CMS [12], our plan is to further improve the performance capabilities of both the producer and consumer applications described in Section 3 and to embed the latter in an XRootD OSS plugin. The plugin is a C++ dynamic library, loaded at run-time by the framework, that provides file system implementations for different types of storage (CephFS, HDFS, POSIX) by extending a C++ interface that replicates all related system calls (e.g., open, close, read, fstat). Both applications will have a protocol in place to encode and decode every filesystem call into NDN names and attach them to Interest and Data Packets in order to serve the higher level of the XRootD system. Users at the LHC write and run their own scientific applications developed using the CMS Software Components (CMSSW) [4]. Once submitted as jobs in the cluster at one of the partner sites of the experiment, these applications request byte-ranges within the events in HEP data files by calling the local running XRootD service which uses the OSS plugin to natively serve any kind of data.

Multi-threaded forwarding and hierarchical caching with VIP. We plan to carry out a significant extension of the VIP joint caching and forwarding algorithm, addressing two limitations of the current implementation. First, our current implementation of the algorithm is single-threaded, meaning its performance is limited by the processing power of a single CPU core (or hyperthread). In order to achieve higher throughput, we will develop a multi-threaded implementation, which will synergize with the development of multi-threaded consumer and producer applications. This implementation will allow each forwarding thread to maintain its own VIP control information, and will balance traffic flows among forwarder threads based on the VIP control information.

Second, the current implementation can only cache data in DRAM, which severely limits cache spaces we can allocate. As an example of potential cache size requirements, we estimate that a 10 TB cache is needed to cover the most popular 400 blocks composing roughly 80% of CMS requests at the block level over a typical two-month period at Caltech. DRAM alone cannot support such large caches. Therefore, we will extend the algorithm to enable hierarchical caching that incorporates various types of memory and storage, including DRAM and NVMe SSDs. The extended algorithm will jointly optimize caching and forwarding while taking into account the read and write speeds of different hardware elements, as well as the costs of migrating data from one type of cache to another in the hierarchy.

Congestion control based on network feedback. NDN's new features of multipath forwarding and in-network caching can greatly improve effective network throughput for end users. At the same time, they also make consumers' data fetching delay difficult to estimate, which invalidates the well established congestion control solutions which operate over end-to-end connections across a single path. Extensive simulation experimentation has taught us that congestion control approaches purely based on end points measurement do not perform well. Thus, future investigations should look into the direction of making use of network provided feedback. We are currently working on an approach which lets network routers attach their packet queue length information to Data Packets, enabling down stream routers and end consumers to estimate the bottleneck capacity along upstream paths towards data producers, and adjust their Interest Packet forwarding rates accordingly. Preliminary results show that this new approach can effectively control congestion while maintaining high data throughput in the presence of caching and multi-path forwarding [30].

FPGA acceleration. While FPGAs have been used for name lookup in NDN, the challenges faced by NDN-DPDK are unique due to the multi-threaded architecture of the forwarder. For instance, a bottleneck in the forwarder occurs in the thread that handles input. Upon receiving an Interest Packet, the input thread assigns it to a forwarding thread by looking up the hash value of the name prefixes in a data structure called the Name Dispatch Table (NDT). Depending on the size of the NDT and the position of the name component used for indexing, this lookup can become a bottleneck. In addition, if the named identifier has a low length to components ratio, hashing can involve significant delay. To address this bottleneck, in our implementation, we will download the NDT to an FPGA-assisted NIC and use the FPGA to accelerate hashing and lookups in the NDT. Preliminary experimental results on a local testbed show a 4x improvement in throughput over the current

CPU implementation. Moving forward, we will experimentally evaluate FPGA acceleration of the NDN-DPDK forwarder on the WAN testbed.

Integration with genomics workflows. Learning from our experience with CMS, we will pursue integration of our system and protocols into the genomics use case. In preliminary work, we have utilized our software stack to integrate NDN-based operations with genomics workflows. Specifically, we utilized the NDN-DPDK file server to publish genomics data in containers, each of which held a number of files based on their namespace. We then deployed these containers using Kubernetes on Google Cloud to create a flexible data lake that can be extended on demand. On the client side, we used the NDNc consumer to integrate NDN-based data retrieval to contemporary workflows. We plan to further integrate the tools and framework developed as part of this project into genomics workflows. These will include upgraded clients with congestion control algorithms and the ability to utilize multiple clients to retrieve data in parallel.

Data integrity and provenance. Although scientific data may not have strict confidentiality requirements, it is important to ensure the data's authenticity. We will use NDN to implement data origin authentication. To make this feasible for large data volumes, we will adopt an approach based on manifests, where each data producer first computes the hash of each data segment, and then signs collections of hashes (the manifest) instead of the raw data itself. The consumers can retrieve the signed hashes in parallel with fetching the raw data, then use the manifest to verify the data authenticity.

Integration with SENSE [15]. We will integrate the N-DISE system with the SENSE platform, which provides dynamic multi-domain circuits with bandwidth guarantees and allows applications to interact with the network and cooperatively make scheduling and traffic engineering decisions. We will leverage the layer 2 and 3 overlay capability of SENSE as a long-term pathway for NDN to move from prototyping and integration to production use. We will also actively leverage P4 programmability available in the SENSE testbed. Specifically, we will leverage the use of P4, a new source-based routing approach, and the use of the Qualcomm GradientGraph decision support software to provide agile path selection.

ACKNOWLEDGMENTS

This work is supported in part by National Science Foundation grants OAC-2019012, OAC-1659403, OAC-2019163, and OAC-2126148.

DISCLAIMER

Any mention of commercial products or reference to commercial organizations is for information only; it does not imply recommendation or endorsement by NIST, nor does it imply that the products mentioned are necessarily the best available for the purpose.

REFERENCES

[1] The Fast Data Project (FD.io). 2022. Shared memory packet interface (memif) library. (2022). https://s3-docs.fd.io/vpp/22.06/interfacing/libmemif/index.html.

[2] Giorgio Apollinari, O Brüning, Tatsushi Nakamoto, and Lucio Rossi. 2017. High luminosity large hadron collider hl-lhc. *arXiv preprint arXiv:1705.08830.*

[3] B. Carpenter and S. Brim. 2002. Middleboxes: Taxonomy and Issues. RFC 3234. RFC Editor, (Feb. 2002). http://www.rfc-editor.org/rfc/rfc3234.txt.

[4] CERN. 2022. CMS software components. (2022). Retrieved May 25, 2022 from https://cms-sw.github.io/.

[5] CERN. 2022. Xrootd: open file system & open storage system configuration reference. (2022). Retrieved May 25, 2022 from https://xrootd.slac.stanford.edu/doc/dev53/ofs_config.htm.

[6] Alvise Dorigo, P. Elmer, Fabrizio Furano, and A. Hanushevsky. 2005. Xrootd - a highly scalable architecture for data access. *WSEAS Transactions on Computers*, 4, (Apr. 2005), 348–353.

[7] 2018. Earth BioGenome Project Aims to Sequence DNA From All Complex Life. [Online; accessed 20. Jan. 2020]. (Apr. 2018). https://www.ucdavis.edu/news/earth-biogenome-project-aims-sequence-dna-all-complex-life.

[8] 2018. Earth BioGenome Project Aims to Sequence DNA From All Complex Life. [Online; accessed 20. Jan. 2020]. (Apr. 2018). https://www.ucdavis.edu/news/earth-biogenome-project-aims-sequence-dna-all-complex-life.

[9] Chengyu Fan, Susmit Shannigrahi, Steve DiBenedetto, Catherine Olschanowsky, Christos Papadopoulos, and Harvey Newman. 2015. Managing scientific data with named data networking. In *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, 1–7.

[10] 2022. Graphql: a query language for your api. (2022). https://graphql.org/.

[11] Igor V Grigoriev et al. 2011. The genome portal of the department of energy joint genome institute. *Nucleic acids research*, 40, D1, D26–D32.

[12] Cătălin Iordache, Ran Liu, Justas Balcas, Raimondas Širvinskas, Yuanhao Wu, Chengyu Fan, Susmit Shannigrahi, Harvey Newman, and Edmund Yeh. 2020. Named data networking based file access for xrootd. In *EPJ Web of Conferences*. Vol. 245. EDP Sciences, 04018.

[13] Elliot J Lefkowitz, Donald M Dempsey, Robert Curtis Hendrickson, Richard J Orton, Stuart G Siddell, and Donald B Smith. 2017. Virus taxonomy: the database of the international committee on taxonomy of viruses (ictv). *Nucleic Acids Research*, 46, D1, D708–D717.

[14] Nimrod Megiddo and Dharmendra S. Modha. 2003. ARC: a Self-Tuning, low overhead replacement cache. In *2nd USENIX Conference on File and Storage Technologies (FAST 03)*. USENIX Association, San Francisco, CA, (Mar. 2003). https://www.usenix.org/conference/fast-03/arc-self-tuning-low-overhead-replacement-cache.

[15] Inder Monga. 2016. *SENSE Project.* https://www.es.net/assets/pubs_presos/SENSE-Thomas-20160217-on-Web.pdf.

[16] N-DISE. 2022. NDNc: a lightweight integration of ndn-cxx with ndn-dpdk to achieve high throughput performance in scientific applications. (2022). https://github.com/cmscaltech/sandie-ndn/tree/master/NDNc.

[17] NIST. 2022. Hardware Compatible with NDN-DPDK. (2022). Retrieved Aug. 26, 2022 from https://github.com/usnistgov/ndn-dpdk/blob/5bc92be0f3706142806dd6a52c6ac7ec042c2c8c/docs/hardware.md.

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

[19] Giovanni Petrucciani, Andrea Rizzi, Carl Vuosalo, CMS Collaboration, et al. 2015. Mini-AOD: a new analysis data format for CMS. In *Journal of Physics: Conference Series* number 7. Vol. 664. IOP Publishing, 072052.

[20] DPDK Project. 2022. DPDK supported hardware. (2022). Retrieved Jan. 28, 2022 from https://core.dpdk.org/supported.

[21] The Named Data Networking Project. 2022. ndn-cxx: NDN C++ library with eXperimental eXtensions. (2022). https://named-data.net/doc/ndn-cxx/current/.

[22] The Hutch Report. 2020. Genomics Report. [Online; accessed 20. Jan. 2020]. (Jan. 2020). https://www.preoncapital.com/wp-content/uploads/2018/06/THR_Genomics.pdf.

[23] Andrea Rizzi, Giovanni Petrucciani, and Marco Peruzzi. 2019. A further reduction in CMS event data for analysis: the nanoaod format. In *EPJ Web of Conferences*. Vol. 214. EDP Sciences, 06021.

[24] Lucio Rossi and Oliver Brüning. 2012. High luminosity large hadron collider: A description for the European strategy preparatory group. Tech. rep.

[25] Susmit Shannigrahi, Chengyu Fan, and Christos Papadopoulos. 2017. 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*, 54–65.

[26] Susmit Shannigrahi, Chengyu Fan, Christos Papadopoulos, and Alex Feltus. 2018. Ndn-sci for managing large scale genomics data. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*, 204–205.

[27] Susmit Shannigrahi, Chengyu Fan, and Craig Partridge. 2020. What's in a name? naming big science data in named data networking. In *Proceedings of the 7th ACM Conference on Information-Centric Networking* (ICN '20). Association for Computing Machinery, Virtual Event, Canada, 12–23. ISBN: 9781450380409. DOI: 10.1145/3405656.3418717.

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

[29] Junxiao Shi, Davide Pesavento, and Lotfi Benmohamed. 2020. NDN-DPDK: NDN forwarding at 100 Gbps on commodity hardware. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 30–40.

[30] Sichen Song and Lixia Zhang. 2022. Effective NDN Congestion Control Based on Queue Size Feedback. In *Proceedings of the 9th ACM Conference on Information-Centric Networking*.

[31] Dongmei Tian, Pei Wang, Bixia Tang, Xufei Teng, Cuiping Li, Xiaonan Liu, Dong Zou, Shuhui Song, and Zhang Zhang. 2019. Gwas atlas: a curated resource of genome-wide variant-trait associations in plants and animals. *Nucleic Acids Research*, 48, D1, D927–D932.

[32] Elisabeth Veeckman, Tom Ruttink, and Klaas Vandepoele. 2016. Are we there yet? reliably estimating the completeness of plant genome sequences. *The Plant Cell*, 28, 8, 1759–1768.

[33] Björn C Willige, Joanne Chory, and Marco Bürger. 2018. Next generation of plant-associated bacterial genome data. *Cell host & microbe*, 24, 1, 10–11.

[34] Edmund Yeh, Tracey Ho, Ying Cui, Michael Burd, Ran Liu, and Derek Leong. 2014. Vip: a framework for joint dynamic forwarding and caching in named data networks. In *Proceedings of the 1st ACM Conference on Information-Centric Networking*, 117–126.