VR Video Conferencing over Named Data Networks

Liyang Zhang, Syed Obaid Amin, Cedric Westphal
Huawei Research Center
Santa Clara, CA 95050
{liyang.zhang,obaid.amin,cedric.westphal}@huawei.com

ABSTRACT

We propose a VR video conferencing system over named data networks (NDN). The system is designed to support real-time, multi-party streaming and playback of 360 degree video on a web player. A centralized architecture is used, with a signaling server to coordinate multiple participants. To ensure real-time requirement, a protocol featuring prefetching is used for producer-consumer communication. Along with the native support of multicast in NDN, this design is expected to better support large amount of data streaming between multiple users.

As a proof of concept, a protoype of the system is implemented with one-way real-time 360 video streaming. Experiments show that seamless streaming and interactive playback of 360 video can be achieved with low latency. Therefore, the proposed system has the potential to provide immersive VR experience for real-time multi-party video conferencing.

CCS CONCEPTS

• **Computing methodologies** → **Virtual reality**; • **Networks** → *Naming and addressing*; *Network experimentation*;

KEYWORDS

Virtual Reality, Named Data Networking

ACM Reference format:
Liyang Zhang, Syed Obaid Amin, Cedric Westphal. 2017. VR Video Conferencing over Named Data Networks. In *Proceedings of VR/AR Network'17, Los Angeles, CA, USA, August 25, 2017,* 6 pages.
https://doi.org/10.1145/3097895.3097897

1 INTRODUCTION

AR/VR will grow to become a $100 billion market in 2025 according to a Goldman Sachs forecast. It is predicted that the deployment of 5G with faster networks with very short round trip times will enable the deployment and wide adoption of AR/VR applications.

However, on its own, 5G may not be sufficient to support AR/VR applications properly. The peak 5G bandwidth of 1Gbps and sub-millisecond RTTs could support some AR/VR applications. Yet, this level of performance may not be achieved by most users most of the times. We believe architectural support will be required to provide most consumers of AR/VR with the proper QoE.

ICN provides some benefits which will be useful for AR/VR. In particular, for a VR application, ICN offers native multicast support which allows multiple users to participate in a virtual or remote environment, multi-point to multi-point communication semantics, and potentially the sharing of the common tiles which compose a Field of View. As for AR, the ability of ICN to cache some content near the users could store enhancements that augment the reality locally.

To demonstrate the benefits of ICN for AR/VR, we consider here the implementation of such a VR application using NDN. Namely, we describe the implementation of an immersive 360 video stream over NDN. One potential application is to enable VR video conferencing, where the remote participant feels like he/she is in the room with the other participants. VR video conferencing is a challenging application with strict delay and scalability requirements, which makes it an interesting first step.

ICN works on a pull-based model i.e. a consumer needs to send out a request, called Interest, to fetch the desired content, called Data. Using pull-based model for implementing a real-time audio/video conferencing system, which has stringent latency requirements[1], requires re-engineering the existing IP-based solutions. Existing work in this regard, like NDN-RTC [7] work in a peer-to-peer model, hence may face scaling issues. Our solution follows the design of our previous work [3] that is based on specialized service edge routers, that can scale well above 40 participants.

Here, the goal is to check the feasibility of ICN-based architecture for VR traffic, not the orchestration of services. We therefore simplify the network architecture, at the cost of more configuration at the end-hosts. Our solution utilizes two channels – one for control signals and one for data communication. The control channel provides mechanisms

[1]Less than 150ms and 350ms, for audio and video traffic respectively [4].

to synchronize the consumer and producer state; whereas the data exchange leverages ICN features, resulting in bandwidth efficiency. The proposed conferencing system provides higher reliability (i.e., faster recovery from the transient network conditions) and better performance with respect to media name sync among participants.

The paper is organized as follows: in Section 2, we take a look at the related work. Then we present an overview of our system in Section 3. We highlight some design details in Section 4. We evaluated our implementation and provide some results in Section 5. Finally, Section 6 offers concluding remarks.

2 RELATED WORK

There has been some work on video streaming in ICN. RFC7933 [14] discusses porting current IP video mechanisms to ICN architectures and considers some of the research challenges in ICN-specific video streaming mechanisms. [6] considers some of the issues arising from the interaction of adaptive video streaming with caching in ICN.

There are fewer works in the literature focusing on real-time communication using NDN; especially in audio/video conferencing. Initial work in this regard like VoCCN [8] and ACT [15] address only real-time audio communication, but demonstrate how a pull based architecture like NDN can be used for real-time communication. In [2] NDNVideo is proposed, which is able to stream live and pre-recorded video over NDN. The work is further expanded in [13]. These works do not offer immersive video nor VR experience.

NDN-RTC in [7] provides a peer-to-peer based framework for real-time audio/video communication using WebRTC [1] . The proposed NDN-RTC works as a wrapper outside WebRTC. On the producer side, it takes the output of WebRTC and packetizes it in NDN-compatible fashion, and sends it out via NDN. On the consumer side, NDN-RTC depacketizes the received data and feeds it back to WebRTC. In this way, NDN-RTC provides a transparent solution for real-time communication over NDN, which does not require the developers to interact with low-level NDN details. However, the application-support features offered by NDN-RTC continue to rely on heuristic approaches implemented at the end hosts, such as measuring response time for Interests during bootstrap phase or during connection disruptions for the mobile consumers. On the other hand, our solution pushes many of these features towards the network as in-network services to reduce the end host complexity, which allows for better scalability performance.

3 SYSTEM OVERVIEW

Our objective is to design a real-time NDN-based VR video-conferencing. There are several goals:

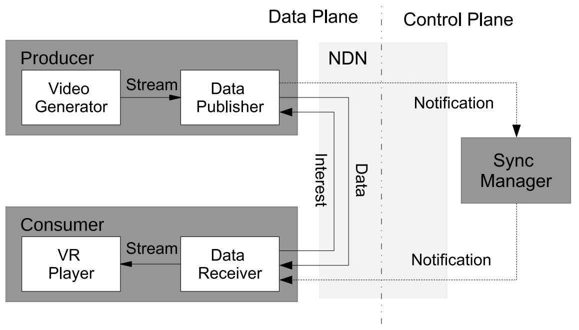

Figure 1: Framework of the NDN VR Conferencing System

(1) Support real-time video streaming between producer and consumer;
(2) Fully compatibility with NDN features, including application suitable naming, native multicast support, etc;
(3) Support seamless VR playback on a wide range of platforms;
(4) Support multi-party conferencing;
(5) Support rate adaption.

To this end, we propose a framework shown in Figure 1. The system comprises of three components, the producer, the consumer, and the sync manager, which are in charge of video generation, video playback, and signaling, respectively.

At the producer side, we use FFmpeg to capture and encode video on-the-fly and feed the stream to the buffer, from which the producer segments, packetizes, and names the data on a per frame base. Notifications about new chunks are sent out. It also responds to consumer-sent interests for video chunks by sending out the corresponding data.

The consumer receives notifications, and decides which chunks to fetch based on a prefetching protocol[2]. Interests for these chunks are sent in subsequence. Meanwhile, the received data chunks are sorted in correct order according to their names, and fed to the buffer. The VR player reads from the buffer, and renders the video in spherical (or equirectangular) mode. It also interacts with the user's behavior, such as selecting the Field of View (FoV) and zooming in and out.

The sync manager works as the signaling server. Its main job is to receive notifications about new video chunks from the producers and to broadcast them to the subscribing consumers. It is also in charge of user joining/leaving. When such an event happens, the involved user must send a notification to the sync manage, which then notifies other users so they can properly handle functionalities such as display window management.

There are several features of the designed system. First, it works in a centralized manner, in the sense that the sync manager coordinates all the participants. Compared with

[2]Prefetching means here that the Interests for the chunks are issued for chunks not-yet-created, to be delivered at a future time.

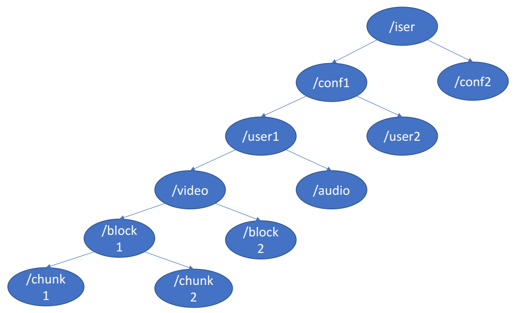

Figure 2: Naming scheme

peer-to-peer solutions such as NDN-RTC, this abstracts a distinct control plane, over which the management of a conference can be easily implemented. Second, on the data plane, we adopt a prefetching protocol that allows a consumer to "reserve" data chunks that are to be generated. In this way, unnecessary latency is avoided. Third, we adopt a lightweight VR player that is universally available, and easily substituted by other players.

4 DESIGN DETAILS

4.1 Namespace

In the proposed VR video conferencing applications, there are mainly two types of packets, namely, interest and data for video, in the data plane. In the control plane, there are notifications for events such as new video blocks, and joining/leaving requests. They are essentially interests with the message in their names. These interests do not have corresponding data, and no one responds to them. For clarity, we will refer them as notifications, and refer to interests solely as the "requests" for data, in the data plane.

We adopt the same naming scheme for video interest and data as in [9], shown in Fig. 2. In the name, the domain *iser* is a network prefix, referring to the "ICN Service Edge Router" (ISER) that hosts the current conference; domain *conf* stands for the instance of conference; *user* identifies an individual participator; *video* and *audio* denotes the media type; and *block* and *chunk* are indices for the data packet. In our implementation, a block corresponds to a frame of the video, which is segmented into small chunks, ordered in sequence. As an example, suppose the 5th frame of the video is produced by user 2, who is participating in conference 3 hosted on iser 1, has 3 segments, the 1st segment is then named as */iser1/conf3/user2/video/block5/chunk1*.

4.2 Real-time Communication Protocol

To reduce the latency and enhance real-time experience, we adopt the prefetching protocol for producer-consumer communication, in the same way as [3]. The protocol is illustrated in Fig. 3. Note that, notifications are not sent directly from producer to consumer, but relayed by the sync manager. But we omit the sync manager in Fig. 3, since only the end-to-end communication is the concern.

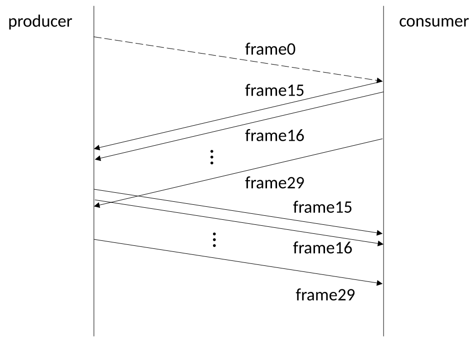

Figure 3: Prefetching protocol

To be specific, the producer sends out a notification about the first frame that is generated in a one-second period (frame 0 in the example shown in Fig. 3). Upon receiving the notification, the consumer sends out interests for the video chunks that will be generated in the *next* second, instead of those in the *current* second. For an agreed-on framerate, the block indices of the frames to be requested can be computed easily. For the example shown in Fig. 3, we assume that the framerate is 15 fps, i.e., 15 frames are generated in one second. So the consumer sends out interests for frames 15 to 29 in a bunch. Although the block indices (each block corresponds to a frame) is perfectly predictable, the consumer does not know how many chunks there will be for each frame in advance. Therefore, it guesses a number based on the expected frame size, and sends interests based on the guessed number.

When an interest is received by the producer, it first checks if the requested data has been generated. If so, the corresponding data is sent back to the consumer; if not, the interest is cached, and the corresponding data is sent back once generated. Since there may be more or less data generated than the number of interests expressed by the consumer, the producer announces the real number of chunks at the head of the first chunk of each frame. In this way, the consumer can remove redundant interests or send additional interests accordingly. Since it will result in latency for additional interests, it is generally favorable to allow some redundancy when guessing the number of chunks.

4.3 Producer

The block diagram of the producer is shown in Figure 4. The two main parts are the video generator and the data publisher.

The video generator produces a video stream playable at the receiver and packetizes it in a NDN-compatible way. We use a Ricoh Theta S 360 camera [10] as the input device,

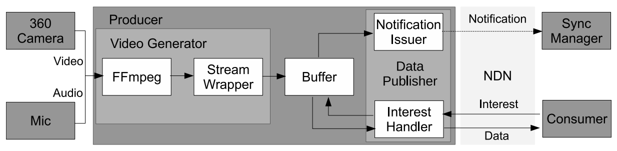

Figure 4: Block Diagram of Producer

which suports 15fps 720p motion-JPEG video output in dual-fisheye in live mode. The output of the camera is encoded and muxed using FFmpeg, a powerful multimedia framework supporting a vast variety of formats.

Compared to traditional video streaming, VR video streaming is expected to generate much larger amount of data, since it has to cover a wide range of view. Therefore, a good encoder should be chosen to encode the raw video. We use H.264 since it offers high quality with good compression rate. Besides, there is also a playback requirement that necessitates this choice (see subsection 4.4 for details).

To be specific, FFmpeg samples raw data from the camera 15 time per second, in accordance with the supported framerate of the camera. Each frame is encoded in H.264 format. Fragmented MP4 format is needed to support the increamental buffer and playback at the consumer side. Therefore, the video is further muxed in MP4 container, with the media streams fragmented into separated media "atoms" (elements in MP4 container). We have each frame in a separate atom, therefore, the output stream of FFmpeg is properly separated frame by frame.

The stream of encoded and muxed frames are piped to the stream wrapper, which wraps the frames into NDN-compatible packets. Since the frame size may exceeds the maximum allowed value for a NDN packet, it is segmented into chunks, each with its unique name according to the naming scheme. The chunks are then cached in a buffer, where the data publisher is able to retrieve them.

The data publisher handles communication with sync manager and consumer. Two functionalities are implemented. The notification issuer handles notifications. It creates a notification for the first frame generated in every second, and sends it to the sync manager so that it can be broadcast to the consumers. The interest handler, on the other hand, keeps track of the received interests. For interests that correspond to generated data, the data is retrieved from the buffer and sent back; for those correspond to data that has not been generated, the interests are cached until the data is generated.

Audio is processed in the same fashion, except there may be different input devices, sampling rates, and encoders. Logically, a separate stream can be maintained for audio. However, Ricoh Theta S 360 camera supports the recording of audio in parallel with video. Besides, MP4 container supports musing of multiple streams. Therefore, we are currently adopting an alternative method, i.e., have both video

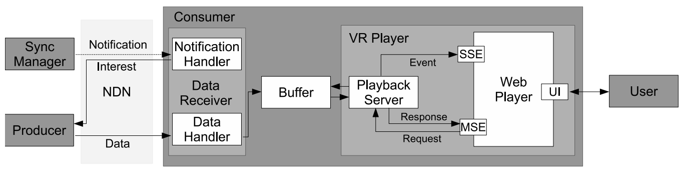

Figure 5: Block diagram of the consumer

and audio muxed together in one stream. The major benefit of this approach is the natural synchronization of video and audio at the encoding and muxing stage. There is also a possible drawback: the consumer has less flexibility in playback mode switching. But this can be easily worked around by generating a separate audio-only or video-only stream upon request. The prefetching mechanism implies that the producer is able to start the requested streams in a timely manner.

4.4 Consumer

The consumer comprise of two parts, data receiver and VR player, as shown in Figure 5.

The data receiver interacts with the NDN interface, receiving notifications from sync manager and conducting interest-data exchange with the producer. To be specific, the notification handler decides which data chunks to be requested according to the prefetching protocol. Note, in practice, due to network delay, the notifications may not arrive in time. The notification handler, therefore, checks the block index in the name of the notification, compare it with those of the sent interests, and decides if it is outdated. If so, the notification is simply discarded; if not, interests for the next second are sent to the producer to fetch the data. As described in 4.2, a number of chunks is guessed based on the expected frame size, and the interests are generated accordingly.

When a data packet is received, the data handler gets the block and chunk indices from the name and process it according to them. If it is the first chunk of a frame, the data handler read the number of chunks from it, since the protocol rules that the producer include the value at the start. Based on this value, data handler decides if there are redundant interests to be removed, or additional interests to be sent. Moreover, since chunks may arrive in an incorrect sequence, the data handler may need to cache and sort them in the correct order. Once all chunks for a frame are received and sorted, they are cached in the buffer, waiting to be played.

For a VR application, an important issue is the choice of player. Unlike players for normal video, a VR player must support VR features such as video stitching, FOV rotation, zooming and so on. The support of VR varies significantly for different state-of-art video players, which may hinder the usability of the proposed application. A lightweight, universally-available player that supports VR stream is therefore a preferable choice. For this we choose a web-based solution.

Ever since HTML5 has been introduced, it has become a popular solution for cross-platform multimedia playback. And since a HTML5-based web application can be easily ported to a hybrid mobile app using frameworks such as Apache Cordova, it has the further potential of covering mobile platforms. Consequently, a HTML5-based player is a good choice to meet our needs. In fact, there are already some open source HTML5-based 360 video players, such as Google's vrView [5] and a preliminary player offered by Ricoh [11], although neither of them supports streaming from a NDN source.

We develop our player based on the latter. The backbone of the player is a JavaScript. It handles the stitching of dualfisheye video to spherical video, the interaction with user behavior, and the playback control of the video element in the HTML page.

The main obstacle that hinders a HTML5-based player from supporting NDN-streaming is that the media fetching is done by the browser running the web application, and the way it fetches media is hidden. Therefore, even if the data receiver has fetched the video chunks to a local buffer, it is still not straightforward to make the player working with it.

To enable the player to stream from the buffer, we employ media source extension (MSE) [12] of HTML5. This extension allows javascript to generate media streams for a HTML media element, based on sequentially fetched data. Therefore, we implement in the player a data fetching agent that can communicate with the buffer, requesting newly added chunks from it. This is done using Asynchronous Javascript and XML (AJAX) requests and responses, for the following reasons: 1) because of the fluctuations in arrival times at the data receiver, and the processing time at the MSE, of different frames, the data fetching process of MSE is naturally an asynchronous process; 2) there is no native way for javascript to fetch data from another process, or rather, the native way for JavaScript to communicate with other processes is through AJAX; 3) although it is possible for JavaScript to read from local files, it is expected to be slower than to read from a buffer.

Therefore, we design a playback server to work alongside the web player. It responds to AJAX requests sent from the player with the corresponding video chunks retrieved from the buffer. Since the player is not aware of newly arrived video chunks, the playback server also handles notifications. This is achieved by implementing an event channel at the server, which pushes an event to the player once a new chunk arrives. The player uses server-sent-event (SSE) API to subscribe to this channel and get updates about new chunks.

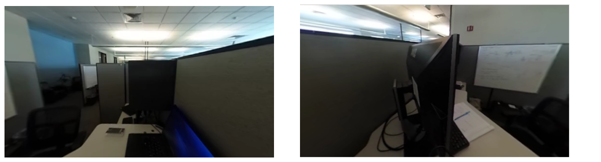
Figure 6: Rendered output at the consumer, with different viewing angles

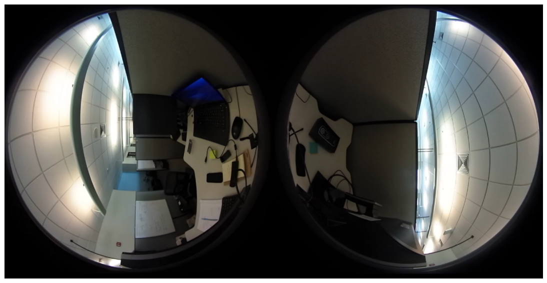
Figure 7: Generated video at the producer

AJAX requests are made once a new event is received. Interestingly, the interaction between the playback server and the web player, i.e., notification-interest-data iteration, resembles how the consumer interacts with sync manager and producer. This implies that, the NDN-related functionalities are hidden behind the playback server, from the player's point of view. Therefore, the proposed application can be easily migrated to other VR players.

5 IMPLEMENTATION AND EVALUATION

We implement a prototype of the proposed NDN VR video conferencing system. A producer and a consumer are hosted on two different computers running Ubuntu 16.04 operating system. The computer hosting the producer also runs the sync manager. In our preliminary testbed, the two computers are connected with an Ethernet cable, and Named Data Networking Forwarding Daemon (NFD) are run on both of them to emulate NDN forwarding. The NDN functionalities are achieved by using jndn library.

On the producer side, a Ricoh Theta S 360 camera is used to capture 360 video; on the consumer side, the VR player is run within the Mozilla Firefox browser. The generated video at the producer, and the rendered output at the consumer are shown in Fig. 7 and 6. Notice that the dualfisheye video is stitched to sperical at the VR player. The application successfully display the live video in real time. The latency between the generation of a video chunk at the producer and its reception at the consumer is shown in Figure 8. The data is collected from a test lasting 30 minutes, which also demonstrates the stability of our system for such length of time. The chunks are indexed according to the sequence with which they are generated. It is clear that, the latency for the

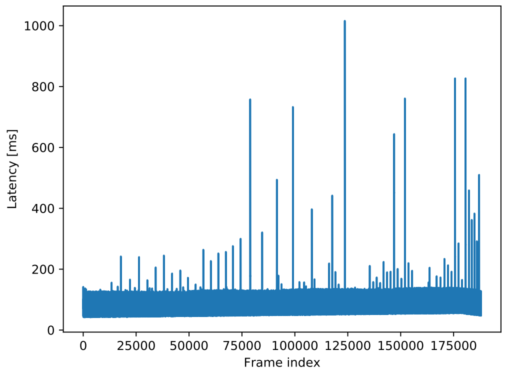

Figure 8: The latency from generation and delivery of video chunl

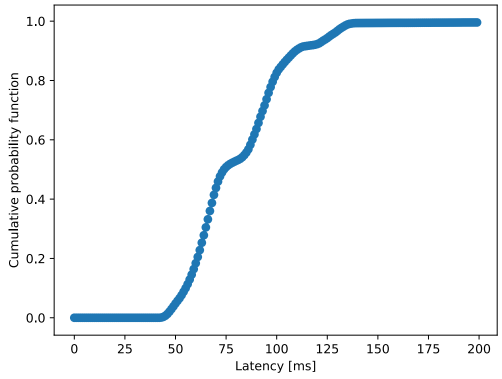

Figure 9: The cumulative distribution function (CDF) of latency for the video chunks

chunks is quite steady and remain under 120 ms for most of them.

To better show the performance in terms of latency, we plot the CDF of latency in Fig. 9. It can be observed that more than 90% of the chunks are delivered within 60 ms in our testbed. Expanding the network size would only add NDN networking delay, and would stay below our 350 ms target. All the results suggest that the proposed system has the potential to support real-time, multi-party VR video conferencing.

6 CONCLUSION AND FUTURE WORK

We have proposed a VR video conferencing system built atop NDN. The system is comprised of three components, namely, the producer, the consumer, and the sync manager. A centralized signaling structure is used, allowing the proper synchronization of the different users. To ensure the real time requirement is met, a prefetching approach is adopted, in which the consumers request video chunks in advance. For the playback of VR video stream, we designed a web player that is universally available. It is able to stitch the dualfisheye video to equirectangular format, and interact with user actions such as view angle rotation. Working with a consumer-side playback server, it has the potential to provide VR experience for a user on a browser.

We have implemented a prototype of the application, which currently supports one-way real-time live streaming of VR video. The latency has been verified to be small and consistent. In the future, we will add functionalities such as multi-user support, more intense VR experience, and head-mounted display using the Google Daydream framework. This application provides a good example of the potential of VR applications enhanced by ICN.

REFERENCES

[1] 2017. WebRTC. https://webrtc.org. (2017). [WebRTC homepage].
[2] Jeff Burke. 2013. Video streaming over named data networking. *E-LETTER* (2013).
[3] Asit Chakraborti, Syed Obaid Amin, Aytac Azgin, Ravishankar Ravindran, and Guo-Qiang Wang. 2017. SRMCA: A Scalable and Resilient Real-time Multi-party Communication Architecture. (2017). arXiv:arXiv:1703.03070
[4] Yan Chen, Toni Farley, and Nong Ye. 2004. QoS Requirements of Network Applications on the Internet. *Inf. Knowl. Syst. Manag.* 4, 1 (Jan. 2004), 55–76. http://dl.acm.org/citation.cfm?id=1234242.1234243
[5] Google. 2017. Google VRView. https://developers.google.com/vr/concepts/vrview. (2017). [Introduction of Google VRView Player].
[6] Reinhard Grandl, Kai Su, and Cedric Westphal. 2013. On the Interaction of Adaptive Video Streaming with Content-Centric Networking. In *IEEE Packet Video Workshop.*
[7] 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-ICN '15).* ACM, New York, NY, USA, 117–126. https://doi.org/10.1145/2810156.2810176
[8] 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 (ReArch '09).* ACM, New York, NY, USA, 1–6. https://doi.org/10.1145/1658978.1658980
[9] A. Jangam, R. Ravindran, A. Chakraborti, X. Wan, and G. Wang. 2015. Realtime multi-party video conferencing service over information centric network. In *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW).* 1–6.
[10] Ricoh. 2017. Ricoh Theta S 360 Camera. https://theta360.com/en/about/theta/s.html. (2017). [Introduction of Ricoh Theta S 360 Camera].
[11] Ricoh. 2017. Thetaview. https://github.com/ricohapi/video-streaming-sample-app/blob/master/samples/common/thetaview.js. (2017). [ThetaView Player].
[12] W3C. 2017. Media Source Extension. https://www.w3.org/TR/media-source/. (2017). [W3C Recommendation on Media Source Extension].
[13] Lijing Wang, Ilya Moiseenko, and Lixia Zhang. 2015. NDNlive and NDNtube: Live and prerecorded video streaming over NDN. *NDN, Univ. California, Los Angeles, CA, USA, Tech. Rep. NDN-0031* (2015).
[14] Cedric Westphal (Editor) et al. 2016. Adaptive Video Streaming in Information-Centric Networking (ICN). IRTF RFC7933, ICN Research Group. (Aug. 2016).
[15] 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 (ICN '11).* ACM, New York, NY, USA, 68–73. https://doi.org/10.1145/2018584.2018601