PCLive: Bringing Named Data Networking to Internet Livestreaming

Teng Liang
Peng Cheng Laboratory
Shenzhen, Guangdong, China
liangt@pcl.ac.cn

Wei Huang
Peng Cheng Laboratory
Shenzhen, Guangdong, China
huangw@pcl.ac.cn

Xinyu Ma
University of California, Los Angeles
Los Angeles, CA, USA
xinyu.ma@cs.ucla.edu

Weizhe Zhang
Peng Cheng Laboratory
Shenzhen, Guangdong, China
Harbin Institute of Technology, Shenzhen
Shenzhen, Guangdong, China
weizhe.zhang@pcl.ac.cn

Yu Zhang*
Harbin Institute of Technology
Harbin, Heilongjiang, China
Peng Cheng Laboratory
Shenzhen, Guangdong, China
yuzhang@hit.edu.cn

Beichuan Zhang
The University of Arizona
Tucson, Arizona, USA
bzhang@cs.arizona.edu

ABSTRACT

The lack of application support is probably the biggest obstacle to ICN/NDN deployment. One approach to tackle this problem is to *NDNize* existing applications by translating between application-level protocols and NDN, which can benefit from NDN's architectural advantages while minimizing development efforts needed. In this paper, we validate the effectiveness of this approach by applying it to Internet livestreaming, and develop *PCLive*, a livestreaming system with NDN embedded as its distribution network. PCLive makes minimal changes to an Internet livestreaming architecture, achieving the maximum compatibility with existing components including video players, OBS, and video transcoders. By solving a number of design issues such as HLS/NDN protocol translation, data translation, naming and security, PCLive is able to run over an NDN network and enjoy its architectural benefits. Since December 2021, PCLive has been running on an NDN testbed consisting of cloud servers from seven cities. It can serve almost four times as many clients as an existing livestreaming system can over IP under the same network conditions; at the same time, the average throughput of the bottleneck link in the NDN testbed is 34.8% lower than that in IP. We also evaluate congestion control and adaptive forwarding with PCLive.

KEYWORDS

Information-Centric Networking (ICN), Named Data Networking (NDN), Realtime Data Distribution, Internet Livestreaming, Network Measurement, Protocol Translation, NDNizing Applications

*Corresponding author

ACM ICN '23, October 9–10, 2023, Reykjavik, Iceland

ACM ISBN 979-8-4007-0403-1/23/10.
https://doi.org/10.1145/3623565.3623711

1 INTRODUCTION

The lack of application support is probably the biggest obstacle to Information-centric Networking (ICN) or Named Data Networking (NDN) [3, 43]. One approach to tackle this problem is to *NDNize* existing applications by translating between application-level protocols and NDN, which can benefit from NDN's architectural advantages while minimizing development efforts needed [20].

In this paper, following the NDNizing-application approach, we investigate NDNizing Internet livestreaming. The motivation of this choice is based on three observations. First, NDN is dedicatedly designed to support real-time data distribution with its built-in multicast and in-network caching mechanisms, hence it is supposed to provide significant benefits to livestreaming. However, today's widespread Internet livestreaming is not using NDN, hence the benefits are not fully verified. Second, NDN research is largely limited by lacking of real-world usage; the efforts to enable widely used livestreaming systems running over NDN can provide a large amount of real traffic in the real-time manner for NDN research. Last, though various NDN-based livestreaming projects have been proposed, there is no open-sourced, stable, and usable application.

Specifically, we develop PCLive, an Internet livestreaming system with NDN embedded as the distribution network. The first design decision to pick the target translating protocols. We decide to NDNize only the distribution part of an Internet livestreaming, because this achieves the benefits of NDN's distribution capability, while reusing the publishing part of the system. Moreover, we choose a widely used livestreaming retrieval protocol, i.e., HTTP Live Streaming (HLS) [28], for the translation. Similar to other NDNizing-application work, PCLive solves common design issues including protocol translation, data translation, naming, and security.

Interestingly, we find this translation practice is simpler than previous efforts, such as IMAP/NDN and XMPP/NDN translations [20]. We believe the simplicity comes from the data-centric nature of the translating application protocol, the well-designed softwares, and the choice of clear system boundaries. Specifically, modern application protocols, such as HLS and DASH, use the receiver-driven communication model, which is the same as NDN, hence

making the translation more straightforward. Moreover, thanks to the well-designed software architectures, PCLive offers a different translation design pattern, i.e., instead of using a proxy to serve as the server of the translating protocol to conduct the translation, PCLive directly translates the data retrieval protocol within the communication process of a video player, and translates livestreaming data together with a video transcoder, making NDN communication modules pluggable. As a result, PCLive achieves the maximum compatibility with existing system components including video players, OBS, and video transcoders.

To evaluate the system design, we implement PCLive with widely-used open-source softwares. From users' perspective, they first download a website containing a video player (i.e., Shaka Player [10]) with NDN plugged in as the communication module; to watch a livestreaming, the player sends HLS requests, which are translated and sent to an NDN network; the retrieved data are assembled and translated back as HLS responses. From a video publisher's perspective, she uses existing OBS softwares to upload a livestream to a video server, using Real-Time Messaging Protocol (RTMP) [31]; on the server, we run Simple Realtime Server (SRS) [1] to transcode RTMP streams into HLS files with different resolutions; we implement a real-time HLS/NDN data translation module, which prepares, stores, and serves NDN Data packets to an NDN network.

Moreover, following the idea of application-driven research, we build a small NDN testbed over the Internet learning from NDN Testbed [13], run PCLive on our testbed, and conduct comprehensive evaluation. The testbed consists of cloud servers in seven different cities; they are running NFD [1] [4] to process NDN packets, and are connected by UDP tunnels. Regarding routing protocols, NLSR [37] is used. Moreover, we build two systems of measurement, one is to collect data from each NFD, to analyze NDN stateful forwarding behaviors; the other one collects data from consumers to evaluate QoS.

The evaluation results show that PCLive can serve 3.95 times clients over the NDN testbed as many as an existing livestreaming system can serve over IP on the Internet under the same network conditions; at the same time, the average throughput of the bottleneck link in the NDN testbed is 34.8% lower than that in IP. Last, we also evaluate other mechanisms, such as congestion control and adaptive forwarding with PCLive.

To summarize, the contributions of this work are as follows.

- Following the NDNizing-application approach, we investigate bringing NDN into real-world livestreaming architecture, and proposes PCLive, which replaces the stream distribution network with NDN by translating HLS/NDN. With system deployment over the Internet, long-time testing, and comprehensive evaluation, we demonstrate that this approach works well with minimized developing, deploying, and configuring efforts. Moreover, this practice offers a different translation pattern with simpler efforts, making NDN as pluggable modules.
- Following the idea of application-driven research, we conduct comprehensive evaluation on PCLive and related NDN designs on a wide-area NDN testbed over the Internet, including the QoS of PCLive, in-network caching, congestion control, stateful forwarding, off-the-grid communication. Our preliminary analysis help with future research directions.

[1] An NDN software forwarder widely used in NDN research communities. It is also used on NDN Testbed.

2 MOTIVATION

2.1 NDNizing Livestreaming Systems

Different from building native NDN applications from scratch, NDNizing an existing application means to translate its application-layer protocol and NDN, enabling it to run over NDN. This approach achieves a good trade-off between developing efforts and gaining NDN's architectural benefits [20].

To exam this argument, we categories related work in its manner. An NDN-based video streaming application, such as NDNVideo (2012) [16] and NDNlive/NDNtube (2015) [38]. In addition, ACT (2011) [44] is an audio conferencing tool for NDN, and NDNRTC (2015) [12] is a real-time conferencing application over NDN. Unfortunately, they are not being actively maintained anymore, hence cannot be used in real world.

Regarding the NDNizing-application approach, VoCCN (2009) was the earliest effort, which translates VoIP/CCN [15]. IMAP/NDN and XMPP/NDN translation was designed and implemented (i.e., mailSync [21]) in 2018 [20]. iVisa (2020) translates DASH/NDN to implement on-demand video streaming [9]. This work takes a step forward and translates HLS/NDN in an adaptive Internet livestreaming system. Because PCLive makes NDN pluggable modules between Shaka Player and Simple Real-time Server (SRS), NDN's distribution capability is fully utilized while the modified part is much easier to be maintained. Therefore, PCLive is more likely to have a longer lifetime as the rest of the system components are actively used and maintained by third parties.

Regarding work of dynamic adaptive video streaming in ICN, a systematic comparison is conducted between NDN and TCP/IP [30]. In addition, DASH over HTTP and CCN is studied and compared [17]. Moreover, several work studied video rate adaptation over ICN in different scenarios, such as in WLAN [40], in cooperation with in-network caching [14], and in a multi-client scenario [34]. This paper focuses on the approach to enabling an existing system with NDN's capability, instead of studying dynamic adaptive video streaming in ICN. Although we compare PCLive with HLS over IP in the Evaluation section, our purpose is to demonstrate the effectiveness of the NDNization approach with real-world usage.

2.2 Application-driven Research

Lacking of real-world usage is limiting the validation and evaluation of NDN protocols and prototypes. Because PCLive translates a popular protocol (i.e., HLS), and it plugs NDN into an existing Internet livestreaming system, it achieves good usability as no configuration or installation is required from users. Therefore, PCLive is a good example of real-world usage, and can bring a large amount of real-world traffic to NDN, hence can help related NDN research.

In this work, in addition to evaluate the functionality and performance of PCLive, we also conduct preliminary study on the performance of NDN's in-network caching, built-in multicast, and adaptive forwarding over the Internet with PCLive. Specifically, we

build an NDN testbed with servers located in seven different cities connected through the Internet, and run PCLive over it.

3 SYSTEM DESIGN

3.1 Design Overview

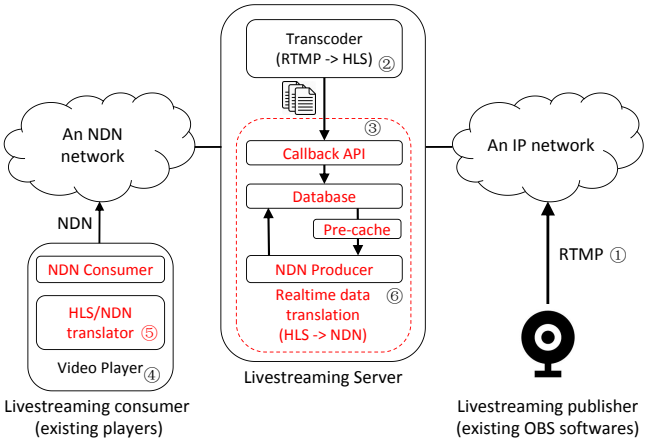


Figure 1: System overview of PCLive

The first step of the NDNization progress is to decide its boundaries in an existing system. A good decision maximizes the benefits with the minimized developing efforts and the least requirements on users. One common Internet livestreaming architecture contains two major parts, livestream publishing and livestream distribution (Fig. 1). PCLive replaces the distribution part with NDN, while reusing the publishing part.

Specifically, in the publishing part, an existing OBS software captures a livestream and pushes it to a server through RTMP (step 1), and the server transcodes the RTMP stream into HLS files with different resolution qualities to achieve adaptive bitrate purpose (step 2). PCLive adds a new process, which translates HLS files into NDN Data formats, stores them into a database, and serves them to an NDN network on the fly (step 3).

The distribution part in NDN consists of three steps. A consumer uses a web browser to download a web page, which contains a video player with NDN plugged in as its communication module (step 4). The video player is able to translates HLS/NDN, and send/receive NDN Interest/Data packets to an NDN node in an NDN network (step 5).

3.2 Naming

The namespace design of PCLive contains two parts (Fig. 2). The first part is to identify a livestream in an NDN network, and the second part follows how the livestream is organized and accessed.

Specifically, the first part contains four name components. The first several name components are formed as a globally routable prefix, i.e., `/pcl/video`, for the livestreaming server. The third name component is `/live`, which is used to identify the target application on the server. The fourth name component is `<uuid>`, which is to uniquely identify a livestreaming.

Because PCLive translates HLS, the second part of its namespace design follows how HLS organize and access data. Specifically, a livestream contains a master playlist (in m3u8 format), separate media playlists (in m3u8 format) for each resolution, and sequential video chunks (in ts format).

Here is the lesson learned by this case study. In the NDNization progress, the namespace design can be summarized as two steps. The first step to design a globally reachable name prefix in an NDN network to identify a group of data or services; the second step is to design name suffixes, referring to the access and organization of data in application logics, which come along with protocol translation and data translation. Moreover, it once more demonstrates that naming design in this approach is significantly simpler than building a native NDN application from scratch, because it avoids designing new application protocols, which requires massive usage to reach as the same maturity as existing widely-used application protocols.

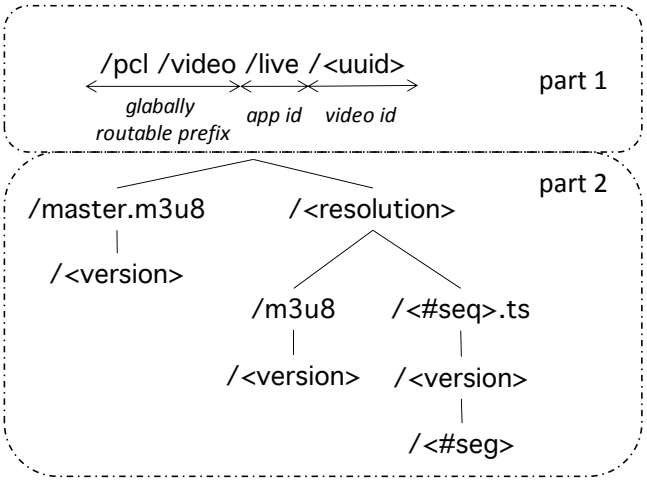


Figure 2: PCLive namespace design

3.3 Protocol translation

Protocol translation is the key step of the NDNization process. Given that application-level protocols contain well-defined logics, translating them into NDN saves the efforts of creating new application protocols. In addition, these protocols explicitly expose application semantics, making their naming more data-centric than lower level protocols with host-centric headers, hence such translation gains more NDN's architectural benefits. However, the challenge is to translate existing message exchange into NDN's data-centric exchange. We take HLS/NDN translation to illustrate two common translation design patterns.

HLS adaptively retrieves livestreaming through three steps (Fig. 3). HLS first fetches a master playlist, then periodically fetches a media playlist based on the estimated bandwidth, which guides fetching new video chunks. Given that HLS is a receiver-driven protocol, and all the retrieval operations are implemented though HTTP GET, translating them into NDN's Interest-Data exchange is straightforward, and can achieve the same real-time data retrieval effect. The translation considers two issues.

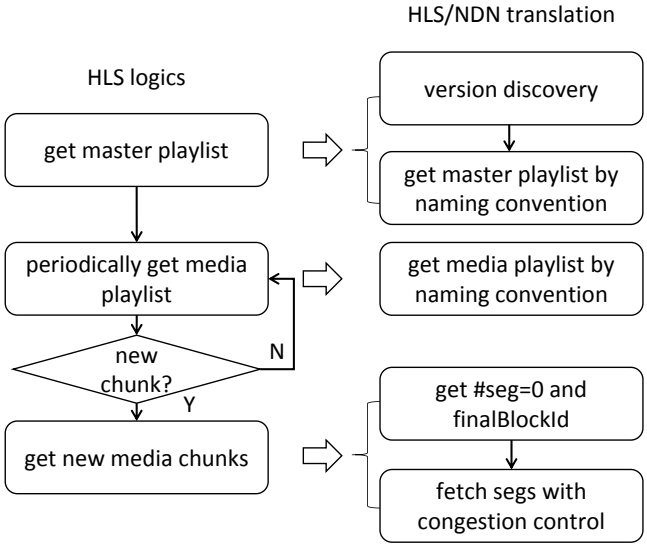


Figure 3: Protocol translation

- **Version discovery:** when fetching the master playlist, PCLive first discovers its latest version, then uses the version number to construct the name to retrieve the playlist. Note that we only use version discovery at this step, and keeps using it for subsequent media playlist and video segments retrieval. Using new version number for newly generated video chunks creates non-trivial overhead of version discovery and data management, without gaining clear benefits for livestreaming.
- **Data retrieval:** in HLS, when requesting a video chunk, the response will be pushed back as an entire file. In being translated into NDN, the video file is too big to fit in to a Data packet, hence has to be segmented into multiple packets, e.g., a 4-s video chunk with 1080P resolution can be segmented into more than a thousand NDN Data packets with a size of 4K bytes. Therefore, PCLive fetches the first segment, which contains the total number of segments, and then fetches all of them with congestion control mechanisms.

3.4 Data translation

We summarize data translation into two parts, data generation and data serving. Regarding data generation, it has two types, on-demand data generation for version discovery and the conversion of video files. Fig. 1 shows the process of converting video files from HLS formats into NDN Data formats. Specifically, when video files are generated on the fly, a callback function is invoked to convert video files into NDN Data formats with naming and signing, which are then stored into a database.

Regarding data serving, a producer program connects to an NDN network, and registers the name prefix `/pcl/video/live` to it. It is responsible to handle both version discovery and livestreaming requests. Regarding version discovery, the producer generates on-demand data that contains the latest version for a version discovery Interest (using Real-time Data Retrieval (RDR) protocol [24]). During our testing, we made an observation that frequent database write and read operations (for every NDN Data packet) causes the bottleneck of PCLive, and optimizes it by merge separate reads of video chunk segments into one read at the first segment read, which preloads an entire video chunk into the cache to serve subsequent segment requests.

3.5 Security

The trust rule in PCLive is simple, i.e., consumers trust the PCLive producer. The PCLive producer has an identity with name `/pcl/video/live`, a public key with name `/pcl/video/live/KEY/<id>`, and a self-signed certificate with name `/pcl/video/live/KEY/<id>/self/<version>`. The producer signs all NDN Data packets using the private key, and the keylocator contains the certificate's name.

In our current implementation, when PCLive consumers download the webpage, they download the certificate and the trust rule together. The trust scheme [42] in PCLive stricts consumers only to trust data packets signed by the identify of `/pcl/video/live`. Because HTTPS is used for website downloading, both the certificates and trust rule are secured. Note that we leave content confidentiality to future work.

4 DEPLOYMENT

4.1 A wide-area NDN testbed

Next, we introduce the NDN testbed we built among cloud servers at different locations over the Internet.

- **Topology:** Our testbed topology is shown in Fig. 4. The testbed contains 7 nodes from different cities.expanding techniques. We host servers in two cities, Shenzhen and Jinghua, and rent cloud servers from Alibaba Cloud in 5 cities, Hongkong, Guangzhou, Shanghai, Chengdu, and Beijing.
- **Networking:** On each node, we run NFD, an NDN forwarding daemon. The link between two nodes is a UDP tunnel. Specifically, NFD has a type of UDP Face, meaning NFD uses UDP encapsulation to transport NDN packets. Moreover, the MTU of UDP encapsulation is intentionally set to be 1420, and NDNLP is used between NDN and UDP for packet fragmentation (more details below).
- **Routing:** We run NLSR routing protocol over the testbed, following its specification which runs over the NDN Testbed.
- **Hardware resources:** All NDN nodes have 2 cores of CPU, 4 GB memory, and are running Linux Ubuntu 20.04. In addition, each node has a limit of 50 Mbps uploading bandwidth.

4.1.1 Packet fragmentation: NDNLP vs. IP. During our early testing of PCLive, we made one observation that an extremely small number of NDN packets are unable to be retrieved through our NDN testbed, hence the corresponding video segment is unable to be assembled at consumers. This problem is eliminated after forcing packet fragmentation between the NDN layer and the underlaying tunnel, i.e., UDP in our testbed, instead of triggering on IP fragmentation.

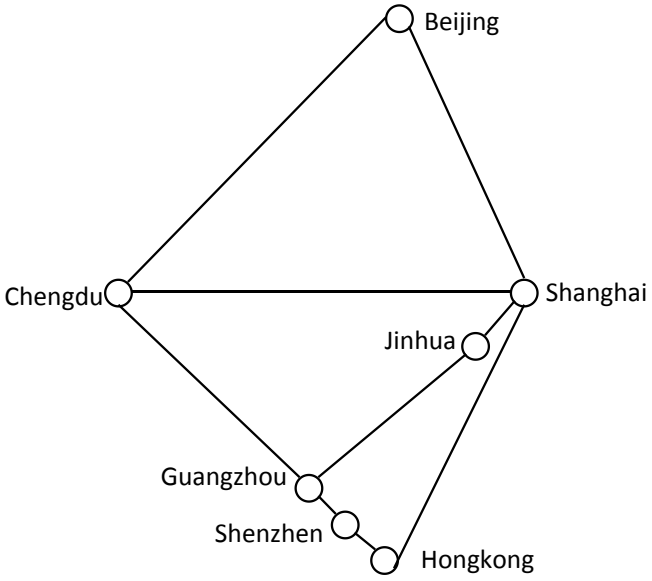


Figure 4: The geo topology of an NDN testbed (including 7 cities) over the Internet

Our NDN testbed uses UDP tunnels for interconnections. Because an NDN Data packet has a size limit of 8800 bytes, therefore when it is encapsulated in a UDP packet, and transmitted over a link with a smaller MTU (e.g., ethernet MTU is 1500 bytes), IP fragmentation is triggered. We hypothesize a few fragmented IP packets are blocked for security considerations, thus contributing to the observation we made.

To tackle this issue, we intentionally set the MTU of an UDP tunnel to be a smaller number (i.e., 1420 in our deployment), hence triggering packet fragmentation defined in NDNLP protocol [33]. As a result, each NDN Data packet is fragmented and encapsulated into a UDP packet with a size of 1420, which is expectedly not to trigger IP fragmentation during Internet transmission.

4.2 PCLive deployment

The PCLive livestreaming service is deployed on a server, which connects to the Shenzhen node of the testbed. The server has 20 cores of GPU, 32 GB memory, and is running Linux Ubuntu 20.04.

- **Livestreaming service:** PCLive uses SRS to handle livestream publishing and RTMP-HLS transcoding; once HLS files are generated, a live scanner web service will be invoked to convert them into NDN Data packets, which are stored into MongoDB [27]. In addition, a livestream producer is running to handle NDN livestreaming requests, serving prefix registration.
- **On-demand video streaming service:** PCLive can also provide on-demand video streaming service with MinIO [26] used as the database, and a video-on-demand producer to handle its service.
- **Web service:** we use Nginx [29] and Flask [11] to host the entry website. PCLive clients use secure websocket [25] to connect the nearest node from the NDN testbed using NDN-FCH protocol [2]. Note that we host a FCH server on our own, instead of using the one for NDN Testbed.

Moreover, two systems of measurement are deployed to collect data for profiling and analysis. All these services are running in dockers with docker-compose for management.

4.3 Systems of Measurement

We collect data from both clients and routers to measure both users' QoE and NDN stateful data plane behaviors. Specifically, QoE metrics are collected from Shaka Player, including startup delay, estimated bandwidth, video resolution, playing status, and round-trip time. Regarding NDN data plane behaviors, NFD contains a monitoring system to collecting metrics, including the number of incoming and outgoing Interest and Data packets on each Face, their size, and the CS hit counter.

To collect QoE metrics, the ELK Stack [5] is used; clients periodically collect videos statistic, then send the data back to the server with search and visualization functionalities. Regarding NFD status, NFD's measurement system provides a daemon that enables the retrieval of NFD status via HTTP protocol [35], and we use a web crawler to periodically collect those status from each NDN testbed node.

5 EVALUATION

5.1 Target

The main goal is to evaluate the effectiveness of how PCLive NDNizes an existing Internet livestreaming. Therefore, our intention is to implement and run it in real world as system evaluation. We built a wide-area NDN testbed as an overlay network among several cloud data centers, and deploy PCLive to the server in Shenzhen.

Because we can collect data from both nodes in network and clients, we second goal is to evaluate some well-known features from NDN, such as in-network caching. We compare it with livestreaming from a single server in IP, just demonstrate the in-network benefits from NDN. Note that livestreaming in IP is highly optimized, e.g., CDN or SRS Edge Server mode can be applied to scale streaming distribution, which is not our goal of comparison.

5.2 Setup

The evaluation of PCLive is conducted with its deployment on the NDN testbed over the Internet since Dec. 2021. Two systems of measurement (Section 4.3) are used to collect both consumers' QoS metrics and in-network states, to analyze the performance of both PCLive and the testbed. First, we compare the real-time data distribution capability between an NDN and IP network (Section 5.3). Next, we compare the choice of packet fragmentation at different layers over the Internet (Section 4.1.1). In addition, we evaluate the impact of Data content size (Section 5.4) to livestreaming. Moreover, we use the data to analyze existing congestion control mechanisms and stateful forwarding behaviors. (Section 5.5 and 5.6). Last, we demonstrate the off-the-grid communication capability of NDN in Section 5.6.

Network	Clients	Avg. throughput (Mbps)	Peak throughput (Mbps)	Bottleneck
IP	21	51.5	82.9	end server
NDN	83	33.6	83.4	edge node

Table 1: A case of livestreaming distribution capability comparison built over IP and NDN

5.3 Distribution capability: NDN vs. IP

One goal of the evaluation is to compare the real-time data distribution capability between NDN and IP. Because PCLive NDNizes an existing livestreaming system, this is a relatively fair comparison as it minimizes external factors, meaning both testing use the same video player, OBS software, and transcoding software. In addition, real-world livestreaming over Internet.

Specifically, regarding NDN testing scenario, we run consumers connecting to different NDN nodes, i.e., virtual machines running on Alibaba cloud data centers in different cities, including Beijing, Shanghai, Chengdu, Guangzhou, Shenzhen, and Hongkong. We increase the number of consumers by opening more websites that connect to an NDN node with the NDN-capable video player running. Regarding the IP testing scenario, we run clients connecting to the centralized singer server running in Shenzhen. In order to create the same network paths for data retrieval, traffic are routed though the aforementioned nodes in the topology, creating the same network conditions. We develop an automation tool to manage thirty machines from a private cloud, which have enough resources to run clients at the same time. In both scenarios, QoS metrics are collected to profile the quality of the livestreaming. To find throughput bottleneck, *iftop* tool is used to measure the throughput at video server for IP testing scenario, while NFD status is used to find that in NDN testing scenario.

We conduct 10 times 3-minute testing for each scenario. The results are summarized in Table 1. In the IP scenario, clients are connected to the centralized server that runs SRS in Shenzhen, hence the bottleneck happens at the end server. As we increase the clients by opening more websites that run the video player, the maximum number of clients is 21 during our testing. If one more website is open after 21 clients are connected, all the video players will stop working. The average throughput of is 51.5 Mbps, and the peak throughput is 82.9 Mbps. These two numbers are collected on the server node.

In the NDN scenario, clients are connected different NDN nodes, which are shown in the wide-area overlay topology 4. We equally increase the number of consumers connect to each NDN node. Thanks to NDN's in-network caching and Interest aggregation, the effect of multicast happens at every NDN node, hence the bottleneck may happen at any node if more clients are connect to it. Fig. 5 shows one testing result for the NDN testing scenario, with data collected from each NFD from the NDN testbed. Circles in Fig. 5 are outliers. The average throughput for all NDN nodes, including Beijing, Chengdu, Guangzhou, Hongkong, Shanghai, and Shenzhen, is 33.6 Mbps, and the peak throughput is collected from Shenzhen node, which is 83.4.

According to the collected data during testing, PCLive can serve 3.95 times as many clients (83) as an existing livestreaming system can over IP (21) under the same network conditions; at the same time, the average throughput of the bottleneck link in the NDN testbed is 34.8% lower than that in IP (Table 1). Note that we expect each NDN node to play the role of the server in IP, hence the NDN testbed is supposed to serve N times as many clients as the IP network can, where N equals to the number of NDN nodes that equip with in-network caching. However, our observation is that the performance of the NDN forwarder (i.e., NFD) cannot reach the performance of the SRS server.

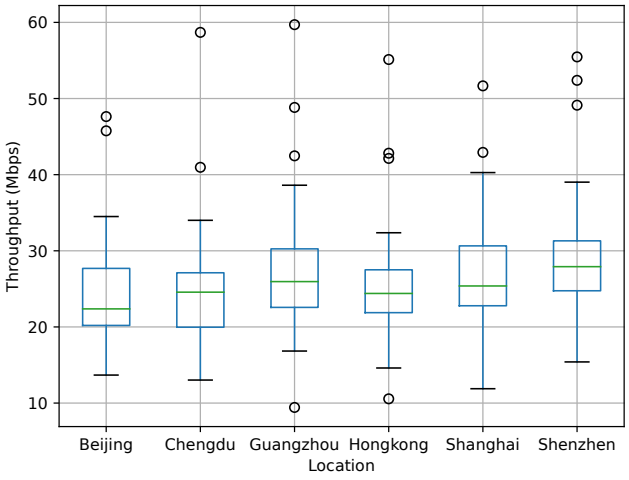


Figure 5: The throughput on different NDN nodes

5.4 Data Translation Overhead

When converting HLS files into NDN formats, one parameter is the content size of a Data packet. Given that a Data packet is limited to 8800 bytes, we evaluate this parameter by choosing its value between 1000 and 8000 with a step of 1000. The result is shown in Fig. 6. The overhead of packet headers, including name and signature, can be as high as more than 25%, when the content size is 1000, while it can be as low as less than 5% when the value is more than 6000. It also has an impact on the performance of the current congestion control mechanism (introduced in next section). Therefore, we recommend choosing a larger content size based on the results.

In addition to extra header overhead, we also evaluated the translation time overhead 7. We conducted tests on HLS file sizes ranging from 1000 bytes to 8000 bytes, packaging HLS data packets into the NDN format, with each file size executed 100 times. The packaging overhead is not significantly impacted by file size. The packaging overhead for each data packet is approximately 2.5 milliseconds.

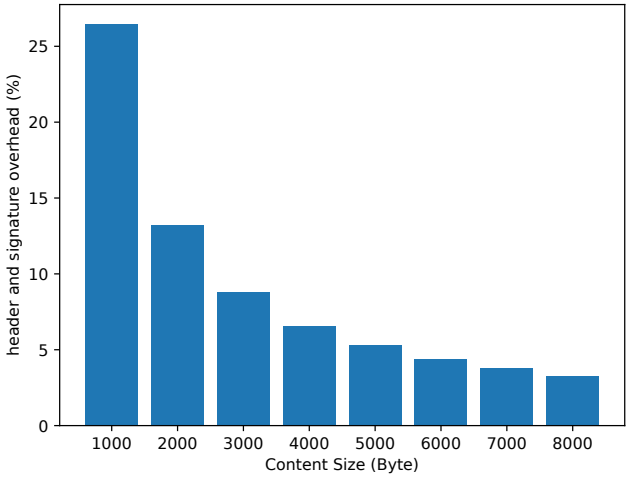


Figure 6: Translating one ts file into NDN Data packets: header and signature overhead

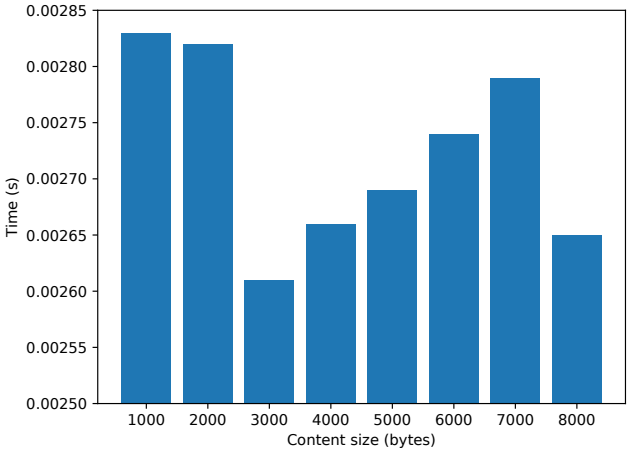


Figure 7: Translating one ts file into NDN Data packets: time overhead

5.5 Congestion Control

Next, we evaluate the current congestion control mechanisms in PCLive. PCLive uses the congestion control mechanism implemented in NDNts library, which uses CUBIC algorithm to adjust the Interest sending window size. The signal to trigger window decreasing only relies the timeouts of data retrieval with an estimated round trip timer.

In one testing scenario, we only run five clients, and observe their livestreaming quality as we change the content size of the translated Data packets. Although the client can always retrieve the livestreaming with 1080P resolution, because its estimated bandwidth is higher than the required value (i.e., 5Mbps). However, the estimated value can be significantly lower than the actual value. As shown in Fig. 8, when the content size is 1000, their estimate bandwidth to be slightly higher than 5 Mbps, while this value increases when the content size increases.

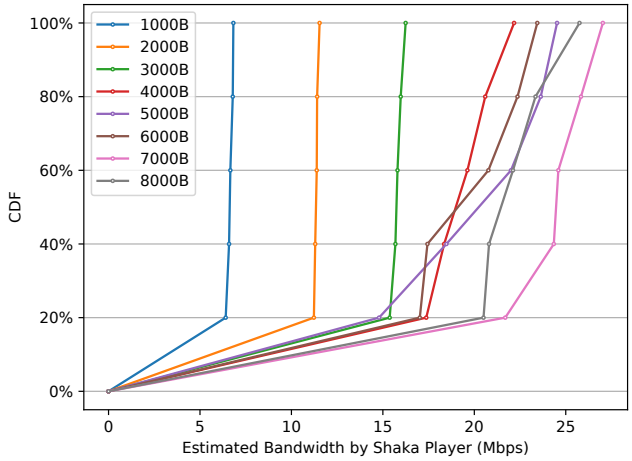


Figure 8: Estimated Bandwith

However, when the content size is 7000 , the estimate bandwidth is higher than that when it is 8000. This is because during the testing the translated video files have different size, and when the content size is 7000, the HLS video file happens to be significantly smaller than that when the content size is 8000 (Fig. 9). Therefore, the actual number of Data packets is smaller even the content size is smaller during test. This further proves that the estimated bandwidth is proportional to the number of Data packets to retrieve; for the same size of video files, the less number of Data packets to be retrieved, the higher estimated bandwidth. This observation suggests that the current congestion control mechanism performs worse when the number of Data packets to be larger.

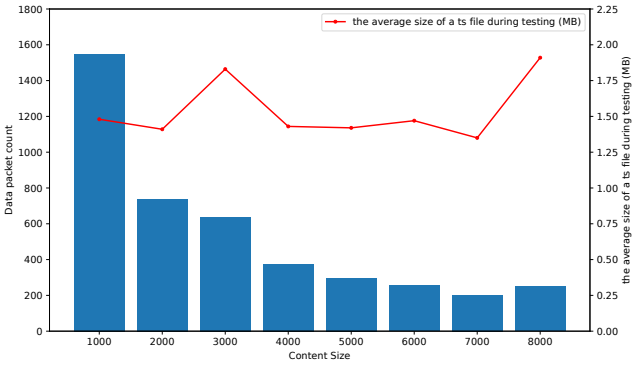


Figure 9: The number of NDN Data packets for different content size during testing

Interest retransmission rate is an important metric to evaluate congestion control mechanisms. During our testing, we observe that Interest retransmission rates are fairly low. Fig. 10 shows Interest retransmission for consumers grouped by their connected cities.

In this testing, each node has 14 consumers connected. During a 3-minute livestreaming testing, the 14 consumers connecting to Beijing node have a summed of about 1.2% Interest retransmission rates.

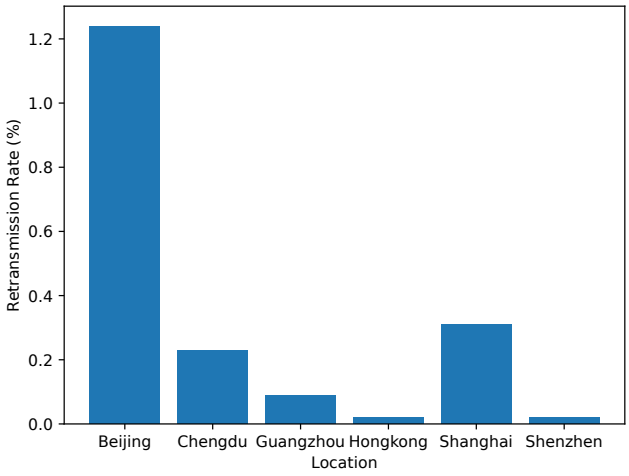


Figure 10: Retransmission rates

5.6 Stateful Forwarding

Next, we are interested in how Interests are handled on the NDN testbed with PCLive, hence we analyze data collected from each NDN testbed node. First, we summarize how an Interest is handled by NDN's stateful forwarding plane.

- Satisfied by Content Store (CS): the Interest is satisfied directly by a data caching. NFD records the counter named as *CS hit* referring to the number of Interests satisfied by local caching.
- Aggregated (suppressed): the Interest is neither satisfied by local cache nor being forwarded, because an Interest requesting for the same data has been forward. This value is not recorded by NFD, and we calculate this value with the following equation

$$SuppressedInterests = IncomingInterests - outgoingInterests - CShits$$

.

- Forwarded: the Interest is forwarded to a next hop. The testbed is running adaptive forwarding strategy (i.e., ASF strategy [18, 41]).

We increase the clients connecting to each of six nodes from the NDN testbed respectfully between 2 to 10, and depict Interest handling on the NDN testbed in Fig. 11. To conclude, CS hit is the major behavior in our testing, whose value increases as the number of consumers increase, which can reaches to 70% when the number of consumers is 10. It is the major factor to NDN's real-time data distribution capability as expected.

Next, the number of suppressed Interests is fairly low compared to CS hit. The explanation is that only a small portion of real-time requests from different consumers are arriving at the same time (with very variations) during our testing, which are suppressed.

Moreover, Interest suppression is less aggressive in the current adaptive forwarding design, as it is more aggressive at utilizing alternative paths. When an Interest is received, if there is no matching local caching, the priority of forwarding it to alternative untried next hop is higher than suppressing it. Interest suppression decision is made based per Face wise. This design also reflects to Interest forwarding. Given that each node has multiple next hops to retrieve data, all these next hops are used instead of only one next hop; the ratio of Interest forwarding on different paths stay almost the same in the five testing.

5.7 Off-the-grid Communication

One unique benefit of NDN is the native support for off-the-grid communication [20], meaning that NDN support applications to achieve infrastructure-free communication without ad hoc mechanisms supported. Although PCLive relies on existing web services to deliver entry website for consumer to use NDN, we have tested its off-the-grid communication capability in a local network. Specifically, we run the OBS and the producer side of PCLive on one machine, which generates livestreaming to a local NDN network; then we run the consumer part of PCLive on directly on another machine in the same local NDN network. In addition, self-learning mechanism [19, 32] is used to build a path in the local network. The consumer can retrieve and play the same livestreaming in the local network as it is over the Internet.

6 DISCUSSION

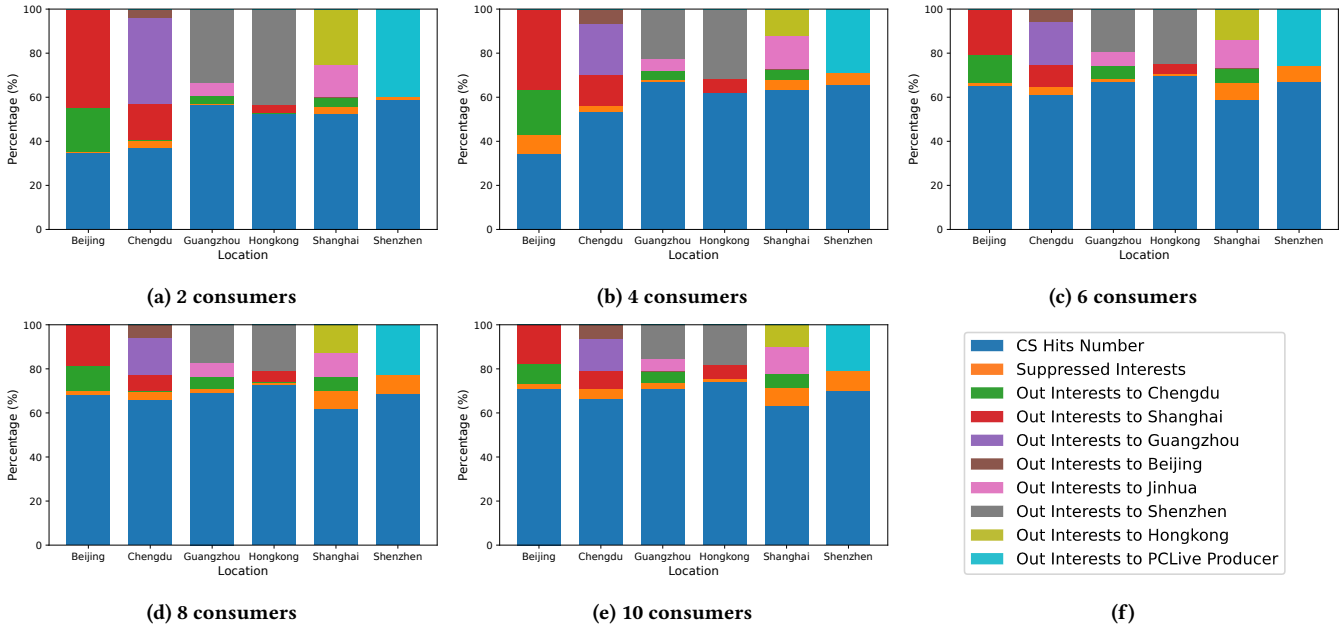
6.1 NDNizing low-latency livestreaming

In addition to HLS/NDN translation, we also investigated in NDNizing Low-latency HLS (LL-HLS) [22]. To reduce latency, LL-HLS allows smaller video chunks to be generated and retrieved, thanks to different video encoding and decoding techniques. Smaller video chunks do not affect how data is translated in NDN. Regarding protocol translation, LL-HLS allows conditional requests from clients and holding requests until responses are generated at servers. In NDN, we use Interests with application parameters added as tags to achieve conditional requests; and we implement wait-for-data logic at producers to achieve immediate data transferring. One concern is that this design remains in-network states for a longer time. The architecture impact requires deeper analysis and study, which is not within this work.

Two lessons are learned from translating LL-HLS/NDN. It once more proves the effectiveness of the NDNizing approach. Moreover, this approach can inherit good designs from existing application protocols, which are beyond non-networking components and have been tested in real world. For example, it is unnecessary to design low-latency livestreaming in NDN from scratch, and optimization can be made on top of LL-HLS/NDN translation.

6.2 NDNizing HTTP

NDN/HTTP translation is not a new idea [39]. Given that many modern applications are built on top of HTTP, this translation can significantly support massive application over NDN. However, a full-fledged HTTP is complicated to be translated into NDN. Though we have not dig into this direction, we foresee that a



(a) 2 consumers **(b) 4 consumers** **(c) 6 consumers**

(d) 8 consumers **(e) 10 consumers** **(f)**

Figure 11: Stateful forwarding behaviors on six nodes during a three-minute testing

HTTP/NDN translation proxy can be used to exchange between HTTP and NDN traffic.

Though HLS uses HTTP to retrieve data, our approach is not NDNizing HTTP. We are NDNizing how applications retrieve data. As long as an application is retrieving data using pull-based communication manner, it suppose to be easily NDNized, since NDN is designed to assist applications to retrieve data. Therefore, the NDNized communication part can be easily plugged into the application as long as the software is reasonably designed. If we take a step forward, as long as an application is using REST architecture, it may use NDN to retrieve data easily.

6.3 Comparison with CDN

Although we built a wide area testbed, and compare livestreaming over IP and NDN, we are aware that real-time content distribution is highly optimized in today's Internet. For example, CDN or SRS Edge Server mode can be applied to scale streaming distribution. In fact, there are several comparison and related work [6–8, 23, 36], and it is not our intention to make a comparison between NDN and any optimized approach in IP. Our focus is try to clarify and demonstrate the simplicity, generality, and effectiveness of our NDNizing approach.

7 CONCLUSION

This work focuses on NDNizing Internet livestreaming. Instead of building a proxy between an existing application and an NDN network to translate an application protocol and NDN, this work replaces the HLS communication module of a livestreaming player with NDN. The semantics and logic of HLS, i.e,. how to adaptively retrieve real-time video streaming, are inherited and unchanged. Surprisingly, this NDNization approach is simple and effective, as we build an NDN-capable system named as PCLive and run it in real world. We believe this a good example to demonstrate that modern applications are designed to retrieve data in a way that is naturally supported by NDN. In addition, we conducted some preliminary evaluation with PCLive deployed on a wide-area NDN overlay testbed. Comparing PCLive with livestreaming running over a single HTTPS&IP server under the same network conditions, PCLive serves 3.95 times as many clients as HTTPS&IP solution with 34.8% lower traffic. Moreover, PCLive also supports off-the-grid communication. Regarding development efforts, PCLive remains compatible with existing components including video players, OBS, and video transcoders, plugging in NDN communications between the transcoders and the players, which avoids reinventing the wheel.

8 ACKNOWLEDGMENTS

We are grateful for invaluable suggestions made by the Shepherd, Patrick Crowley, and all comments from anonymous reviewers. We also thank Min Li, Kun Wang, Xuhui Liu, and Sunao Yang for their engineering work. This material is based upon work supported by National Key R&D Program of China (2022ZD0115303). Any findings, discussions, and recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the sponsor.

REFERENCES

[1] 2013. *SRS(Simple Realtime Server).* https://github.com/ossrs/srs
[2] Alexander Afanasyev. 2021. *NDN-FCH (Find Closest Hub).* https://github.com/named-data/ndn-fch
[3] Alex Afanasyev, Jeff Burke, Tamer Refaei, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2018. A brief introduction to Named Data Networking. In *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM).* IEEE, 1–6.
[4] Alexander Afanasyev, Junxiao Shi, Beichuan Zhang, Lixia Zhang, Ilya Moiseenko, Yingdi Yu, Wentao Shang, Yi Huang, Jerald Paul Abraham, Steve DiBenedetto,

et al. 2014. NFD developer guide. *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0021* (2014).
[5] Elasticsearch B.V. 2022. *Search. Observe. Protect.* https://www.elastic.co/guide/index.html
[6] Ishita Dasgupta, Susmit Shannigrahi, and Michael Zink. 2022. A hybrid NDN-IP architecture for live video streaming: From host-based to content-based delivery to improve QoE. *International Journal of Semantic Computing* 16, 02 (2022), 163–187.
[7] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. 2020. Far cry: Will cdns hear ndn's call?. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*. 89–98.
[8] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. 2020. icdn: An ndn-based cdn. In *Proceedings of the 7th ACM Conference on Information-Centric Networking*. 99–105.
[9] Chavoosh Ghasemi, Hamed Yousefi, and Beichuan Zhang. 2021. Internet-scale video streaming over NDN. *IEEE Network* 35, 5 (2021), 174–180.
[10] Inc. Google. 2019. *Shaka Player JavaScript library.* https://github.com//shaka-player
[11] Miguel Grinberg. 2018. *Flask web development: developing web applications with python.* " O'Reilly Media, Inc.".
[12] Peter Gusev and Jeff Burke. 2015. Ndn-rtc: Real-time videoconferencing over named data networking. In *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. ACM, 117–126.
[13] John Hartman. 2014. *NDN Testbed.* https://named-data.net/ndn-testbed/
[14] Yusaku Hayamizu, Koki Goto, Masaki Bandai, and Miki Yamamoto. 2021. QOE-aware bitrate selection in cooperation with in-network caching for information-centric networking. *IEEE Access* 9 (2021), 165059–165071.
[15] Van Jacobson, Diana K Smetters, Nicholas H Briggs, Michael F Plass, Paul Stewart, James D Thornton, and Rebecca L Braynard. 2009. VoCCN: voice-over content-centric networks. In *Proceedings of the 2009 workshop on Re-architecting the internet*. 1–6.
[16] Derek Kulinski and Jeff Burke. 2012. Ndn video: Live and prerecorded streaming over ndn. *The NDN Project Team, Tech. Rep* (2012).
[17] Stefan Lederer, Christopher Mueller, Christian Timmerer, and Hermann Hellwagner. 2014. Adaptive multimedia streaming in information-centric networks. *IEEE Network* 28, 6 (2014), 91–96.
[18] Vince Lehman, Ashlesh Gawande, Beichuan Zhang, Lixia Zhang, Rodrigo Aldecoa, Dmitri Krioukov, and Lan Wang. 2016. An experimental investigation of hyperbolic routing with a smart forwarding plane in NDN. In *2016 IEEE/ACM 24th International Symposium on Quality of Service (IWQoS)*. IEEE, 1–10.
[19] Teng Liang, Ju Pan, Ashiqur Rahman, Junxiao Shi, Davide Pesavento, Alexander Afanasyev, and Beichuan Zhang. 2020. Enabling Named Data Networking Forwarder to Work Out-of-the-Box at Edge Networks.. In *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*. 1–6.
[20] Teng Liang, Ju Pan, and Beichuan Zhang. 2018. NDNizing existing applications: research issues and experiences. In *Proceedings of the 5th ACM Conference on Information-Centric Networking*. 172–183.
[21] Teng Liang and Beichuan Zhang. 2018. Enabling Off-the-grid Communication for Existing Applications: A Case Study of Email Access. (2018), 1–6.
[22] Teng Liang, Yang Zhang, Beichuan Zhang, Weizhe Zhang, and Yu Zhang. 2022. Low latency internet livestreaming in named data networking. In *Proceedings of the 9th ACM Conference on Information-Centric Networking*. 177–179.
[23] Ge Ma, Zhen Chen, Junwei Cao, Zhenhua Guo, Yixin Jiang, and Xiaobin Guo. 2014. A tentative comparison on CDN and NDN. In *2014 IEEE international conference on systems, man, and cybernetics (SMC)*. IEEE, 2893–2898.
[24] Spyridon Mastorakis, Peter Gusev, Alexander Afanasyev, and Lixia Zhang. 2018. Real-time data retrieval in named data networking. In *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, 61–66.
[25] Alexey Melnikov and Ian Fette. 2011. The WebSocket Protocol. RFC 6455. https://doi.org/10.17487/RFC6455
[26] Inc. MinIO. 2014. *Multi-Cloud Object Storage.* https://min.io/
[27] Inc. MongoDB. 2022. *MongoDB.* https://www.mongodb.com/
[28] Roger Pantos and William May. 2017. HTTP Live Streaming. RFC 8216. https://doi.org/10.17487/RFC8216
[29] Will Reese. 2008. Nginx: the high-performance web server and reverse proxy. *Linux Journal* 2008, 173 (2008), 2.
[30] Jacques Samain, Giovanna Carofiglio, Luca Muscariello, Michele Papalini, Mauro Sardara, Michele Tortelli, and Dario Rossi. 2017. Dynamic adaptive video streaming: Towards a systematic comparison of ICN and TCP/IP. *IEEE transactions on Multimedia* 19, 10 (2017), 2166–2181.
[31] Henning Schulzrinne, Anup Rao, Rob Lanphier, Magnus Westerlund, and Martin Stiemerling. 2016. Real-Time Streaming Protocol Version 2.0. RFC 7826. https://doi.org/10.17487/RFC7826
[32] Junxiao Shi, Eric Newberry, and Beichuan Zhang. 2017. On broadcast-based self-learning in named data networking. In *2017 IFIP Networking Conference (IFIP Networking) and Workshops*. IEEE, 1–9.
[33] Junxiao Shi and Beichuan Zhang. 2012. NDNLP: A link protocol for NDN. *NDN, NDN Technical Report NDN-0006* (2012).
[34] Xiaobin Tan, Lei Xu, Jiawei Ni, Simin Li, Xiaofeng Jiang, and Quan Zheng. 2021. Game theory based dynamic adaptive video streaming for multi-client over NDN. *IEEE Transactions on Multimedia* 24 (2021), 3491–3505.
[35] NDN Team. 2022. *nfd-status-http-server.* https://named-data.net/doc/NFD/current/manpages/nfd-status-http-server.html
[36] Rama Krishna Thelagathoti, Spyridon Mastorakis, Anant Shah, Harkeerat Bedi, and Susmit Shannigrahi. 2020. Named data networking for content delivery network workflows. In *2020 IEEE 9th International Conference on Cloud Networking (CloudNet)*. IEEE, 1–7.
[37] Lan Wang, Vince Lehman, AKM Mahmudul Hoque, Beichuan Zhang, Yingdi Yu, and Lixia Zhang. 2018. A secure link state routing protocol for NDN. *IEEE Access* 6 (2018), 10470–10482.
[38] Lijing Wang, Ilya Moiseenko, and Lixia Zhang. 2015. Ndnlive and ndntube: Live and prerecorded video streaming over ndn. *NDN, Technical Report NDN-0031* (2015).
[39] Sen Wang, Jun Bi, Jianping Wu, Xu Yang, and Lingyuan Fan. 2012. On adapting http protocol to content centric networking. In *Proceedings of the 7th international conference on future internet technologies*. 1–6.
[40] Fan Wu, Wang Yang, Ju Ren, Feng Lyu, Peng Yang, Yaoxue Zhang, and Xuemin Shen. 2020. NDN-MMRA: Multi-stage multicast rate adaptation in named data networking WLAN. *IEEE Transactions on Multimedia* 23 (2020), 3250–3263.
[41] Cheng Yi, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. 2012. Adaptive forwarding in named data networking. *ACM SIGCOMM computer communication review* 42, 3 (2012), 62–67.
[42] Yingdi Yu, Alexander Afanasyev, David Clark, Van Jacobson, Lixia Zhang, et al. 2015. Schematizing trust in named data networking. In *Proceedings of the 2nd International Conference on Information-Centric Networking*. ACM, 177–186.
[43] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. 2014. Named data networking. *ACM SIGCOMM Computer Communication Review* 44, 3 (2014), 66–73.
[44] Zhenkai Zhu, Sen Wang, Xu Yang, Van Jacobson, and Lixia Zhang. 2011. ACT: audio conference tool over named data networking. In *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. 68–73.