Real-Time Data Retrieval in Named Data Networking

Spyridon Mastorakis
UCLA
mastorakis@cs.ucla.edu

Peter Gusev
UCLA REMAP
peter@remap.ucla.edu

Alexander Afanasyev
FIU
aa@cs.fiu.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

***Abstract*—The Named Data Networking (NDN) architecture names and secures data directly at the network layer, thus enabling in-network data caching, which in turn facilitates large scale data dissemination. Applications fetch the desired data by names, and the data can come from either the original data producers or router caches. This data retrieval design works seamlessly when the applications know the *exact* names of data, but poses challenges for realtime applications, such as video and audio conferencing, where new participants may not know the exact names of the latest data production when they join. In this paper we present Realtime Data Retrieval (RDR), a simple protocol that enables applications to discover the latest data. Through prototyping and simulation-based study, we show that RDR can effectively retrieve realtime data with minimal additional delays.**

***Index Terms*—Named Data Networking (NDN), Information-Centric Networking (ICN), Realtime data discovery**

I. Introduction

The Named Data Networking (NDN) [1] architecture changes network communication from pushing packets to destinations identified by IP addresses to fetching application-named and secured data packets. Applications running over NDN can directly request data from the network without performing any additional mapping between their data names (e.g., URLs, file names) and the locations used by network delivery (e.g., IP addresses). By using data names, an NDN network can fetch the requested data from any available source, including the original data producer, router caches, or managed storage (repos).

NDN applications use a combination of inputs from users and naming conventions [2] to construct exact names to request data; when the exact names are unknown, an application may request data by using a name prefix, and the network will respond by fetching a data packet whose name has a matching prefix. A prominent example of the latter case is video conferencing, where data is being produced continuously, and a newly joined participant may not know the current video frame number to construct the exact name for the desired data. However, when requesting data using a prefix, the conferencing application needs to be aware that the network may return data from router caches, which were produced seconds or minutes ago. Therefore, the new participants need effective means to discover the latest video frame number in real time, so that they can properly catch up with the ongoing conference.

Generally speaking, solutions to this problem fall into one of the following approaches: (i) offering a network service (e.g., through a centralized server) to let consumers learn the names of the latest data, or (ii) assuming clock synchronization among all entities and realtime applications using timestamps as a part of the data names, so that consumers can construct desired data names according to their clocks. However, both approaches rely on the existence of network infrastructure that can ensure the availability of servers and keep all clocks synchronized.

In this paper, we present a simple design, called Realtime Data Retrieval (RDR), that can work in both infrastructure-based (e.g., the Internet backbone) and infrastructure-free (e.g., mobile ad hoc) environments. RDR utilizes NDN protocol features to let realtime applications discover names of the latest data, without assuming either infrastructure support or clock synchronization. As a tool for fetching data produced in real time, RDR only makes the natural assumption that both the producer (or its delegate) and consumer applications are online at the same time. To illustrate the basic mechanisms of the RDR design, we use NDN-RTC [3], a realtime video-conferencing library over NDN, as our application use case.

Our contributions in this paper are twofold: (1) we present the RDR design, which utilizes a few basic NDN protocol primitives to discover the latest data produced by realtime applications, and (2) we show our prototype implementation and simulation evaluation to confirm the effectiveness of RDR in helping realtime applications discover and retrieve the latest data.

The rest of this work is organized as follows: §II discusses some brief background on NDN and NDN-RTC, and prior related work. §III presents our design, and §IV our preliminary evaluation. Finally, §V summarizes what we have learned through designing the RDR protocol.

II. Background & Prior Work

In this section, we give a brief overview of the NDN architecture and the NDN-RTC application. We also discuss prior work on the development of realtime applications over NDN.

This work is partially supported by the US National Science Foundation under award CNS-1719403.

978-1-5386-4870-4/18/$31.00 ©2018 IEEE

A. NDN Overview

NDN enables applications to directly fetch data identified by a given name through a consumer-driven communication model. A consumer application sends requests (*interest packets*) for the desired data, which are forwarded toward the data producer(s) based on their names by NDN forwarders [4]. Once an interest reaches a node that has a *data packet* with the matching name or name prefix, this data packet is returned back to the consumer by following the reverse path of the corresponding interest.

To forward interests and the corresponding data packets, an NDN forwarder maintains three key data structures: (1) a Forwarding Information Base (FIB) that contains a number of name prefixes along with the outgoing interface(s) to direct interest forwarding, (2) a Pending Interest Table (PIT) that contains all the interests that have been forwarded, but the corresponding data packet has not been received yet, and (3) a Content Store (CS), where recently retrieved data packets are stored to satisfy future interests.

Figure 1 depicts the overall structure of an interest and a data packet, as defined by the NDN packet format specification [5]. An interest contains: (a) a required "`Name`" element that identifies the data to be fetched; (b) an optional "`MustBeFresh`" flag, indicating that the Interest can only be satisfied by a data packet that is be considered *fresh* based on its "`FreshnessPeriod`" value (see the next paragraph for definition); (c) another optional "`CanBePrefix`" flag to indicate that a data packet can satisfy this interest if only the prefix of a data packet's name equals the name carried in the interest (otherwise, names in interest and data packets must be equal); and (d) potentially a few other optional elements to guide Interest forwarding by intermediate forwarders.

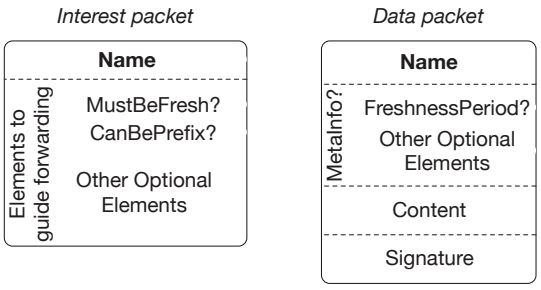

Fig. 1. NDN packet format

A data packet contains: (a) a required "`Name`" element that uniquely identifies the data packet,[1] (b) an optional "`MetaInfo`" section, including a "`FreshnessPeriod`" element that specifies for how long a data packet is considered fresh each time it is cached at a forwarder, (c) a payload, and (d) a "`Signature`" element that cryptographically binds together all the previous elements.

[1]In addition to the "exact" name carried in this field, data packets also have a logical "full name" that includes name in the data packet and additional "implicit digest" name component [5].

B. NDN-RTC

NDN-RTC [3] is a realtime video conferencing library for NDN, built on top of the WebRTC library [6]. An NDN-RTC producer collects audio samples and video frames from media inputs, segments them into data packets, and serves them in response to incoming interests. Figure 2 illustrates the structure of the NDN-RTC namespace. All NDN-RTC data packets are published under a producer-specific prefix. Audio/video streams are split into different stream types ("`mic`", "`cam`"), stream qualities ("`low`", "`mid`", "`high`"), and individual frames. Frames are further split into frame types ("`key`", "`delta`"), and each is assigned a sequential frame number. As frames usually do not fit into a single network MTU-sized packet, they are split into multiple segments under the "`data`" type. There is also a "`parity`" type, under which NDN-RTC publishes additional control information about the frame. In addition, NDN-RTC uses the "`[NDN-RTC prefix] /session-info`" namespace (not shown in Figure 2) to publish additional meta-information, such as the frame generation rate, and the estimate number of segments per frame type.

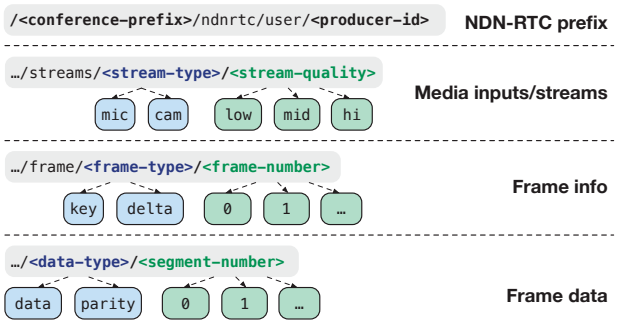
/<conference-prefix>/ndnrtc/user/<producer-id>
NDN-RTC prefix
…/streams/<stream-type>/<stream-quality>
mic
cam
low
mid
hi
Media inputs/streams
…/frame/<frame-type>/<frame-number>
key
delta
0
1
…
Frame info
…/<data-type>/<segment-number>
data
parity
0
1
…
Frame data

Fig. 2. NDN-RTC namespace

C. Prior Work

In the original design of NDN-RTC [3], newcomers discover the latest key frame using a "chasing" mechanism. More specifically, a new consumer C_{new} starts by expressing an interest with the name prefix of a key frame, e.g., "`/<NDNcall>/ndnrtc/user/<Spyros>/streams/cam/mid/frames/key`". This interest retrieves a data packet D for some segment of a frame x. Since D may be retrieved from a router cache, frame x may not be the latest one. To find the latest frame number, C_{new} sends interests for frames $x+1$, $x+2$, ... at a rate much higher than the frame generation rate (which is contained in the retrieved data packet D). By monitoring the inter-arrival gaps of the returned data packets, C_{new} can estimate whether it is receiving data from caches (data packets arriving at the interest sending rate), or has retrieved fresh data, i.e., the up-to-date key frame (when frames arrive at the expected frame generation rate instead of the interest sending rate). In the latter case, C_{new} starts retrieving delta frames for the fresh key frame. Whenever NDN-RTC consumers suspect a loss of synchronization, they repeat this chasing process again. Note that this "chasing" scheme can take a number of round

trip times and lead to large interest spikes during chasing. Also note that network congestions may interfere with the data packet arrival rate, reducing the accuracy of consumers estimates on whether they have caught up with the realtime data generation.

VoCCN [7] was among the first efforts to experiment with realtime applications over NDN. To provide interoperability with VoIP solutions, the standard VoIP protocols were encapsulated into NDN packets. However, this work did not address the issue of how to fetch the latest realtime data.

ACT [8] is another NDN audio conferencing tool. It uses naming conventions to enable users to discover ongoing conference calls and the data producers of each call. An audio data producer appends a monotonically increasing sequence number to the name of each generated data packet, and consumers fetch the desired data using the sequence number. When a newcomer C_{new} to a conference call does not know the latest sequence number of an audio data stream, C_{new} uses the conference name as a prefix to fetch the first data packet, whose name shows the sequence number n. C_{new} then sends subsequent interests with increasing sequence numbers after n at a rate higher than the regular voice data generation rate R. C_{new} catches up with the realtime data generation when the observed data arrival rate matches R, an approach similar to "chasing" that used in NDN-RTC.

Zhang et al. [9] proposed VR video conferencing over NDN through the use of a central signaling server that helps newcomers discover the latest data names. Each producer creates a notification for the first frame generated in every second and sends this notification to the signaling server. The server broadcasts the notification to all the consumers. A similar approach is also taken by Jangam et al. [10] in the implementation of a realtime multiparty video conferencing service. A network controller is responsible for handling "join" requests from newcomers and for providing namespace information to help them join the data generation.

Our approach does not assume the existence of network services to help consumers discover the latest data. As we describe in the next section, RDR uses a combination of metadata, naming conventions, and basic primitives from the NDN protocol to discover the latest realtime data within sub-RTT (round-trip time) scales.

III. Design

In the this section, we present the design of Realtime Data Retrieval (RDR) protocol, and use a simplified scenario based on NDN-RTC to illustrate the RDR protocol concepts.

A. Protocol Overview

To retrieve dynamically generated data, consumers must be able to deterministically construct the name for a desired piece of data. This requires that NDN applications name data in a systematic way. In NDN-RTC example, frames and frame segments are named sequentially, so that consumers, after seeing one piece of data, can construct names for future data by simply increasing the frame and frame segment numbers. However, to fetch realtime data with minimal delay, consumers must know what is the "most recent" frame number and how many interests need to be pipelined to achieve timely retrieval of all frames and frame segments produced in real time.

RDR provides the above necessary information to consumers through the use of *metadata* packets. A realtime producer publishes metadata about its data production, either periodically or in response to interests. To let consumers fetch the metadata in a timely manner in the presence of router caches, RDR makes a combined use of the "`FreshnessPeriod`" carried in metadata packets and the "`MustBeFresh`" in *discovery* interest packets (i.e., interests to fetch "fresh" metadata). As we explain in detail in subsequent subsections, the discovery interest may fetch the metadata packet from a router cache if the packet is still within its "`FreshnessPeriod`", otherwise the interest will fetch the metadata from the producer directly.

Once the metadata is retrieved, the consumer uses information from the metadata to infer the name of desired data to fetch (e.g., the next key/delta frame number $current + 1$, or viewing the frames produced previously), and the timing of interest packet transmissions. In NDN-RTC, the metadata provides sufficient information for consumers to send the necessary number of interests to retrieve all segments of the next key/delta frame.

B. Packet Formats

The key elements in RDR are *metadata* data packets (Figure 3) and *discovery* interests (Figure 4) that retrieve metadata.

The metadata data packets are named by attaching special word "metadata" and a version number to the application prefix (e.g., "`[NDN-RTC prefix]/metadata/version`"); the version number helps uniquely identify the produced metadata. At the minimum, the metadata includes the name of the latest generated data (in NDN-RTC, the name of the latest delta and key frames), i.e., the base name from which consumers can infer the names of future data. In addition, metadata may also include application-specific information about the data generation rate, the estimated number of segments per frame and frame type, and if the packet size permits, opportunistically carry a segment of the latest data.

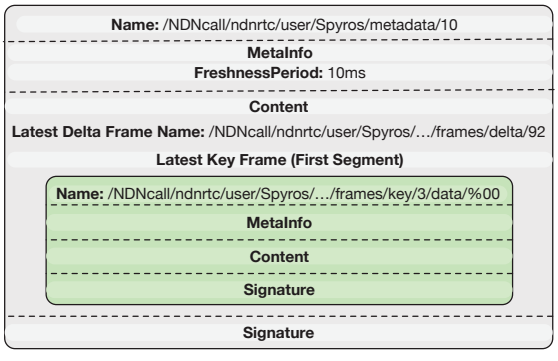

Fig. 3. NDN-RTC metadata example

A discovery interest aims to retrieve the latest version of the metadata. Its name includes only the metadata prefix

without the version number (as it is unknown), and it sets both "`CanBePrefix`" and "`MustBeFresh`" flags. The discovery interest either retrieve the latest metadata directly from the producer or a "fresh" version from in-network caches; "`CanBePrefix`" flag lets NDN forwarders satisfy an interest by a data packet whose name matches and extends the interest name with a version number component.

Fig. 4. NDN-RTC discovery Interest example

The discovery Interest name in the NDN-RTC example, as shown in Figure 4, refers to a conference call with prefix "`/NDNcall`" and to a producer with prefix "`/Spyros`". Its last name component indicates that this Interest is to fetch the producer's *"metadata"*.

C. Producer Side

When creating metadata packet, the producer selects an appropriate value for "`FreshnessPeriod`" under the "`MetaInfo`" section of the data packet. The "`FreshnessPeriod`" is a per-hop metric and specifies how long a data packet can be considered fresh when cached by an NDN forwarder.[2] The producer sets the "`FreshnessPeriod`" value based on how frequently it can handle interests for the metadata, while making sure that the metadata, when cached at forwarders, stay fresh enough to allow consumers catch up with the realtime data production. In NDN-RTC case, to avoid delta frame information lagging, freshness value of metadata should be no larger than half of the inter-delta frame generation interval and can be lower if producer can handle extra requests.

We emphasize that realtime data producer uses the "`FreshnessPeriod`" element carried in the metadata to inform consumers of the latest frame numbers. After retrieving the metadata, a consumer can fetch all desired data packets by using exact data names, independent from their freshness. For example, a latecomer to a conference call may want to fetch previously produced data, in addition to the new data being generated in real time.

D. Consumer Side

A consumer is responsible for expressing a discovery Interest to fetch the producer's metadata and measuring the round trip time (RTT). Upon receiving the metadata packet, the consumer discovers the latest generated data (within "`FreshnessPeriod`" plus network delay) and infers names of the data generated in the recent past and data that will be generated in the future. Thus, the consumer is able to request either previous or future key frame and delta frames using exact data packet names. In our example metadata packet (Figure 3), as soon as the consumer decodes its content, it deduces that it fetched the first segment (sequence number 0) of the latest key frame (key frame number 3). It also infers that the latest generated delta frame is number 92, which is the 6th delta frame generated for the latest key frame.[3]

At this point, the consumer knows: (a) the data generation rate and the average number of segments per frame (this information can be either published under the "`session info`" namespace or be included in the metadata's content by the producer), (b) the approximate RTT to the producer, and (c) the latest generated key and delta frame names. Therefore, the consumer can estimate when the next delta and key frames will be generated by the producer and decide when and how many interests to express to effectively fetch all frame segments, e.g., using the average number of segments per frame from metadata and the measured RTT estimate. Note that the metadata-based inference of when and which data is generated allows consumers to synchronize their interest transmissions accordingly, minimizing the number of interests staying in routers' PIT tables.

E. Baseline Example Scenario

To illustrate protocol operations, let us assume that the NDN-RTC *Producer* in Figure 5 starts generating data at time $T = 1$. After a few seconds ($T = 2$), *Consumer 1* comes online. Given that *Consumer 1* has no knowledge about the exact name of the latest data, it sends a discovery interest "`/NDNcall/ndnrtc/user/Spyros/metadata`" with the "`MustBeFresh`" and "`CanBePrefix`" flags enabled, which is forwarded to the producer (through forwarders *C* and *E*) and fetches the producer's metadata "`/NDNcall/ndnrtc/user/Spyros/metadata/0`".

Let us also assume that *Consumer 2* and *Consumer 3* come online around the same time at $T = 3$. *Consumer 2* starts first and expresses a discovery interest (again with name "`/NDNcall/ndnrtc/user/Spyros/metadata`" and "`MustBeFresh`" and "`CanBePrefix`" flags) that is forwarded to the *Producer* through forwarders *A*, *C*, and *E*. At forwarders *C* and *E*, the metadata retrieved by *Consumer 1* might still be cached, but because of 10 ms freshness period of the metadata packet with version 0 (Figure 3), it does not satisfy the interest. Therefore, the discovery interest from *Consumer 2* will reach the *Producer* and retrieve another version of the metadata ("`/NDNcall/ndnrtc/user/Spyros/metadata/1`"). The discovery interest of *Consumer 3* will follow the path to the *Producer* through forwarders C and E. This interest is either getting aggregated with the interest from *Consumer 1* or retrieves metadata from CS of forwarder C as it would still be fresh (i.e., within 10 ms since the arrival at forwarder C).

[2]Note that a forwarder starts counting down the "`FreshnessPeriod`" upon receiving a data packet. Therefore theoretically speaking, a data packet D with a freshness period s msec may still look "fresh" after $n \times s$ msec since its departure from the producer, if D has traveled through more than n hops and stayed at each hop close to s msec before being pulled to next forwarder. However in practice, such cases are unlikely.

[3]We assume that for each key frame, 29 delta frames are generated, and that the frame number for both key and delta frames starts from 0. Therefore, 93 delta frames have been generated in total ($93 \ mod \ 29 = 6$).

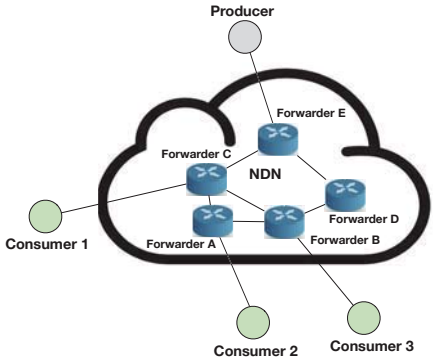

Fig. 5. NDN-RTC Latest Data Discovery Example

IV. Preliminary Evaluation

In this section, we show preliminary evaluation of our design through simulation-based study of a conceptual NDN-RTC client/server,[4] and a small-scale study of the prototype NDN-RTC conferencing application.[5] In our study we focused on (a) delays between realtime data generation and retrieval of data by consumers, (b) impact of different network delay, and (c) effects of in-network caching on data retrieval.

A. Simulation Setup

For our simulation evaluation we use ndnSIM module for NS-3 network simulator [11]. Given that the collection of shortest path from all consumers to a producer forms a tree, we use a simple 4-level tree topology with a conceptual NDN-RTC producer at the root node and NDN-RTC consumers at the leaves (Figure 6). All the other nodes in the topology act as NDN forwarders. The NDN-RTC producer generates 30 frames per second, and each delta and key frame consists of 5 and 30 segments respectively. The "`FreshnessPeriod`" of the metadata is set to 10 ms, and the metadata is generated upon requests (i.e., every time that a discovery interest reaches the producer). Each consumer has a random starting time in the range 1-2 seconds from the beginning of the simulation (Poisson distribution). The producer starts at time $T = 0$ seconds, and the simulation lasts for 30 seconds. We run our experiment in 4 different scenarios (Table I) and run each experiment 20 times.

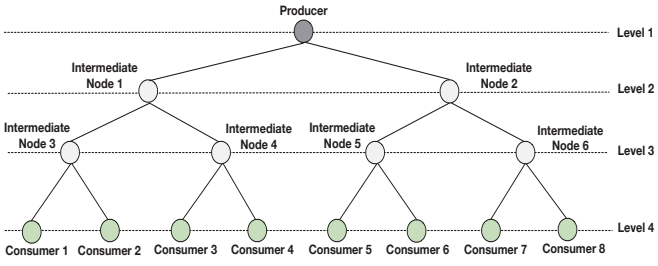

Fig. 6. Simulation topology

[4]https://github.com/spirosmastorakis/ndn-rtc-simulations
[5]https://github.com/remap/ndnrtc

TABLE I
Simulation scenarios

Scenario Case	Links	Link Delay	Scenario
1	All links	10ms	All links are fast.
2	Level 1 - 2	10ms	Producer links are fast. Consumers' link is slow.
	Level 2 - 3	10ms	
	Level 3 - 4	40ms	
3	Level 1 - 2	40ms	Producer links are slow. Consumers' link is fast.
	Level 2 - 3	10ms	
	Level 3 - 4	10ms	
4	Level 1 - 2	40ms	Producer and consumer links are slow.
	Level 2 - 3	10ms	
	Level 3 - 4	40ms	

B. Simulation Results

In Table II, we present the average and standard deviation of the time between the generation of frames by the producer and their retrieval by the consumers. The results show that our design enables NDN-RTC consumers to fetch the generated key and delta frames as soon as they are generated by the producer, achieving retrieval times close to the ideal case (i.e., the network delay between the producer and the consumer, which is equal to half of RTT). As we vary the link delay, the time between frame generation and retrieval does not change significantly. This demonstrates that our design provides an effective mechanism for consumers to discover the latest generated data in face of arbitrary network delays.

TABLE II
Simulations: Average and Standard Deviation of Time Between Frame Generation by the Producer and Retrieval by the Consumer

Scenario Case	Time between frame generation by the producer and retrieval by the consumer	
	ms	RTTs
1	30.11 ± 0.03	0.50 ± 0.001
2	60.50 ± 0.09	0.50 ± 0.001
3	60.61 ± 0.11	0.50 ± 0.001
4	91.11 ± 0.27	0.51 ± 0.002

In Table III, we present the average and standard deviation of the time between the generation of frames by the producer and their retrieval by each consumer for a varying metadata FreshnessPeriod (FP) value. The results show that when the FP is greater than the producer's generation period (which is about 33 ms for NDN-RTC), and a consumer fetches cached metadata, the retrieved metadata might contain stale information (i.e., an older key/delta frame name). This can lead to consumer lagging a few delta frames behind the producer. For metadata with a short "`FreshnessPeriod`", as presented in Table II, the retrieval of cached metadata does not impact the retrieval delay of realtime data.

C. Prototype Implementation Results

We integrated our design with the NDN-RTC prototype implementation and reproduced the topology of Figure 6. For rapid application deployment, we used docker [12] and deployed one docker container per node. The producer generates

TABLE III
SIMULATIONS: AVERAGE AND STANDARD DEVIATION OF TIME BETWEEN FRAME GENERATION BY THE PRODUCER AND RETRIEVAL BY THE CONSUMER

Scenario Case	Time between frame generation by the producer and retrieval by the consumer		
	FP = 30ms	**FP = 60ms**	**FP = 90ms**
1	0.50 ± 0.002	0.52 ± 0.006	0.53 ± 0.009
2	0.51 ± 0.002	0.52 ± 0.006	0.54 ± 0.010
3	0.52 ± 0.003	0.53 ± 0.008	0.57 ± 0.012
4	0.52 ± 0.004	0.54 ± 0.008	0.57 ± 0.012

a 320x240 video (1 KB bitrate). The consumers randomly join between the 1-2 seconds since the start of the experiment and start fetching the video. We ran the experiment 10 times for each scenario presented in Table I.

The results presented in Table IV show that the time between the generation of frames by the producer and their retrieval by the consumers is slightly higher than the time indicated in the simulation results. We estimate that this increase is mostly attributed to the processing delay added by intermediate forwarders and applications, which is not modeled in the simulation-based study. Overall, the prototype implementation results follow the same trend as the simulation results, and we verified that consumers fetch the frames as soon as they are generated.

TABLE IV
PROTOTYPE IMPLEMENTATION: AVERAGE AND STANDARD DEVIATION OF TIME BETWEEN FRAME GENERATION BY THE PRODUCER AND RETRIEVAL BY THE CONSUMER

Scenario Case	Average Measured RTT	Time between frame generation by the producer and retrieval by the consumer	
	ms	**ms**	**RTTs**
1	62.86	32.69 ± 0.170	0.52 ± 0.010
2	84.13	43.75 ± 0.273	0.52 ± 0.012
3	83.64	44.33 ± 0.329	0.53 ± 0.014
4	182.03	100.12 ± 0.936	0.55 ± 0.017

V. CONCLUSION

In this paper, we presented the design of the Realtime Data Retrieval (RDR) protocol which enables realtime applications to discover the names of latest data production, without requiring infrastructure service support or clock synchronization. In concluding the paper, we step up a level to share our insights in the solution development and to address a few questions that have been raised repeatedly.

First, retrieving realtime data across a network full of caches requires an effective means to limit the time that all valid responses can stay in router caches as they cross the network. RDR's main advantages over server-based solutions, e.g. [9], [10], include simplicity (no server/server configuration needed), robustness, and NDN's native scalability with the number of receivers through multicast delivery and in-network caching of metadata packets.

Second, RDR achieves its design goal by using two existing NDN protocol primitives: the "`MustBeFresh`" flag in interest packets and the "`FreshnessPeriod`" of data packets. It showcases the power of exploring the combination of protocol features to achieve desired application functions.

Third, data producers may view "`FreshnessPeriod`" in their data packets as an effective means to defend against overload. If a producer application sets the "`FreshnessPeriod`" of an outgoing data packet D to n seconds, it should not receive another request for D from the same neighbor node within n seconds, bar the case of caches running out of space.[6]

Finally, to answer a commonly asked question of whether a data packet *must* be removed from cache when its freshness period expires, we would like to point out that the lack of freshness (staleness) is a different concept from obsoleteness. A piece of stale data can still be useful to some applications. On the other hand, if a cache becomes full, it is up to the cache management policy to decide the preferences of data removals.

As next step, we plan to explore the use of RDR to serve other realtime applications running over NDN, such as augmented reality [13], and to identify further improvement of the protocol from different usage scenarios.

REFERENCES

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM Computer Communication Review*, July 2014.
[2] NDN Team, "NDN Technical Memo: Naming Conventions," *NDN Technical Report NDN-0022*, 2014.
[3] P. Gusev and J. Burke, "NDN-RTC: Real-time videoconferencing over Named Data Networking," in *Proc. of ACM ICN*, 2015.
[4] A. Afanasyev, J. Shi *et al.*, "NFD developer's Guide," NDN, Technical Report NDN-0021, 2015.
[5] "NDN Packet Format Specification," https://named-data.net/doc/ndn-tlv/.
[6] A. Bergkvist, D. C. Burnett, C. Jennings, A. Narayanan, B. Aboba, T. Brandstetter, and J.-I. Bruaroey, "WebRTC 1.0: Real-time communication between browsers," *Working draft, W3C*, vol. 91, 2012.
[7] V. Jacobson, D. K. Smetters, N. H. Briggs, M. F. Plass, P. Stewart, J. D. Thornton, and R. L. Braynard, "VoCCN: voice-over content-centric networks," in *Proc. of Workshop on Re-architecting the internet*, 2009.
[8] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang, "ACT: audio conference tool over Named Data Networking," in *Proc. of ACM SIGCOMM Workshop on Information-Centric Networking*, 2011.
[9] L. Zhang, S. O. Amin, and C. Westphal, "VR video conferencing over Named Data Networks," in *Proc. of Workshop on Virtual Reality and Augmented Reality Network*, 2017.
[10] A. Jangam, R. Ravindran, A. Chakraborti, X. Wan, and G. Wang, "Realtime multi-party video conferencing service over information centric network," in *Proc. of IEEE Conference on Multimedia & Expo Workshops (ICMEW)*, 2015.
[11] S. Mastorakis, A. Afanasyev, and L. Zhang, "On the evolution of ndnSIM: An open-source simulator for NDN experimentation," *ACM SIGCOMM Computer Communication Review*, 2017.
[12] "Docker Containers," https://www.docker.com/what-docker.
[13] J. Burke, "Browsing an augmented reality with Named Data Networking," in *Proc of ICCCN*, 2017.

[6]It is important to choose an appropriate "`FreshnessPeriod`" value. Ideally, it could be the time gap between the current data D_n and next data D_{n+1} production (i.e., D_n is no longer considered fresh when D_{n+1} comes). However, given "`FreshnessPeriod`" is a per-hop metric (§III-C), one must take into consideration the possibility of the same data packet being cached at multiple hops. A sound engineering value could be something between the inter-arrival gap of metadata interests that the producer can comfortably handle and half of the time period to the next data production.