Designing Hydra with Centralized versus Decentralized Control: A Comparative Study

Siqi Liu
UCLA
siqi.liu@ucla.edu

Varun Patil
UCLA
varunpatil@cs.ucla.edu

Tianyuan Yu
UCLA
tianyuan@cs.ucla.edu

Alexander Afanasyev
Florida International University
aa@cs.fiu.edu

Frank Alex Feltus
Clemson University
ffeltus@clemson.edu

Susmit Shannigrahi
Tennessee Technological University
sshannigrahi@tntech.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

ABSTRACT

Today's networked and distributed applications, by and large, rely on cloud services. However, solely cloud-based services are not the ideal solution for all use cases, in particular, the case of high volume data sharing in scientific computing whose cloud usage costs could be prohibitively high. Thus we take on a task of building a distributed, federated data repository, dubbed *Hydra*, for sharing large volume scientific data. In this paper, we compare two design choices: designing Hydra over TCP/IP with a centralized controller, and designing Hydra over Named Data Network (NDN) to enable distributed control. Our study shows that (i) building Hydra over TCP/IP with a central controller offers a simple, straightforward design; (ii) however, the controller necessarily needs to be replicated for scalability and reliability, and cloud CDN is needed to scale data delivery, both bringing additional complexity into the overall design; and (iii) building Hydra over NDN *automatically* offers scalable and efficient data dissemination at volume, as well as enables distributed control with high resiliency.

CCS CONCEPTS

• **Networks** → **Network services**; **Network design principles**.

ACM Reference Format:
Siqi Liu, Varun Patil, Tianyuan Yu, Alexander Afanasyev, Frank Alex Feltus, Susmit Shannigrahi, and Lixia Zhang. 2021. Designing Hydra with Centralized versus Decentralized Control: A Comparative Study. In *Interdisciplinary Workshop on (de)Centralization in the Internet (IWCI '21), December 7, 2021, Virtual Event, Germany*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3488663.3493690

IWCI '21, December 7, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-9138-2/21/12…$15.00
https://doi.org/10.1145/3488663.3493690

1 INTRODUCTION

21st century scientific experiments accumulate large amounts of data that are often generated and analyzed in geographically distributed facilities. Existing storage solutions utilize either centralized repositories in the cloud, large institutional repositories, or a hybrid approach that makes use of both types of storage. These highly centralized approaches not only require huge storage with the associated high costs, but also constrain what researchers may be allowed to publish, slowing down data publication and research progress. Additionally, once data are published to these centralized repositories, moving data in and out of proprietary cloud offerings can be difficult and expensive. Further, researchers can lose precise control of the datasets, and any update or deletion of the datasets needs to go through the central repository control which can be a slow process.

A more sustainable approach is to allow individual scientists and small consortia to publish data to a distributed storage system they control. Thanks to ever increasing volume and decreasing cost of storage technology, many scientists have access to local file servers or institutional repositories. We conjecture that an easy-to-deploy and easy-to-use storage framework that utilizes scattered storage can help fulfill modern research needs. Such a system will allow scientists to not only retain control of the datasets but also publish datasets that are not accepted by central repositories, or are too expensive to host in commercially available cloud storage services at this time.

In this work, we discuss the design of Hydra, a federated file repository to meet the need for storing and managing a large number of files for distributed science use cases. We examine two different design choices: an IP-based design that utilizes a central server in the cloud, and a distributed design over Named Data Networking (NDN) [13]. In making our design choice in building Hydra, we articulate the insights into the most critical factors that enable such decentralized designs in today's pervasively consolidated deployment environments: distributed control, security management, and dataset synchronization.

In the rest of the paper, we first describe the needed functionality of Hydra, then a design sketch of implementing Hydra functionality over TCP/IP using a centralized server, followed by Hydra design

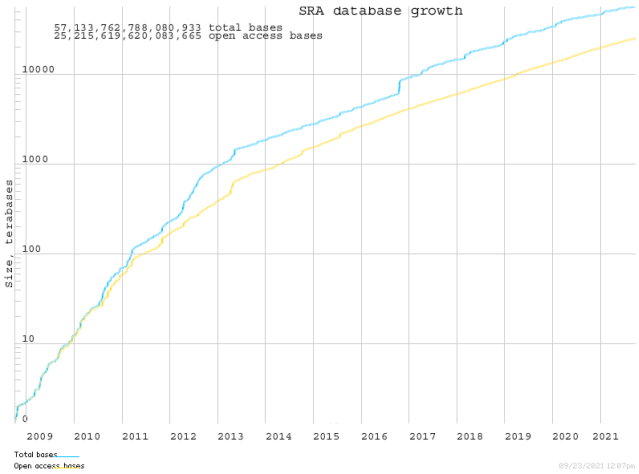

Figure 1: National Center for Biotechnology Information's Sequence Read Archive [9]

over NDN. Lastly, we will discuss the key factors in the formation of the solution and their impact on network centralization today.

2 A FEDERATED FILE REPOSITORY

To obtain the storage capacity mentioned above, scientists and small consortia need a distributed storage system that utilizes scattered storage resources and provides the same interface as centralized repositories. A federated file repository is designed to satisfy this need. This section first elaborates on the consumer use case in genomics data, then generally explains the advantages and interface requirements of a federated file repository.

2.1 Case Study: Genomics Data

In 2021, genomics data accumulation is exploding in centralized data repositories. As shown in Figure 1, a single database at the National Center for Biotechnology Information's Sequence Read Archive (NCBI-SRA) [9] has accumulated over 57 petabytes of DNA sequence data, representing over 8.8 million experiments that can be mined for biological patterns impacting biomedicine to agriculture. There are hundreds of other data repositories large (e.g. NASA Genelab: genelab.nasa.gov, UK/EU ENSEMBL: ensembl.org) and small (e.g. Tripal sites: tripal.info), where data is difficult to find due to vagaries in finding the right IP address, file path, and the appropriate data transfer protocols supported by the data host. Further, individual research groups store DNA sequence data on local file systems that are invisible to the outside world. In order to extract new knowledge from all the available data, we need a decentralized storage system that allows distributed publications from dispersed data stores, as well as retrieval into data-intensive compute workflows. Therefore, a federated file repository is a great fit for this use case.

2.2 Why Using A Federated File Repository

The simplest file storage system is a single file server, storing both the metadata and the actual contents of all files in the server. All users can access files in the server over the network by directly connecting to the server using traditional point-to-point protocols such as FTP.

A federated file repository, on the other hand, is built on a federation of multiple geographically distributed storage servers on a network. They are collectively responsible for storing all published files, with each file replicated for reliability. The whole repository must be secured, so that only authorized users can perform file insertion and deletion operations, and access the contents of available files.

Compared to such a file server, a federated file repository has several advantages:

- Resiliency: files can be replicated in multiple servers, a file may be persistent and available in the events of failures of individual nodes or network partitions.
- Scalability: Since storage for the repository is distributed in multiple servers, each server only needs to provide small storage space for the files, as opposed to having a single large storage node. Scaling up storage by adding more servers is generally easier than scaling up against hardware constraints.
- High file read throughput: Since files in a federated repository are distributed across multiple nodes and disks, the read and write requests can be handled by multiple servers in the federation, which drastically increases the aggregate transfer throughput.
- Utilizing scattered storage: As storage gets cheaper and achieves higher volume with time, many scientists have access to an increasing amount of institutional storage. However, these community-owned resources are difficult to utilize due to the lack of robust protocols that can consolidate them into a coherent file repository. A federated file repository can reduce the cost of cloud storage, improve the utilization of existing infrastructure, and reduce the bottleneck of centralized storage.

These advantages are critical for file repositories based on the community as well as the cloud providers. This design allows a shared file repository in a community, in which members of a community can pool a portion of unused storage space from their individual owners for shared benefits, federated file repository reduces the running costs of the repository. We note that, even for cloud providers, a federated file repository over virtualized servers is key for the high resiliency and availability of the storage services. In fact this economy of scale is a key driving force for the prosperity of the cloud.

2.3 Basic Functions

File repositories provide two basic functions: read and write operations. Users can access the published data in the federated file repository with a read command. Authorized users can manage files to the repository with write commands, which consist of two key actions:

- the insertion of files into the repository
- the deletion of files from the repository

These changes would then be accessible using read operations to other authorized users of the repository. These actions are used as

building blocks of more complex interactions of users with the file repository such as file modification and renaming.

2.4 Security Considerations

All actions above require authentication and authorization to ensure that only legitimate users can access or modify the files. The file repository should validate the identity of the user, then satisfy the commands based on the access control data specified for the target file.

Each operation has its own access control; for a file, a user who is not the owner of the file may not be able to modify or delete that file. For simplicity, we assume here that access control rules are predefined and can be checked for a given pair of user and file after user authentication. Often, federated file repositories do not provide file ownership by allowing the owner to choose the file storage location. Instead, the file owner should rely on the access control on the file access commands or apply an additional layer of encryption on the stored file.

3 A DESIGN OVER TCP/IP

Several existing data repository designs provide the functionality of a federated file repository over today's TCP/IP protocol stack. In this section, we explore such a design and how it would perform the functions of federated file repository. We also take a look at contemporary systems that are designed for similar functionality.

3.1 Design Model

The most straightforward idea for building a distributed file repository over TCP/IP is to control the repository from a centralized server, or controller, hosted in the cloud. The controller's basic job includes both checking the authenticity of users and file servers, and orchestrating file replication locations. These are relatively easy tasks for a central controller, which can perform password-based authentication for users via secure TLS connections, and control the storage nodes in the repository via a leader-follower model.

The basic insertion and retrieval of files in the file repository can be achieved with a centralized controller by centrally maintaining a database of all files stored in the repository along with their locations. Since each file in the repository must be replicated across multiple storage nodes for resiliency, the centralized controller can directly make decisions on which files must be stored and replicated on which nodes. These decisions may be performed on the basis of one or more factors such as the size of the file, availability of storage space on a particular node, the type of data, grouping semantically similar content, etc. The controller may also decide to shard one file across multiple nodes to improve read performance, or if the file is too large to fit on one node.

For write operations on the file repository, the client contacts the controller to retrieve a write lock and the location of the storage node to which a data block must be sent. The client then establishes a point-to-point link with the storage server and writes the file to the server. If the controller desires to store the file at multiple locations, the client must either upload the data block to the other servers again, or the controller must orchestrate the transfer from the upload server to the others.

For read operations, the client must first connect to the controller to learn the location of the desired file or data block. Since the controller has a database of all files and their locations, it can then redirect the client to one or more storage nodes containing the file. The client can then retrieve the file from the storage node over a TCP connection. Therefore, locating and retrieving content from the users' nearest (or preferred) node requires the central controller to measure or record the performance parameters of each node as well as users' relative preferences. When a user wants to upload or download data, the central controller will coordinate the "best" server for the operation.

For security, a data repository design requires mutual authentication between the repository and the user. Mutual authentication establishes the identity of the repository and the authenticity of the data to a retrieving user and allows the repository to authenticate the user and enforce access control. The individual storage and controller servers must be configured with certificates that are either explicitly trusted by all clients using a common trust anchor, or the system must use existing public key infrastructure for authentication. For authentication of users, in addition to using passwords, the system may use a Single Sign-On approach. We further elaborate on the differences in Section 5.3.

A single central controller, however, creates a single point of failure in the system. Such a system can also be disrupted by network partitions between the controller and the data storage nodes. As a result, more resilient designs of distributed file repositories require the usage of several centralized controllers that are geographically distributed and perform write operations on the system through the usage of locking mechanisms (BigTable [3]) or assume immutability of data (Haystack [1]). Such a design with multiple centralized controllers adds a layer of complexity over the simpler design described earlier, since achieving synchronization between the distributed controllers is not straightforward in the presence of losses and network partitions.

The current Internet topological structures mean that the central controller and even storage nodes need to be placed in the cloud for deployment simplicity. First, the server needs to be reachable from any location. This means that the server needs to be placed in a server-hosting platform, instead of network edge locations that are behind a Network Address Translation (NAT) router or assigned with dynamic IP addresses. Then, the server needs security protection from the cloud. The cloud, with its Content Delivery Network (CDN) facilities, is able to provide mitigation against Distributed Denial of Service (DDoS) attacks by decrypting and filtering traffic. Such services are becoming crucial as individually deployed servers do not have DDoS mitigation capabilities today. Lastly, the data dissemination over TCP/IP unicast is not scalable, and therefore also needs CDNs to scale services. As elaborated in §5.1 and §5.3, if multiple users are fetching the same file, the storage nodes needs to re-build and re-encrypt response, and therefore send out multiple copies of the same data. As the number of users increases, the nodes may be overloaded by encrypting and service large pieces of data. These considerations contribute to the increasing centralization of the Internet.

3.2 Related Work

Many distributed systems over IP follow the leader-follower model as described above. The Hadoop Distributed File System (HDFS) [12] uses a single controller, called NameNode, for orchestrating the file location to DataNodes, the storage nodes, in order to act as a distributed file system, providing partial functionality of the federated file repository.

More resilient designs of file repositories require the usage of several controllers or even multiple controllers with each playing different roles. Some of these distributed systems perform write operations on the system through the usage of locking mechanisms. BigTable [3], a storage database for structural data by Google, for example, uses a hierarchy of controllers called tablets for storage and control of meta-information. It uses a lock service called Chubby [2] to achieve consistency on the meta-data. Haystack [1], a photo storage service for Facebook, uses a set of servers to form the Haystack directory for orchestrating the location of the photo data.

4 A DESIGN OVER NDN

To eliminate the single point of failure, we explore the realization of a federated file repository over NDN to design a distributed control architecture. As described next, this design has no central controllers; instead, every node performs the same function of storing files and supplying them to authenticated users, forming a peer-to-peer model.

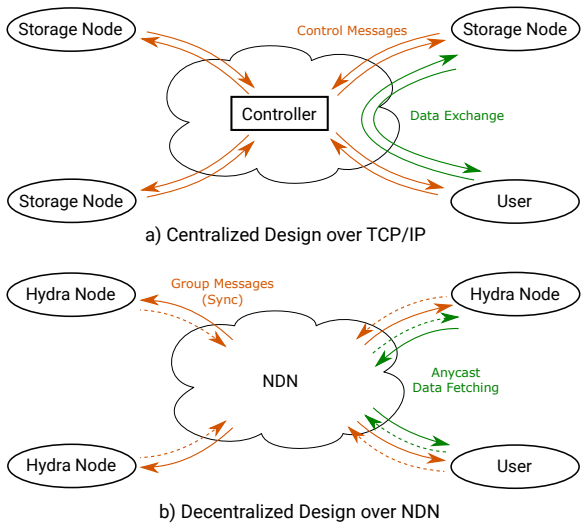

Figure 2: Comparison of the Hydra designs over TCP/IP and NDN

4.1 Named Data Networking

In an NDN network, each data packet is uniquely identified by its name. Unlike the TCP/IP model which pushes packets to destinations, NDN lets consumers fetch data by names from the network. Thus, data flow in NDN is driven by the consumer rather than the producer of the data.

Each packet in NDN is also cryptographically signed by the producer. This allows the network to cache packets during their transit. If multiple users request the same data packets, only one request is forwarded to the producer, and returned data packets are forwarded to all requesters, enabling multicast data delivery from the producer to all consumers. Since packets are individually signed, all consumers can verify their authenticity.

To fetch a data packet produced by the producer by sending an Interest, a consumer first needs to know that the data packet was produced, and its identifying name. Dataset Synchronization, or Sync [8], plays the role of the transport service for applications running over NDN. Whenever a producer produces new data, Sync propagates this change over the network and informs all communicating parties in the same application (the "Sync group") about the name of the newly produced data. Consumers can then fetch this data if they desire.

Sync enables applications running over NDN to easily synchronize application state among multiple distributed nodes. Whenever the application state changes, the application can produce event data informing others about the update. This event data will be fetched by all the other participants in the Sync group for whom the update is relevant, and hence learn about the state change at the producer. Further, an NDN Sync protocol, such as SVS [7], is inherently resilient to losses, varying network delays, and intermittent connectivities, make it a good fit for distributed systems running over wide area networks.

4.2 Hydra: A Distributed Control Design

Hydra, a design of distributed federated file repository, can be naturally implemented over NDN. Hydra runs over a collection of file servers that share the same role and responsibility in the system. Each Hydra node in the distributed design stores a portion of the files. All the Hydra nodes collectively maintain a "global view" of the system, which contains a complete list of all the files in the system, and for each file F, a list of nodes that holds a copy of F. We call this "global view" the *metadata* of the system, and every Hydra node keeps a consistent copy. Hydra uses the Sync protocol SVS to keep the metadata up to date at all the nodes. Whenever the state at a Hydra node N is updated by a write operation, N publishes this change event over Sync, which is then propagated to all other nodes. Sync uses multicast to ensure efficient dissemination of the metadata to all servers.

Users can make read/write requests to Hydra using NDN Interest packets. Since NDN uses name-based network layer forwarding, users does not need to know the name or address of any Hydra node. Instead, users Interest packets reach Hydra via anycast, so the request is forwarded to the "closest" Hydra node according to the routing system. With the global view of Hydra at every node, the nearest Hydra node can then directly serve the request according to the meta-data that it knows.

For read operations, the file requests can be delivered over NDN with the benefit of multipath forwarding. Furthermore, due to NDN Interest aggregation, the replies are multicast if multiple users request the same piece of data, or fetched from in-network caching, drastically reducing network traffic.

Hydra uses an NDN-based security framework [16] to ensure authentication and authorization. Instead of securing the client-server channel like TLS, NDN secures the Data packet directly. Each

Data packet containing the file chunk will be signed by the Hydra node to ensure authenticity and integrity. This design allows the same file chunk to be distributed securely to multiple clients using multicast and in-network caching.

To provide authentication, the nodes and clients need to be bootstrapped with their identities. Hydra uses a Network Operating Center (NOC) as a trust anchor. Even though this server is a single node, Hydra still continues to run without it, except that no new nodes can be added. The end users can also be assigned certificates by the NOC, through the out-of-band identity challenges that utilize existing authentication.

5 DISCUSSION

The approaches presented in §3 and §4 are radically different. With different design decisions, there are associated differences in costs and benefits. For both approaches, the design choices have roots in the underlying network and transport protocols.

5.1 Data Transport

The leader-follower file repository solution design is based on TCP/IP, which provides point-to-point data transportation. Therefore, for insertion and retrieval of the file, the user must first find the server's IP address to contact. In the leader-follower design, this connection is established with the help of the controller, which provides the user the IP address of the server. Hydra, on the other hand, uses a data-centric network transport. Even though Hydra is made of a collection of file servers, users are not directly interfacing with those servers; they are instead querying for files from the network by using names. Therefore, users directly request desired files by putting their names into NDN Interest packets, which are anycast to the Hydra system and reach nearest Hydra nodes which can then direct serve the data if they have the files locally, or otherwise redirect users to other Hydra nodes with the requested file. The rendezvous between a user's request and the data is achieved through name-based network forwarding in the NDN network.

NDN network's name-based forwarding functions could be viewed as more complex than the model of traditional IP based forwarding with a single controller, but it brings with great benefits. Exploiting the named-based network transport, the routers along the transport path are able to provide intelligent forwarding of the Interests. For example, the routers can provide multipath forwarding for the Data to fetch the replicated Data from all available Hydra nodes. With the stateful network transport and in-network caching of NDN, the routers also automatically perform multicast delivery, as well as CDN functions for other users that may fetch the same data packet in near future. These networking features are provided in NDN network service abstraction that is transparent from users, providing them the benefit automatically, without demanding any additional setup.

We note that the features of multicast, anycast, and multipath forwarding have also been proposed for IP networks in the past, but except anycast, the others have faced deployment limitations that prevent their wide usage. IP anycast is usable on the public internet; however, it faces challenges in choosing the closest server in the public Internet because of the BGP routing design and therefore requires careful traffic engineering[14]. DNS-based "anycast", in which the DNS name server chooses the closest server for clients based on the client's address, is widely used by CDNs. However, the name server's role is congruent to the file repository controllers in the file location orchestration. IP multicast is not supported in the public Internet because of its complexity, which in turn is due to its design paradigm differing from IP protocol in fundamental ways: the former is stateless, while the latter requires stateful forwarding. Finally, IP-based multipath forwarding is also supported in a preliminary phase, with custom protocols like multipath TCP[11] that require additional kernel and application support at the host.

Above all, all the TCP/IP-based solutions are node-centric, that is one node send packets to another. Therefore, a server needs to resend the same data for each of the users who request the same data, putting traffic burden on the server and the network. Interestingly, serving popular datasets over IP increases the load on the servers while serving popular datasets over NDN makes no impact on the servers (the same load as serving a single user). These additional features greatly increase the efficiency and capability of the data transport for the file repositories.

5.2 Control Architecture

In the leader-follower solution design, the file location is determined by the controller. Therefore, only the controller needs to keep the metadata for the system. On the other hand, Hydra shares the metadata with all peers. This means that the update to metadata needs to be synchronized among all peers. Hydra use NDN Sync protocols such as SVS for metadata updates. The design of the metadata update means that for each change, communication is needed for all the peers, increasing the traffic flow. Therefore Hydra's distributed design has a higher cost than the central controller design when measured by the total number packets that need to be exchanges among all the nodes.

With the associated cost, however, the Hydra design enjoys the benefit of high resiliency. Keeping meta-data at each node removes the single point of failure of the control server. Since each Hydra node now has all the metadata including the locations of each file, a file will become unavailable for users only if every server that stores the file has failed or cannot be reached due to network partitions. As a result, the distributed solution can sustain a lot more node failures and network instability than the centralized solution.

Additionally, Hydra's design has system scalability built in. The distribution of the control allows the file orchestration task to be spread to multiple nodes, instead of having all requests processed by the single controller. Therefore, Hydra's solution is able to scale for a much higher request throughput.

For the above reasons, existing TCP/IP-based solutions also try to create a decentralized design that uses a small number of controllers instead of a single central controller. However, in contrast to NDN, where Sync can directly synchronize the state between the controllers, TCP/IP based solutions must implement such functionality for distributed consensus at the application layer, using mechanisms such as $n \times n$ TCP connections, which will typically have very high overhead. This is generally achieved using database services, adding an additional layer of complexity to the controller.

5.3 Security

Nowadays TCP/IP solutions such as BigTable [3] usually leverage the cloud infrastructure for storage resource scaling, and the TLS connections between application users and the cloud provider the security support. TLS binds the communication security with the communication endpoint, therefore individual application users have to trust the cloud endpoint who also provides the resources. The data security relies on users' implicit trust in the cloud providers (or their Certificate Authorities) for TLS certificates, and the correct operations of the cloud-managed servers. It forms a single point of failure problem, where if the servers in the cloud are compromised, the stored data are no longer secured. This is because the *trust anchor* of the data security is in the cloud providers, instead of users' hands. Even worse, recent data breach incidents in clouds [4–6, 10] indicate that trust anchors may not always be as trustworthy as we expected.

Distributed control solutions built on NDN can take a fundamentally different approach. NDN's data-centric security design [16] empowers users to build decentralized trust management solutions by establishing trust over name semantics. Data repository users can establish their own local trust anchors (e.g., self-signed certificates) in Hydra NOC, therefore build and control their own zone of trust, instead of outsourcing the trust management to a third party like the cloud.

This distributed solution secures the data directly. Each individual Data packet is signed by the producer, and the packet need not be signed again when re-transmitting to a different destination. As a result, the Data can be cached and re-used in the in-network cache, maintaining the security property for the more complex actions. This property also applies to NDN Data storage like Hydra. Although individual Hydra nodes can be compromised, the data authenticity and confidentiality of the stored files still hold, since each file is directly signed by the producer's private key, and encrypted by the producer's corresponding content encryption key. Any file receiver can verify the data authenticity by validating the signature by its producer's public key and decrypt the content using the content decryption key from the producer. Named-based Access Control [15] can facilitate this process with an efficient yet simple design that leverages NDN's named, secured data.

6 CONCLUSION

Although the IP network was designed to support distributed systems initially, its lack of basic supports for distributed *applications*, in particular, security, distributed dataset synchronization, and scalable data dissemination, have resulted in its deployment evolving towards an increasingly centralized structure over time. It is undeniable that the economy of scale fueled this evolution, but we also believe that IP's point-to-point communication model, and the consequential client-server application model built upon it, provided a convenient technology path for the market to quickly grab the opportunity of controlling the server end in the client-server model, turning end users, the clients, to revenue-generating "eyeballs". Users do not have choices since they have neither an easy way to identify each other, an easy way to reach each other, nor an easy way to authenticate each other, as needed to enable secure peer-to-peer communications.

To steer the Internet away from centralization, we recognize that commercial interests and resource optimization can be important blockades that may need legislative actions to counter. However, we also believe that viable technical solutions can play a fundamental role in the overall solution space. TCP/IP's node-centric model and lack of build-in security make decentralized solutions complex to build and maintain. On the other hand, NDN has the potential to address these challenges with its new data-centric networking model. By naming and securing data directly, NDN has in-network caching and multicast data delivery built-in to support effective data dissemination. By fetching semantically named, secured data at the network layer, NDN enables distributed parties to rendezvous *inside the network*, removing TCP/IP's reliance on central servers to rendezvous, a fundamental technical barrier to decentralization. The centerpiece in the NDN design is a semantic namespace, and we believe that today's DNS namespace can serve as a great starting point.

ACKNOWLEDGMENTS

This work is partially supported by the National Science Foundation under awards 1659300, 1719403, 2019012, 2019085, 2019163, and 2126148.

REFERENCES

[1] Doug Beaver, Sanjeev Kumar, Harry C. Li, Jason Sobel, and Peter Vajgel. 2010. Finding a Needle in Haystack: Facebook's Photo Storage. In *9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 10)*. USENIX Association, Vancouver, BC. https://www.usenix.org/conference/osdi10/finding-needle-haystack-facebooks-photo-storage

[2] Mike Burrows. 2006. The Chubby lock service for loosely-coupled distributed systems. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[3] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. 2006. Bigtable: A Distributed Storage System for Structured Data. In *7th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. 205–218.

[4] Forbes. 2020. Confirmed: 2 Billion Records Exposed In Massive Smart Home Device Breach. https://www.forbes.com/sites/daveywinder/2019/07/02/confirmed-2-billion-records-exposed-in-massive-smart-home-device-breach. Accessed: 2021-09-24.

[5] Health IT Security. 2021. CVS Health Faces Data Breach,1B Search Records Exposed. https://healthitsecurity.com/news/cvs-health-faces-data-breach1b-search-records-exposed. Accessed: 2021-09-24.

[6] HIPAA Journal. 2021. Cancer Treatment Centers of America Announces 105,000-Record Data Breach. https://www.hipaajournal.com/cancer-treatment-centers-of-america-announces-105000-record-data-breach/. Accessed: 2021-09-24.

[7] Philipp Moll, Varun Patil, Nishant Sabharwal, and Lixia Zhang. 2021. *A Brief Introduction to State Vector Sync.* Technical Report NDN-0073, Revision 2. Named Data Networking. 1–4 pages.

[8] Philipp Moll, Wentao Shang, Yingdi Yu, Alexander Afanasyev, and Lixia Zhang. 2021. *A Survey of Distributed Dataset Synchronization in Named Data Networking.* Technical Report NDN-0053, Revision 2. Named Data Networking. 1–18 pages.

[9] National Library of Medicine (US), National Center for Biotechnology Information. 2008. Sequence Read Archive. https://trace.ncbi.nlm.nih.gov/Traces/sra. Accessed: 2021-11-01.

[10] Observer. 2021. Parler Was Hacked on WordPress, The Internet's Biggest Platform. Is Everyone At Risk? https://observer.com/2021/01/how-parler-was-hacked-on-wordpress-risk/. Accessed: 2021-09-24.

[11] Costin Raiciu, Christoph Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. 2012. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation* (San Jose, CA) *(NSDI'12)*. USENIX Association, USA, 29.

[12] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. 2010. The Hadoop Distributed File System. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST)*. 1–10. https://doi.org/10.1109/MSST.2010.5496972

[13] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, kc claffy, Patrick Crowley, Christos Papadopoulos, Lan Wang, and Beichuan Zhang. 2014. Named Data Networking. *ACM SIGCOMM Computer Communication Review (CCR)* 44, 3 (July 2014), 66–73.

[14] Xiao Zhang, Tanmoy Sen, Zheyuan Zhang, Tim April, Balakrishnan Chandrasekaran, David Choffnes, Bruce M. Maggs, Haiying Shen, Ramesh K. Sitaraman, and Xiaowei Yang. 2021. AnyOpt: Predicting and Optimizing IP Anycast Performance. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference* (Virtual Event, USA) *(SIGCOMM '21)*. Association for Computing Machinery, New York, NY, USA, 447–462. https://doi.org/10.1145/3452296.3472935

[15] Zhiyi Zhang, Yingdi Yu, Sanjeev Kaushik, Alex Afanasyev, and Lixia Zhang. 2018. NAC: Automating Access Control via Named Data. 626–633. https://doi.org/10.1109/MILCOM.2018.8599774

[16] Zhiyi Zhang, Yingdi Yu, Haitao Zhang, Eric Newberry, Spyridon Mastorakis, Yanbiao Li, Alexander Afanasyev, and Lixia Zhang. 2018. An Overview of Security Support in Named Data Networking. *IEEE Communications Magazine* 56, 11 (November 2018), 62–68. https://doi.org/10.1109/MCOM.2018.1701147