Published in final edited form as:

Proc IEEE Int Conf Cloud Netw. 2020 November ; 2021: . doi:10.1109/cloudnet51028.2020.9335806.

Named Data Networking for Content Delivery Network Workflows

Rama Krishna Thelagathoti*, Spyridon Mastorakis*, Anant Shah†, Harkeerat Bedi†, Susmit Shannigrahi‡

*Computer Science Department, University of Nebraska, Omaha, USA

†Verizon Digital Media, USA

‡Computer Science Department, Tennessee Tech University, Cookeville, TN

Abstract

In this work we investigate Named Data Networking's (NDN's) architectural properties and features, such as content caching and intelligent packet forwarding, in the context of Content Delivery Network (CDN) workflows. More specifically, we evaluate NDN's properties for PoP (Point of Presence) to PoP and PoP to device connectivity. We use the Apache Traffic Server (ATS) platform to create a CDN-like caching hierarchy in order to compare NDN with HTTP-based content delivery. Overall, our work demonstrates that several properties inherent to NDN can benefit content providers and users alike through in-network caching of content, fast retransmission, and stateful hop-by-hop packet forwarding. Our experimental results demonstrate that HTTP delivers content faster under stable conditions due to a mature software stack. However, NDN performs better in the presence of packet loss, even for a loss rate as low as 0.1%, due to packet-level caching in the network and fast retransmissions from close upstreams. We further show that the Time To First Byte (TTFB) in NDN is consistently lower than HTTP (~ 100ms in HTTP vs. ~ 50ms in NDN), a vital requirement for CDNs. Unlike HTTP, NDN also supports transparent failover to another upstream when a failure occurs in the network. Finally, we present implementation-agnostic (implementation choices can be Software Defined Networking, Information Centric Networking, or something else) network properties that can benefit CDN workflows.

I. Introduction

The primary goal of the current Internet is content distribution. Content Delivery Networks (CDNs) serve a large number of users, predominately over HTTP(s). HTTP over TCP/IP has worked well so far, though not without complex configuration and extreme engineering solutions.

Named Data Networking (NDN) [1] is a future Internet architecture that focuses on content distribution and utilizes content names for most in-network operations such as packet forwarding and content caching. The synergy between CDNs and the NDN architecture is that both aim to optimize content delivery. Further, CDNs and NDN both utilize similar

Author Manuscript

concepts - CDNs utilize named content in the form of HTTP URLs while NDN uses named content for data delivery and in-network operations such as routing and forwarding. By aligning the underlying network to CDNs' requirements through the use of content names, we can expect several benefits - less complicated infrastructure and software stack, optimized data delivery, and automatic path selection and failure recovery at the network layer. We envision that NDN can be deployed as an "overlay" on existing CDN infrastructures between Points-of-Presence (PoPs) that will not require any infrastructure change. After the benefits of a name-based network are well understood and evaluated at scale, we can consider native NDN deployments.

Author Manuscript

This work attempts to apply NDN to a CDN workflow, specifically for inter-PoP communication scenarios. To better understand NDN's architectural properties, we also look at content distribution over HTTP using Apache Traffic Server (ATS), a contemporary open-source and high performance caching software that can act both as reverse and forward proxy. For content distribution over NDN, we use the standard NDN software such as ndn-tools (producer/consumer tools) and NDN forwarding daemon (NFD) [2]. Our goal is to evaluate NDN's performance based on salient features of a CDN topology.

Author Manuscript

Our contribution in this paper is twofold. First, we analyze the content distribution properties of NDN and HTTP on a testbed of virtual machines. These comparisons demonstrate the scenarios where NDN performs better than HTTP. The second contribution is to identify and quantify the benefits that the NDN architecture can bring to a contemporary CDN workflow. We show that NDN's architectural artifacts such as seamless failover, the ability to choose the "best upstream", and partial data retrieval can speed up content delivery, quickly recover from loss, and lower the overall resource utilization.

Motivation

Author Manuscript

CDNs aim to deliver content in a location-transparent way to the user. However, the current TCP/IP paradigm is end-to-end. Creating a location-transparent application on top of an end-to-end protocol stack requires working around several end-to-end artifacts. For example, TCP needs to establish end-to-end connections even when the transfer size is small. Upstream hosts (caches and origin servers) need to be enumerated and monitored continuously, and the requests need to be routed to them. When an upstream host fails, TCP sessions need to restart, potentially throwing away already retrieved content. On the contrary, NDN provides a stateful hop-by-hop packet forwarding layer that automatically provides location independent data retrieval. Since NDN's network model perfectly aligns with CDN's application model, our motivation is to explore these synergies.

Paper organization:

Author Manuscript

The rest of the paper is organized as follows - Section II presents a brief introduction to a contemporary CDN architecture and NDN. It also discusses previous work that attempted to either apply NDN to CDN or compare NDN to HTTP. Section III presents an overview of our experimental topology and setup. Section IV evaluates NDN's architectural properties and points out the benefits of NDN. Section V discusses our future directions and concludes.

II. Background and Related Work

A. Named Data Networking

Named Data Networking (NDN) [1] is an instance of the Information Centric Networking (ICN) paradigm [3]. NDN utilizes application-defined, semantically meaningful, and hierarchical names for data publication. Once data is named and published, a user sends an Interest packet (query) into the network specifying the requested content's name. The intermediate routers maintain a name-based forwarding table, perform longest prefix match on the Interest name, and forward it over the appropriate interface(s). Once a producer receives the Interest, it returns a digitally signed Data packet. Intermediate routers cache these Data packets for future requests. For brevity, we do not discuss the NDN architecture further but refer the reader to the relevant publications [1][2].

NDN utilizes a hop-by-hop communication primitive [4] and not an end-to-end principle like TCP/IP. NDN also has one to one flow balance - for one Interest packet, the network returns only one data packet. A file on the disk is broken into several smaller Data packets (typically 8800 bytes or less). The client sends out an Interest for each of these Data packets. This hop by hop principle and the one-to-one flow balance means that in-network routers can perform several operations that are not easy to perform with the traditional TCP/IP architecture. Functions such as transparent failover to another source, parallel data retrieval from multiple upstreams, and utilizing the "best" path are simple to implement in NDN and can greatly benefit CDN operations.

Further, since there is no end-to-end connection setup, data retrieval starts faster (time to first byte) and helps with faster content delivery. Since the content is broken into smaller chunks, NDN supports partial data retrieval through caching; only the part that is not present in a cache can be retrieved from the origin server, reducing the total transfer volume.

B. Previous work

Yuan et al. presented an early study between HTTP and CCNx (an incarnation of the Information Centric Networking concepts) [5]. The work compares HTTP and CCNx and shows that in networks with very high drop rates (10% or more) NDN outperforms HTTP. The results do not apply to CDNs since such high loss rates in a CDN infrastructure are unacceptable.

Jiang et al. present nCDN that embeds NDN into the existing CDN framework by utilizing a hybrid model that uses NDN over UDP/TCP for content delivery [6]. The work requires a proxy that translates requests from the browser into NDN Interests before sending them into an NDN network. nCDN aims to improve the consumer experience while our work focuses on improving the backend infrastructure (e.g., PoP to PoP and POP to the origin server).

Lertsinsruttavee et al. present a study of CDN vs. NDN in a wireless mesh network [7]. This study is for wireless networks with completely different characteristics (e.g., loss and delay). Ma et al. present a comparison between CDN and NDN [8]. They show that NDN greatly simplifies the CDN design, provides better bandwidth utilization, and makes the CDN more scalable. The work focuses solely on throughput and misses the more useful parameters for

an actual CDN provider, such as time to first byte (TTFB) and partial content delivery from multiple caches.

We have also studied various aspects of IP and NDN. Mastorakis et al. performed comparative studies between NDN-based and IP-based solutions in the context of edge computing [9], [10] as well as for data sharing in infrastructure-based [11] and infrastructure-less [12] environments. Finally, Shannigrahi et al. present several studies that apply NDN for scientific data delivery in a fashion similar to CDN [13], [14], [15], [16]. However, these studies did not focus on CDN scenarios.

Finally, recent studies have explored the applicability of NDN for CDNs [17], [18]. Both of these studies focus on end-user experience, while our study focuses on finding NDN properties that can be beneficial to a contemporary CDN backend infrastructure.

III. Methodology

Figure 1 shows the topology we used for evaluating NDN in the CDN context. While simplistic, this topology is sufficient to demonstrate NDN and HTTP's properties. We utilized Ubuntu virtual machines (VMs) deployed on the Google Cloud Platform (GCP) to perform these experiments. Each VM had 4 CPUs, 4GB RAM, and 10GB disk space. The VMs were also located in the same region on Google Cloud. Depending on the experiment, we utilize the Linux utility "tc" to set delay and latency on the links (see Figure. 1).

The client represents a device that requests content from the CDN. The client-side cache can be deployed near the users to reduce latency and traffic volume. The intermediate caches provide hierarchical caching and origin shielding[19]. Any request not served by the client-side cache comes to one of these intermediate caches. These caches typically work as a reverse proxy, download the data from the origin and cache it for future requests. The origin servers serve authoritative copies of the content. Depending on the use case, a content provider might have one or more origin servers located at different geographical locations. In our experiments, the client-side cache is 50 ms away from the client[20], the latency between other components are 10ms, and the latency between intermediate cache and origin servers are 10-50ms that enumerates distances to different origin servers. These parameters are approximate but similar to what we will find in real-life CDN deployments [20]. The actual numbers will slightly vary depending on the region of the world and the service providers.

For the HTTP experiments, we utilize Apache Traffic Server (ATS) [21]. CDN providers often use homegrown and performance-tuned HTTP servers for their services; ATS is the closet open-source alternative that allowed us to create a CDN-like caching hierarchy. In the topology (Figure. 1), we configured ATS on the client-side cache to act as a forward proxy - it simply forwards the requests from the clients to the intermediate caches and caches the return data. The intermediate caches act as reverse proxies to the origin server. They receive the client-side cache requests, check if data is available in their local caches, potentially bring the data from the origin server before returning it to the requesting entity (the client-

side cache in this example). Multiple server-side caches are often utilized in a CDN to reduce individual caches' load and provide redundancy.

For the HTTP experiments, we used Apache Web Server on the origin server as the content producer. On the intermediate caching nodes, we utilized HTTP caching with cache sizes set to 2-4GB. The content we requested were much smaller than the available caches sizes. We also manually mapped each ATS instance to their respective upstream nodes. In ATS, these mappings are static and must be configured before running the experiments. Additionally, we had to set traffic as cachable using HTTP headers so that downstream ATS instances do not reject them.

For failure and switching traffic between multiple upstream caches, we used the ATS load balancer plugin. We used a round-robin policy that alternated traffic between two upstream servers. Note that upstream failover and load balancing can also be done with DNS, even though we did not explore DNS failover in this work. Given the additional communication overhead needed for DNS, we anticipate the results to be worse. Finally, we used "CURL" to retrieve content over HTTP.

For the NDN experiments, we published data from the origin server using `ndnputchunks`, a standard `ndn` tool [2]. We added UDP overlays between the nodes for a namespace (e.g., /test) and requested content from the client using another standard NDN tool, `ndncatchunks`. The intermediate caches were either disabled or set to a large capacity (2-4GB) depending on the experiment. We did not need any additional code for failover experiments since NDN supports such failover by default. For experiments with multiple upstreams, we added additional servers with higher latencies. We ran each experiment at least ten times for statistical accuracy.

IV. Evaluation

In this section, we discuss NDN and HTTP's performance in the context of a CDN. We compare the properties of HTTP over TCP (using ATS) with NDN. Note that HTTP over TCP is highly developed and optimized over the last forty years, while NDN is still a research prototype. Given the difference in the quality of code and optimization, HTTP performs better in raw throughput numbers. However, our goal in this work is not to compare throughput (or goodput) but to point out NDN's unique properties that can benefit CDN operations.

A. Baseline Goodput

First, we compare NDN with ATS for goodput and establish a baseline. Figure 2 shows this difference - HTTP is consistently an order of magnitude faster than NDN. However, the difference in performance is not exorbitant. For example, retrieving a 20 MB file from the origin server takes approximately 0.1 seconds over HTTP, while the same file over NDN takes 0.4 seconds. The numbers are similar when caching is enabled. Both NDN and HTTP times are proportionately lower with caching enabled. However, more importantly, the increase in delay follows the same pattern for both NDN and HTTP as the file size grows. The performance gap observed can be reduced by careful performance tuning of NDN such

as multithreading (NDN forwarder is currently single-threaded), reducing the number of copies, moving the code to the kernel space or using DPDK libraries[22].

HTTP over TCP does not perform well in the presence of loss. Surprisingly, the percentage of loss does not have to be significant for NDN to outperform HTTP. In this experiment, we set the loss on the access link (between the client and the client-side cache) to 0.08%. The other upstream links (between client-side cache and intermediate caches, and between the intermediate caches and the origin servers) a loss rate of 0.01%. We took these loss rates from another recent study on CDN traffic characteristics[23]. We performed these experiments with caching enabled and repeated ten times.

NDN performs better when loss is present as retransmitted data comes from the client-side cache rather than from the origin. This allows the clients to get the data faster without going into a congestion avoidance phase. With HTTP, the connection is end-to-end, and packet loss means the TCP goes into congestion avoidance mode, slowing down the transfer. With NDN, the loss recovery is faster, improving the throughput.

B. Time To First Byte (TTFB)

In this experiment, we measure the time between a request is sent and the client receives the first byte of data (TTFB) for HTTP and NDN. TTFB is essential for a CDN since it often dictates when the subsequent objects can be fetched (for example, various objects embedded in a webpage). It also affects the responsiveness of an application that can, in turn, affect the customer experience.

Our results indicate that TTFB is consistently lower for NDN compared to HTTP, with and without caching. Specifically, TTFB for HTTP range between 100 ms and 230 ms with and without caching, while TTFB for NDN range between 50 ms and 135 ms with and without caching, respectively. HTTP runs on top of TCP that performs a handshake to establish a connection before retrieving any data, resulting in higher delay. On the other hand, NDN can directly fetch data without requiring a connection set up or tear down, resulting in lower TTFB values. A lower TTFB is useful for delay-sensitive applications, for example, streaming videos.

C. Cache Utilization

CDNs are systems with cache hierarchies of large capacities. Maintaining a lower cache utilization per object across the system allows the CDNs to fit more content in the caches, speeding up the delivery of a diverse set of content. However, as CDNs start to retrieve and cache content, cache utilization per content goes up. For example, if a piece of content is retrieved through one upstream cache and later the system uses another upstream, the content is now cached at two places. The effective cache utilization by this content is then

$$\frac{2 \times \text{content size}}{\text{total cache size}}$$

With NDN, the caching granularity is in packets. If half of the content is fetched from upstream cache one and the rest from upstream cache two, the utilization remains at

$$\frac{\text{content size}}{\text{total cache size}}$$

Figure. 3 shows this experiment. We start with a large cache for both NDN and HTTP that is much larger than the content we are retrieving. We enable caching for the intermediate caches but not on the origin and client-side cache. We start by retrieving a certain sized content (e.g., 100 MB) through intermediate cache 1, and in the middle of the retrieval, switch to the second source. If the switch happens after 10MB data is retrieved, after the second pull, NDN consumes 10MB at Intermediate cache one and 100MB at intermediate cache two (10 MB from the first pull + 100 MB from the second pull) in total. However, HTTP consumes 200MB of cache space across Intermediate cache one and Intermediate cache two since it must bring in the complete file.

D. Partial Data Retrieval from Cache

In this experiment, we retrieve partial content from intermediate caches. With ATS, there are two ways to accomplish this. By default, the requests bypass the ATS caches and are directly forwarded to the origin server. Configuring ATS this way can negate origin shielding. A user can send several requests that request a sizable portion of a large file. ATS supports byte-range requests by bringing in the whole file in the cache from the origin server and then serving the byte-range requests from there. However, this approach is also problematic since requesting a small byte range from a large file will bring in the whole file into ATS's cache, reducing cache utilization. On the contrary, NDN creates smaller chunks from a larger content object. For example, a large file (e.g., /data_file) would be chunked into n data packets, where the packets will have names like "/data_file/segment=1", "/data_file/segment=2", ..., "/data_file/segment=3". These segments are individually cached in the intermediate caches. An application asking for segments 1 to 10 can retrieve them from the cache without bringing the whole file in or going to the origin server. When partial data is cached, NDN can serve the subsequent requests from the cache - request for any content that is not in the cache to the origin server.

Figure. 4 shows the comparison of HTTP vs NDN Byte range requests. NDN is consistently faster than HTTP for a lower volume of data. NDN outperforms ATS for partial data sizes up to 20MB. However, as the size of the byte range increases, NDN becomes slower compared to HTTP since it is not optimized for performance.

E. Transparent Failover

In a CDN, there are multiple intermediate caches (a hierarchy of caches). With HTTP and NDN, a user or a cache can be redirected to a better upstream based on network condition or upstreams' health. However, in HTTP, ongoing dataflows stick to the same degraded path (TCP is end to end) even when multiple paths are available. When the upstream fails, the already retrieved data might be thrown away, and the client has to restart the retrieval. The client may attempt to resume the data retrieval from another source but it requires a new end-to-end connection. On the contrary, NDN forwards packets hop by hop - it can switch upstreams in the middle of a transfer without restarting the retrieval. Upstream switching in the network is faster and less cumbersome than changing path at the application level.

In this experiment, we configured both ATS and NDN with multiple upstream servers. We pointed the ATS instance on the client-side-cache to two upstreams ATS instances that were

acting as reverse proxies for the origin server. Similar to ATS, we pointed the NDN instance on the client-side-cache to two upstream NDN caches (on intermediate nodes 1 and 2). First, we started the transfer from the client, and as data starts to come in, find and kill the intermediate node that was serving the data. For example, the first client (HTTP) request went through the following path - Client $\rightarrow$ Client-Side Cache $\rightarrow$ Intermediate Cache 1 $\rightarrow$ Origin Server. When data started coming back, we stopped the ATS instance on Intermediate cache 1. Note that while ATS was configured with multiple upstreams, TCP connections are end-to-end, requiring to reconnect on failure. In this scenario, the content retrieval stopped after the failure occurred.

We followed the same steps with NDN. We started the data transfer and killed the Intermediate node serving the request (the upstream), the client-side-cache immediately switched over to another available route, and there was no interruption in the transfer. The delay between two subsequent packets during the failover was 5.72ms with a standard deviation of 3.15ms. Figure 5 demonstrates this scenario. The HTTP transfer and the NDN transfer failed around the same time (at second 3 during the transfer). However, the NDN transfer continued with the alternate upstream while the HTTP transfer failed. Note that the NDN upstream switching is configurable and can happen without failures. NDN can also switch paths based on observed delay, loss, and any other relevant parameters to the CDN as we will show in the next section.

F. Automatically switch to the better path

In this experiment, we pick a “best” link depending on the network condition. NDN’s hop by hop forwarding allows network operators to monitor network performance and seamlessly move over to better data sources or links. Compare this to TCP flows, where performance degradation will require setting up a new TCP connection and restart the data retrieval. By not throwing away already retrieved content, NDN can improve completion time and provide a better experience for clients (they do not see the losses or quality degradation). In the context of a CDN, this is not only applicable to Client to PoP but also Pop to Pop communications.

In this experiment, we utilize two upstreams from the client-side cache. For both NDN and HTTP, we configure upstreams routes with similar weights. The weight is a function of delay and loss. For simplicity, we choose the function to be (arbitrarily) $f(n) = [100 * \text{delay in ms} + 100 * \text{percent packet loss}]$. In the middle of data retrieval, we randomly change delay and loss. The loss is randomly chosen between 0.001% and 1%. The delay is somewhere between 50-200ms. With a 50 ms delay and 0.001% loss, the weight of the path is 5000. When the delay increases to 100ms and the loss to 1%, the weight becomes 5100. Note that the actual numbers are not important in this experiment as long as a route’s weight increases with the loss and delay, allowing us to switch over to a “better route.” In the context of a CDN, various other parameters can be added to the equation along with their respective weights.

With HTTP, an existing transfer sticks to the path even when the path deteriorates. ATS provides a mechanism to transparently switch over to another upstream by rewriting its configuration file. This mechanism also requires a separate monitoring service that rewrites

the ATS configuration file to direct traffic to a more preferred origin server. However, switching the upstream origin does not change flows that are already transferring data. These flows will stick to the worse path until the transfers finish or completely fail.

In NDN, the data plane monitors the path quality and automatically switches over to an alternate path without involving any control plane mechanism. Additionally, by not throwing away already retrieved content, NDN can improve completion time and provide a better experience for clients (they don't see the losses or quality degradation on a specific path).

V. Conclusion and Future Work

In this work, we demonstrate how NDN's architectural artifacts can help a CDN infrastructure. By moving important functionality into the network (rather than handling them at the application layer) NDN can simplify a CDN's infrastructure and improve its capabilities. We utilize ATS to emulate a CDN-like caching hierarchy to compare HTTP based content delivery to NDN based content delivery. We show that NDN can deliver better performance in the presence of loss (even when the loss is small), reduce per-content cache utilization, and provide better time to the first byte that improves user experience. Further, NDN enables several mechanisms that are difficult to achieve using HTTP. Properties such as partial data retrieval, the ability to automatically switch to a better path, transparent failover, and the ability to retrieve content from multiple sources are important properties that not only simplify a CDN infrastructure but also enhance user experience. Table I summarizes these observations.

We acknowledge that this work is preliminary. We plan to extend this study in several ways. First, we plan to utilize a representative CDN network topology with a larger set of links and nodes. Second, we plan to utilize real data access patterns observed in a CDN to investigate how NDN can benefit users and the CDN provider in terms of better resource utilization and efficient content delivery. Third, we plan to create a pilot deployment inside a large scale CDN to understand better the trade-offs of applying an Internet architecture with a difference design philosophy to a massively large-scale, complex, and distributed system. Finally, we plan to make use of a high-performance version of the NDN forwarder, ndn-dpdk [24], [22] that is under active development and can reach up to 100Gbps of forwarding speed.

Acknowledgments

This work is partially supported by a pilot award from the Center for Research in Human Movement Variability and the NIH (P20GM109090), Tennessee Tech Faculty Research Grant, and the National Science Foundation (awards 2016714, 2019163, and 2019012).

References

- [1]. Zhang L, Afanasyev A, Burke J, Jacobson V, Claffy K, Crowley P, Papadopoulos C, Wang L, and Zhang B, "Named data networking," ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, pp. 66–73, 2014.
- [2]. Afanasyev A, Shi J, Zhang B, Zhang L, Moiseenko I, Yu Y, Shang W, Li Y, Mastorakis S, Huang Y et al. "Nfd developer's guide," NDN-0021, 2014.

- [3]. Xylomenos G, Ververidis CN, Siris VA, Fotiou N, Tsilopoulos C, Vasilakos X, Katsaros KV, and Polyzos GC, "A survey of information-centric networking research," *IEEE communications surveys & tutorials*, vol. 16, no. 2, pp. 1024–1049, 2013.
- [4]. Vusirikala S, Mastorakis S, Afanasyev A, and Zhang L, "Hop-by-hop best effort link layer reliability in named data networking," *NDN*, Technical Report NDN-0041, 2016.
- [5]. Yuan H and Crowley P, "Experimental evaluation of content distribution with ndn and http," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 240–244.
- [6]. Jiang X and Bi J, "ncdn: Cdn enhanced with ndn," in *2014 IEEE Conference on Computer Communications Workshops (INFOCOM WK-SHPS)*. IEEE, 2014, pp. 440–445.
- [7]. Lertsinsruttavee A, Mekbungwan P, and Weshsuwannarugs N, "Comparing ndn and cdn performance for content distribution service in community wireless mesh network," in *Proceedings of the AINTEC 2014 on Asian Internet Engineering Conference*, 2014, pp. 43–50.
- [8]. Ma G, Chen Z, Cao J, Guo Z, Jiang Y, and Guo X, "A tentative comparison on cdn and ndn," in *2014 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2014, pp. 2893–2898.
- [9]. Mastorakis S, Mtibaa A, Lee J, and Misra S, "ICedge: When Edge Computing Meets Information-Centric Networking," *IEEE Internet of Things Journal*, vol. 7, no. 5, pp. 4203–4217, 2020.
- [10]. Mastorakis S and Mtibaa A, "Towards service discovery and invocation in data-centric edge networks," in *2019 IEEE 27th International Conference on Network Protocols (ICNP)*. IEEE, 2019, pp. 1–6.
- [11]. Mastorakis S, Afanasyev A, Yu Y, and Zhang L, "nTorrent: Peer-to-peer file sharing in named data networking," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*. IEEE, 2017, pp. 1–10.
- [12]. Mastorakis S, Li T, and Zhang L, "Dapes: Named data for off-the-grid file sharing with peer-to-peer interactions," *arXiv preprint arXiv:2006.01651*, 2020.
- [13]. Shannigrahi S, Fan C, and Papadopoulos C, "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*, 2017, pp. 54–65.
- [14]. Fan C, Shannigrahi S, DiBenedetto S, Olschanowsky C, Papadopoulos C, and Newman H, "Managing scientific data with named data networking," in *Proceedings of the Fifth International Workshop on Network-Aware Data Management*, 2015, pp. 1–7.
- [15]. Lordache C, Liu R, Balcas J, Šrivinskas R, Wu Y, Fan C, Shannigrahi S, Newman H, and Yeh E, "Named data networking based file access for xrootd," in *J. Phys. Conf. Ser.(to appear)*, 2020.
- [16]. Shannigrahi S, Fan C, and Papadopoulos C, "Named data networking strategies for improving large scientific data transfers," in *2018 IEEE International Conference on Communications Workshops (ICC Workshops)*, 5 2018, pp. 1–6, DOI: 10.1109/ICCW.2018.8403576.1109/ICCW.2018.8403576.
- [17]. Ghasemi C, Yousefi H, and Zhang B, "icdn: An ndn-based cdn," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 99–105.
- [18]. —, "Far cry: Will cdns hear ndn's call?" in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 89–98.
- [19]. Afergan MM, Ellis AB, Sundaram R, and Rahul HS, "Method and system for protecting web sites from public internet threats," 8. 21 2007, uS Patent 7,260,639.
- [20]. ProspectOne, "CDNPerf - CDN Performance and Uptime monitoring, comparison and analytics - RUM data," 10 2020, [Online; accessed 30. Oct. 2020]. [Online]. Available: <https://www.cdnperf.com>
- [21]. "Apache Traffic Server," 8 2020, [Online; accessed 23. Aug. 2020]. [Online]. Available: <https://trafficserver.apache.org>
- [22]. Shi J, Pesavento D, and Benmohamed L, "Ndn-dpdk: Ndn forwarding at 100 gbps on commodity hardware," in *Proceedings of the 7th ACM Conference on Information-Centric Networking*, 2020, pp. 30–40.
- [23]. "BBR Evaluation at a Large CDN - eng. age," 9 2019, [Online; accessed 30. Oct. 2020]. [Online]. Available: <https://eng.verizondigitalmedia.com/2019/09/27/bbr-explore>

- [24]. Khoussi S, Nouri A, Shi J, Filliben J, Benmohamed L, Battou A, and Bensalem S, “Performance evaluation of the ndn data plane using statistical model checking,” in International Symposium on Automated Technology for Verification and Analysis. Springer, 2019, pp. 534–550.

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

Fig. 1:
Experimental Topology. The infrastructure inside the box represents a CDN backend (PoP-to-PoP) - the focus of this work.

Fig. 2:

HTTP and NDN file retrieval performance. HTTP/ TCP is faster without any loss. In the presence of loss NDN performs better. The loss was set to 0.08% on the access link and 0.01% on the other two links for a total of 0.1% loss.

Fig. 3:
HTTP and NDN cache utilization

Fig. 4: HTTP and NDN Partial Data Retrieval. The x-axis shows the byte ranges requested over HTTP and NDN. ATS either needs to fetch the full file or send the request to the origin server before serving a byte range request. NDN can reuse content chunks that are already present in the cache.

Fig. 5:

HTTP and NDN file retrieval performance with failure. NDN's hop by hop forwarding transparently switches routes when upstreams fail. On the contrary, ATS connections are broken when upstream failure happens and forces the client to restart the transfer. While currently HTTP is faster than NDN, this gap will close as the NDN software stack matures.

Fig. 6:

NDN and HTTP performance with upstream performance degradation with different file sizes. This experiment uses multiple upstreams. As the upstream deteriorates, NDN moves to a better path while ATS sticks to the original path. The effect is more prominent as the file size increases.

TABLE I:

Summary of the experiments and our observations

Experiment	Observations
A. Content retrieval	With lossy links, NDN retransmits faster from cache, resulting in better throughput. Without loss, HTTP has better throughput due to optimizations
B. TTFB	NDN follows a hop-by-hop model, TTFB is faster since no end-to-end connection set up is required
C. Cache Utilization	NDN provides packet level caching. Overall cache utilization is lower since the caching granularity is packets and not files
D. Partial retrieval	NDN provides packet level caching. Any data packet that is in the cache is reusable and the rest can be fetched from the origin. The ability to reuse packets from cache means NDN is faster when partial content is requested.
E. Failover	NDN can support multiple upstreams without special configurations. Combined with hop-by-hop forwarding, NDN can seamlessly switch over to another upstream when the current upstream fails.
F. Path switching	NDN strategies can monitor path parameters (defined by network operators) and automatically switch to better paths when current paths degrade.