NDN-RTC: Real-Time Videoconferencing over Named Data Networking

Peter Gusev
UCLA REMAP
peter@remap.ucla.edu

Jeff Burke
UCLA REMAP
jburke@remap.ucla.edu

ABSTRACT

NDN-RTC is a videoconferencing library that employs Named Data Networking (NDN), a proposed future Internet architecture. It was designed to provide a platform for experimental research in low-latency, real-time multimedia communication over NDN. It aims to provide an end-user experience similar to Skype or Google Hangouts, while implementing a receiver-driven approach that takes advantage of NDN's name-based forwarding, data signatures, caching, and request aggregation. As implemented, NDN-RTC employs widely used open source components, including the WebRTC library, VP9 codec, and OpenFEC for forward error correction. This paper presents the design, implementation in C++, and testing of NDN-RTC on the NDN testbed using a demonstration GUI conferencing application, *ndncon*, which provides HD videoconferencing over NDN to end-users.

1. INTRODUCTION

Named Data Networking (NDN) is a proposed future Internet architecture that shifts the "thin waist" of the Internet from the current host-centric paradigm of IP to data-centric communication. In NDN, every chunk of data has a name, which is often hierarchical and human-readable, and a cryptographic signature binding name, data, and the key of the publisher. Consumers of data issue "Interest" packets for these "Data" packets by name. Signed, named Data packets matching the Interest can be returned by any node on the network, including opportunistic caches on routers. NDN's intrinsic caching can be leveraged by content distribution applications to reduce the load on data publishers in multi-consumer scenarios [8]. Duplicate Interests for the same content are also aggregated in routers, further reducing the load on those publishers and the network. NDN is described in more detail in publications on the project website[1], including [18, 19, 6].

[1] `http://named-data.net`

ICN'15, September 30–October 2, 2015, San Francisco, CA, USA.

ACM 978-1-4503-3855-4/15/09 ...$15.00.
DOI: http://dx.doi.org/10.1145/2810156.2810176.

Efficient content distribution has long been a driver application for NDN research and the broader field of Information Centric Networking (ICN). Prior work in this area, including our own, is covered briefly in Section 4. However, low-latency applications, such as "real-time" videoconferencing, present particular design and implementation issues that have not been as widely explored in publicly available prototypes or the NDN and ICN literature. For example, obtaining the "latest data" from a network with pervasive caching, without relying on direct consumer-producer communication (which impacts scaling potential) and while trying to keep application-level latency low, appears to be a significant challenge.

The NDN project team uses application-driven research to explore NDN's affordances for modern applications and to refine the architecture itself. The NDN-RTC library was created to explore real-time communications (RTC) experimentally. Though it is based on what was learned from our NDNVideo project [8], NDN-RTC is a clean slate design with new goals. The project is ongoing; this paper presents the current design and initial evaluation. To work towards the goal of using NDN-RTC in NDN project-related videoconferences and meetings, we needed reasonable CPU and bandwidth efficiency, echo cancellation, and modern video coding performance. Therefore, NDN-RTC is built in C++ for performance and leverages the widely used WebRTC library, incorporating its existing audio pipeline and video codec.

The remainder of this paper is organized as follows: Section 2 briefly describes potential NDN benefits for real-time communication (RTC). Section 3 details our goals for NDN-RTC. Section 4 covers background and prior work. Section 5 describes the architecture of the library, designed namespace, data structures and algorithms. Section 6 discusses implementation details. Section 7 evaluates main outcomes. Finally, Section 8 provides a conclusion and explains future work.

2. WHY USE NDN FOR RTC?

Given that NDN proposes a general Internet architecture, we are motivated initially to show its viability for applications beyond the content distribution examples most often discussed in the literature. However, we can also present a few potential benefits of using NDN for RTC that are exciting for us:

- By using names rather than IP addresses for routing and forwarding, as with any other NDN application, RTC applications stand to inherit the benefits for mobility, scal-

ability, and simplifications of network infrastructure that are currently being researched in the ICN community, a potential boon for RTCs applications. With NDN-RTC, we use a straightforward naming and communication scheme, leveraging conventions where possible, and build on the current libraries, forwarder, and testbed to increase the likelihood that these benefits can be inherited (or at least explored) in future work.

- Receiver-driven architectures requiring minimal publisher coordination can gain consumer scalability from the network, which is viable for streaming playout, as shown in past work discussed below. Early tests of NDN-RTC, described in later sections, demonstrate such network-supported scalability for RTC. This suggests that in the future, by using broadcast or group encryption schemes, NDN could efficiently support secure one-to-many or few-to-many low-latency broadcasts with very little additional application infrastructure – whether of entertainment content, presentations, closed-circuit cameras, computer vision sources, etc. – in addition to interactive conversations.
- Finally, because NDN-RTC builds directly on the thin waist of the NDN architecture, what is learned from exploring low-latency transmission of time series begins to provide broadly applicable insight into handling other high-rate and/or low-latency time series, such as sensor data feeds.

3. DESIGN OBJECTIVES

As discussed above, NDN-RTC aims to explore low-latency audio/video communication over NDN, and to support a working multi-party conferencing application that can be used by NDN project team members across the existing NDN testbed. It also aims to preserve network-supported scalability by avoiding direct consumer-producer communication (i.e., Interests that cannot be aggregated). Further, it explores if "real-time" communication can be achieved in a manner consistent with most other NDN data dissemination applications (described at a high level, as follows).

Applications using the library implement bidirectional communication by acting as a publisher of their own media streams and consumer of others. A publisher 1) acquires and transforms media data, 2) names, packetizes and signs it, and then 3) passes the packets to an internal or external component that responds to Interests received from the "black box" of the NDN network with signed, named data chunks. A consumer issues Interests with appropriate names and selectors to that "black box" of the NDN network, at the rate necessary to achieve its objectives and be a good citizen of the network[2]–as informed by the performance of the network it observes in response to its requests. It reassembles and renders them. Rate adaptation can be handled at the consumer by publishing in multiple namespaces corresponding to multiple bitrates or to layers of a scalable video stream and enabling the consumer to select them on-the-fly.

Once the namespace is defined, the publishing problem in this scenario (and in practice, at least so far) is relatively straightforward. Complexity is at the the consumer, which must determine what names to issue at what rate, to get the best quality of experience for the application. For real-time conferencing, this means low-latency access to the freshest data that the "black box" of the NDN network can deliver to a given consumer.

Conference setup and multi-party chat could be handled by applying techniques such as those developed in ChronoChat [16], which uses set reconciliation-based synchronization protocols to exchange messages in a many-to-many scenario.

Based on this high-level concept, specific design goals were developed for the NDN-RTC library:

- **Low-latency audio/video communication.** The library should be capable of maintaining low-latency (approx. 250-750ms) communication for audio and video, similar to consumer videoconferencing applications.
- **Multi-party conferencing.** Publishing and fetching several media streams simultaneously should be straightforward.
- **Passive consumer & cacheability.** There should be no explicit negotiation or coordination between publisher and consumer for the media transmission itself, to enable exploration of network-supported scaling to very high consumer to producer ratios.
- **Multiple bitrates.** The library and namespace should support multiple bitrates (from which consumers will select), enabling near future work on adaptive rate control.
- **Data verification.** The library should provide content verification using existing NDN features, as a building block for trust management and encryption-based access control.

Our design approach was further influenced by the relatively young state of NDN research:

- **Segment-level control.** At the protocol level, NDN-RTC works directly with data segments rather than video frames, group-of-pictures (GOP) blocks, or other higher-level constructs that it uses at the application level. This choice was made because most abstractions for most Interest-Data exchange explored in current research and available implementations do not handle low-latency, deadline-driven playout, assume large buffer sizes relative to network latency variations, and because frame sizes often exceed current NDN data object maximum sizes. While in the future, successful fetching and buffering patterns may be abstracted to a lower-level library, this approach enabled us to experiment with Interest expression and buffering at fine granularity. For example, NDN-RTC uses per segment metadata that can be exploited by the consumer to adjust fetching behavior.
- **Assumptions about the network.** Throughout the paper there are a number of assumptions about caching performance and other network behavior. Although future networks may have more complex behavior, the intention here is to explore the performance of basic assumptions on the NDN testbed, rather than proposing schemes to address the emergent behavior of complex ICN networks.

4. BACKGROUND AND PRIOR WORK

Most video streaming work on ICN has focused on playout without the constraints of supporting interactive conversations. For example, UCLA's NDNVideo, which supported live video and playback, was tested and deployed over NDN

[2]While work on congestion control and defining proper behaviour of such applications on the network is underway, it is future work with respect to this paper.

[8], scaling to approximately one thousand consumers from a single, simple publisher over plain-vanilla NDN. [4] Its data chunks were named sequentially according to NDN naming conventions [17], with a second namespace mapping sequence numbers to a timeline to enable efficient time-based random access by consumers. Though the project worked well for live and pre-recorded media streaming, it did not meet requirements for low-latency communication in its Interest pipelining approach. Also, its media architecture, based on GStreamer, was not immediately extensible for use as a conferencing solution. Subsequent work at UCLA, NDNlive and NDNtube [15] demonstrated a new API for application developers, the Consumer/Producer API [10], which works with higher-level "Application Data Units" rather than Interests/Data packets. To our knowledge, that work also does not target or achieve RTC, and like NDNVideo, is built on GStreamer, which does not easily support the audio pipeline needed for interactive conversations. Other streaming video work includes [9] and [12], which explore the advantages of using ICN networks for MPEG Dynamic Adaptive Streaming over HTTP (DASH). Although not directly related to low-latency streaming, these works also leverage ICN networks' caching ability for serving chunks of video files efficiently to multiple consumers. In contrast with the above, NDN-RTC was developed from the ground up with low latency and interactive conversations in mind.

The Voice-over-CCN project [5] was an early exploration of real-time communication over ICN, providing a similar level of quality compared to VoIP solutions with much greater scalability potential and simpler, more flexible architecture. VoCCN introduced pipelining Interests in a real-time scenario, an idea also employed in NDN-RTC. Subsequently, an audio conferencing application, ACT, was developed early in the NDN project [20]. It leveraged use of the Mumble library and successfully used NDN as a transport. Efforts for conference and user discovery were made in this work as well. However, echo cancellation quality was poor, which made it difficult to use, and only preliminary work in video was performed. This led us to build NDN-RTC on top of the WebRTC library, despite the additional, significant implementation complexity, in order to use its audio-processing capabilities and video codecs, and potentially give an opportunity for easier integration with supported web browsers.[3] Finally, the most recent related work in ICN-based real-time communication, of which we are aware, is [7]; however, as we understand, it currently handles Interest retransmission and buffering at the GOP level and does not (yet) meet our latency requirements.

5. APPLICATION ARCHITECTURE

There are two roles in the NDN-RTC library: producer and consumer. In bidirectional communication, applications use the library to play both roles, but a variety of other multi-party and one-to-many scenarios can be achieved. The library is built on a consumer-driven approach directly following NDN's Interest-Data exchange model. In contrast to the sender-driven approach of typical IP-based RTC, the producer publishes data to network-connected storage at its own pace, while the consumer requests data as needed and manages the relationship between outgoing Interests, incoming data segments, and buffer fill.

5.1 Producer

The producer's main tasks are to acquire video and audio data from media inputs, encode them, marshal the data into packets, *name* those packets, sign them, and store them in an application-level cache[4] that will asynchronously respond to incoming Interests. Flow control responsibility is shifted to the consumer, and scaling is supported by network caches downstream rather than publishing infrastructure at the application level.

5.2 Namespace

A primary design question is how the data should be named so it can be retrieved by the consumer with the desired properties. The NDN-RTC namespace defines names for media (segmented video frames and bundled audio samples), error correction data, and metadata, as shown in Figure 1. The namespace is designed to efficiently support consumer-driven communication, as introduced in Section 5 and detailed in Section 5.4. Note that NDN-RTC handles audio and video streams independently, which enables the library to support audio-only streaming, and for applications to prioritize audio over video for increased quality of experience for interactive conversations in bursty or low-bitrate scenarios.

5.2.1 Media

NDN-RTC employs the abstraction of a *media stream*, which describes a flow of one type of media, such as video frames or audio samples, coming from a source–currently, an input device on the producer. A typical publisher will publish several media streams simultaneously–e.g., video from a camera, audio from a microphone, video from screen capture. The data from a stream may be encoded at one or more bitrates, so in the name hierarchy, each stream has children corresponding to different encoder instances called *media threads*. Media threads allow the producer to, for example, provide the same media stream in several quality levels, such as low, medium and high, so that the consumer can choose the one suitable for its requirements and current network conditions.

NDN-RTC packetizes the WebRTC video encoder output directly. Encoded media segments published under the hierarchical names described above, with video frames further separated into two namespaces per frame type, `delta` and `key`, each numbered sequentially and independently. (Section 7 elaborates more on the reasons why this separation is needed.)

The next level in the tree separates data by type, either media or parity. Parity data for forward error correction, if the producer opts to publish it, can be used by a consumer to recover frames that miss one or more segments. In the case of both parity and regular frames, the deepest level of the namespace defines individual data segments. Any media packet, except audio, consists of one or more segments.[5]

[3]No modifications are needed to WebRTC [2] as used in NDN-RTC, which we believe will enable us to make comparisons between IP and NDN implementations of the library in the future.

[4]Currently, this cache is provided to the application by the NDN-CCL library.

[5]For example, the average sizes of frames for 1000 kbps stream using VP8/VP9: key frames are ≈ 30KB, and delta frames are ≈ 3-7KB. Therefore, depending on the under-

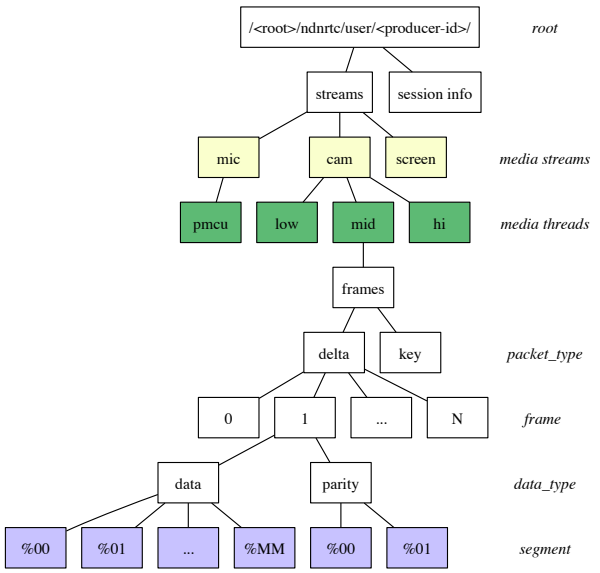
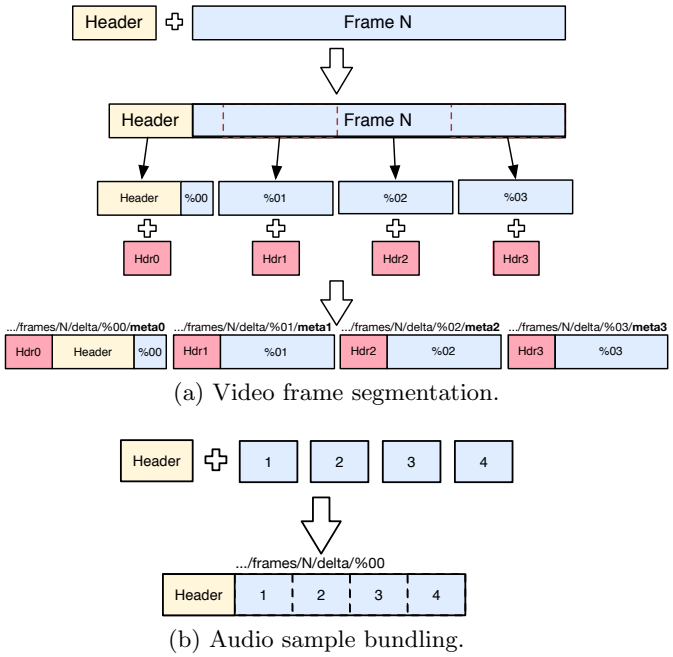


Figure 1: NDN-RTC namespace



(a) Video frame segmentation.

(b) Audio sample bundling.

Figure 2: Segmentation and bundling

These segments are numbered sequentially, and their names conform to NDN naming conventions [17].

Audio stream samples are much smaller than the maximum payload size, and there is no equivalent to the key/delta frame distinction in the audio codecs in use. Therefore, all audio packets are published under the `delta` namespace. Multiple audio samples are bundled into one data packet, until the size of one data segment is reached, and published only after that.

5.2.2 Metadata

NDN-RTC uses both stream-level and packet-level metadata. Consumers need to know the producer's publishing configuration and, to save them from traversing the producer's namespace, the producer publishes meta-information about current streams under `session info` and updates it whenever the configuration has changed.

Additionally, data names carry further metadata as part of each packet, which can be used by consumers regardless of which frame segment was received first. Four components are added at the end of every data segment name:

$$.../delta/frame\_no/data/seg\_no/n_{seg}/pl/pr_{seq}/par_{num}$$

n_{seg} - total number of segments for this frame;

pl - absolute playback position for current frame (this is different from the *frame*, which is a sequence number for the frame in its domain, i.e. `key` or `delta`);

pr_{seq} - sequence number of the corresponding frame from other domain (i.e., for delta frames, it is the sequence number of the corresponding key frame required for decoding);

par_{num} - number of parity segments for the frame.

lying transport's performance for delivering objects of this size, the producer may need to segment encoded frames into smaller chunks and name them in a way that makes reassembly straightforward. Based on our current observations of performance and the prevalence of UDP as a transport for the NDN testbed, NDN-RTC currently packetizes media into segments that are less than the typical 1500 byte MTU.

Metadata in the name, rather than in the packet, is expected to be useful for application components or services that may not need to understand the packet payload.

5.3 Data objects

The producer generates signed data objects from input media streams and places them into an in-memory, application-level cache. These objects contain stream data and metadata.

5.3.1 Media stream

Video stream data contains raw bytes received from the WebRTC library's video encoder. For audio, NDN-RTC captures and encapsulates RTP and RTCP packets coming from the WebRTC audio processing pipeline, in order to obtain echo cancellation and gain control and other features, which are then fed into a similar pipeline on the consumer side for proper rendering and corrections.[6]

5.3.2 Metadata

Apart from metadata provided in the namespace (as was described earlier), there is also metadata supplied in the data objects themselves. Every media sample is prepended with frame-level metadata, the *frame header* (see Figures 2 and 3), which carries encoding and timing information. Another type of metadata, the *segment header*, is appended to individual segments and carries the producer's observations of Interest arrival. This information is used by consumers to adjust fetching and playback mechanisms. Segment headers make use of the Interest nonce value, and thus may not be as useful in larger multi-party calls. At this time, they are used primarily for experimental and evaluation purposes:

[6]This is an artifact of the current implementation to benefit from the full audio pipeline of WebRTC, which is difficult to unbundle from RTP.

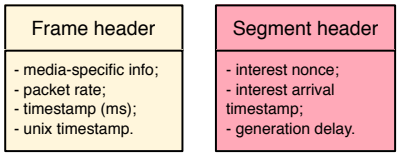


Figure 3: Frame and (experimental) segment headers.

Interest nonce: Nonce of the Interest *first* received at the publisher for a particular segment. Example interpretations include: 1) Value belongs to an Interest issued previously: Consumer received non-cached data requested by previously issued Interest; 2) Value is non-zero, but it does not belong to any of the previously issued Interests: Consumer received data requested by some other consumer; data may be cached; 3) Value is zero: Data requested after it has been produced; data is cached.

Interest arrival timestamp: Timestamp of the first Interest's arrival at the producer. Monitoring publisher arrival timestamps may give the consumer that issued the Interest information about how long it takes for Interests to reach the producer.

Generation delay: Time interval in milliseconds between Interest arrival and segment publishing. If the nonce is its own, a consumer can use this value in order to control the number of outstanding Interests.

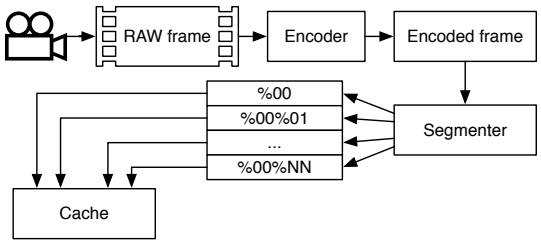
5.4 Consumer

In NDN-RTC's receiver-driven architecture, the consumer aims to 1) choose the most appropriate media stream bandwidth from those provided by the producer, e.g., by monitoring network conditions; 2) fetch and, if necessary, reassemble media in the correct order for playback; 3) mitigate, as far as possible, the impact of network latency and packet drops on the viewer's quality of experience. The consumer implements Interest pipelining and data buffering, as shown in Figure 4(b). An asynchronous Interest pipeline issues Interests for individual segments. Independently, a frame buffer handles re-ordering of packets, and informs the pipeline of its status to prompt Interest reexpression.

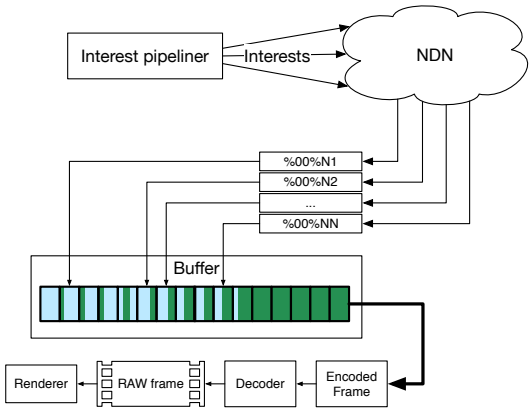
5.4.1 Frame fetching

The consumer uses an estimate of the number of segments it must fetch for a given frame, issuing M initial Interests, as illustrated in Figure 5. If Interests arrive too early, they will be held in the producer's PIT and stay there until the frame is captured and packetized. The delay between Interest arrival and availability of the media data is called the **generation delay**, d_{gen}. Conceptually, this interval should be kept low, to avoid accumulating outstanding Interests with short lifetimes; however, Interests should not arrive after data is published, as this increases latency from the end-user's perspective. Once the encoded frame is segmented into N segments and published, Interests $0-M$ are answered, and the Data returns to the requestor(s).

Upon receiving the first Data segment, the consumer knows from the metadata the exact number of segments N for the current frame, and issues $N-M$ more Interests for the missing segments, if any. These segments will be satisfied by data with no generation delay, as the frame has been published already by the producer. The time interval between receiving



(a) Producer



(b) Consumer

Figure 4: NDN-RTC producer and consumer operation.

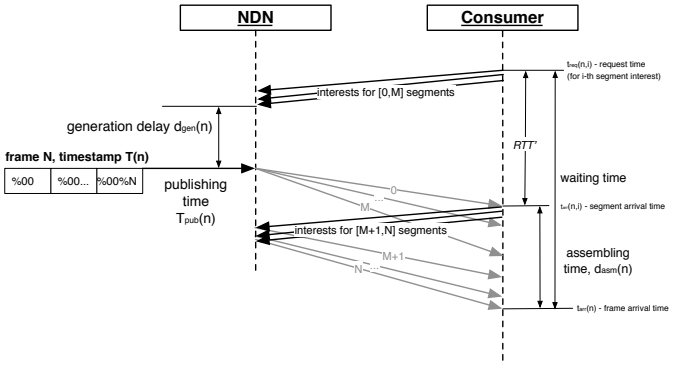


Figure 5: Data retrieval timeline.

the very first segment and when the frame is fully assembled is represented by d_{asm} and called **assembly time**. Note that for frames that have less segments than the estimate ($N < M$), some Interests may go unanswered.

More accurate estimates of the number of initial Interests per frame can help avoid additional roundtrips. The library estimates the average number of segments for each frame type for a given bitrate, and uses this evolving estimate to calculate the number of initial Interests to issue for the next frame in each namespace.

A similar process is used for fetching audio, though for now, audio bundles are carried by just one segment.

5.4.2 Buffering

The consumer uses a *jitter buffer* to manage out-of-order data arrivals and variations in network delay, and as a place

to assemble segments into frames. Our receiver-based paradigm requires the consumer to request data by name explicitly, and organize it by frame as well as segment. Outstanding Interests are represented in the buffer by "reserved slots" - those that have partial frame data or no data at all. The NDN-RTC jitter buffer's size is expressed in terms of two values measured in milliseconds at any given point in time. Its *playback size* is the playback duration in milliseconds of all complete ordered frames; its *estimated size* is *playback size* + *number of reserved slots* $\times$ $1/$*producer rate*, which reflects the estimated size of the buffer when all reserved slots have data. Each frame-level slot has an associated set of interests. The difference between estimated buffer size and playback size corresponds to the effective RTT, called RTT' (this cannot be smaller than the actual network RTT value).

Playout progress of the jitter buffer is used for retransmission control. At J milliseconds from the buffer end (see Figure 10) there is a checkpoint, after which it is estimated to be too late for another round trip. When a frame reaches the checkpoint, it is checked for completeness. If the frame is incomplete and cannot be recovered using available parity data, Interests for the missing segments are re-issued.

5.4.3 Interest expression control

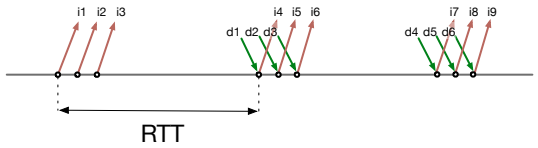
A key challenge of a consumer-driven model for videoconferencing, in a caching network, is how to ensure the consumer acquires the latest data without (per our design goal) resorting to direct producer-consumer communication. To get fresh data, the consumer cannot rely on flags in the protocol, such as *AnswerOriginKind* and *RightMostChild*. The frame period for streaming video is of the same order of magnitude as network round-trip time, suggesting there is no guarantee that the data satisfying those flags will be the most recent data received by the consumer. Instead, it is necessary to use other indicators to ensure that the consumer is requesting and receiving the most up-to-date stream data possible given its (potentially evolving) network connectivity.

Our current solution is to leverage the known sample publishing rate, which is available in stream-level metadata, and note that, under normal operation, old, cached samples are likely to be retrieved more quickly than new data. [7] We define the **interarrival delay** (d_{arr}) as the time between receipt of successive samples by a given consumer.

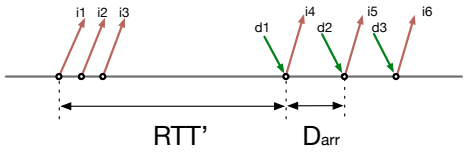
The library currently assumes that delays in the most recent samples follow the publishers' generation pattern, but older, cached data will follow the pattern of Interest expression.[8] Therefore, by monitoring inter-arrival delays of consecutive media samples and comparing them to the timing of its own Interest expression, which is distinct from the expected generation pattern, consumers can estimate whether they are receiving fresh data or cached data (see Figure 6). The consumer's objective is to obtain fresh data at a consistent rate from the network as a black box, not for Interests to "reach" the producer directly.

[7]If the consumer is the *only* consumer of the stream, its Interests will go directly to the publisher, which also yields the correct behavior. A more complex challenge, for further study, is when segments are inconsistently cached in different ways along the path(s) that Interests take.

[8]Though this assumption has proved successful in tests so far, we acknowledge more work is required to address more complex network conditions.



(a) Bursty arrival of cached data, which reflects Interests expression pattern and indicates that the data is not the latest.



(b) Periodic arrival of fresh data, reflects publishing pattern and sample rate.

Figure 6: Getting the latest data: arrival patterns for the cached and most recent data

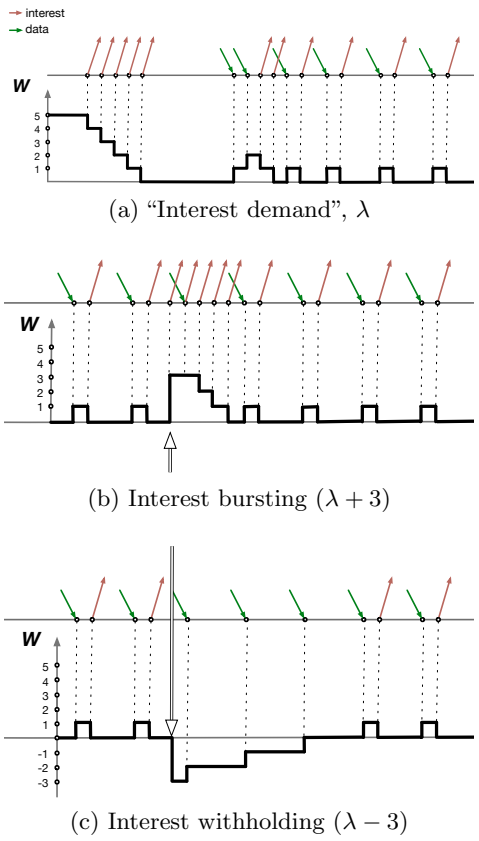
NDN-RTC interest expression is managed in two modes, *bootstrapping* and *playback*. During bootstrapping, the consumer "chases" the producer and aims to exhaust network cache of historical (non-real time) segments. By increasing the number of outstanding Interests, the consumer "pulls cached data" out of the network, unless the freshest data begin to arrive. In order to control Interest expression, the NDN-RTC consumer tracks a quantity called "Interest demand", λ, which can be interpreted as how many outstanding Interests should be sent at the current time (see Figure 7). The consumer expresses new Interests when $\lambda > 0$. For example, before the bootstrapping phase, the consumer initializes λ with a value which reflects the consumer's estimate of how many Interests are needed in order to exhaust network cache and reach the most recent data. In playback, every time a new Interest is expressed, λ is decremented, and when new data arrives, λ is incremented, thus enabling the consumer to issue more Interests.[9]

Bootstrapping. In the current design, there are two experimentally determined indicators that are used by the consumer to adjust λ: effective RTT (RTT') and inter-arrival delay d_{arr}. As described above, at bootstrapping (and re-acquisition), the consumer interprets d_{arr} stabilization around a relatively constant period, in order for the consumer to receive the freshest data available from the network. However, this does not necessarily ensure that the consumer issues Interests efficiently. Figure 8(a) displays that although the consumer has exhausted the cache rather quickly, RTT' is three times larger than the actual RTT for the network (100ms), which means that the majority of the issued Interests remain pending while waiting for the requested data to be produced.

The consumer makes several iterative attempts to adjust λ during bootstrapping, which can be described as follows:

1. The consumer initializes Interest demand with λ_d, and initiates Interests expression.

[9]While inspired by the TCP congestion window, the Interest demand, as currently employed in NDN-RTC, may play a different role in ICN networks, which we are exploring experimentally in this application.

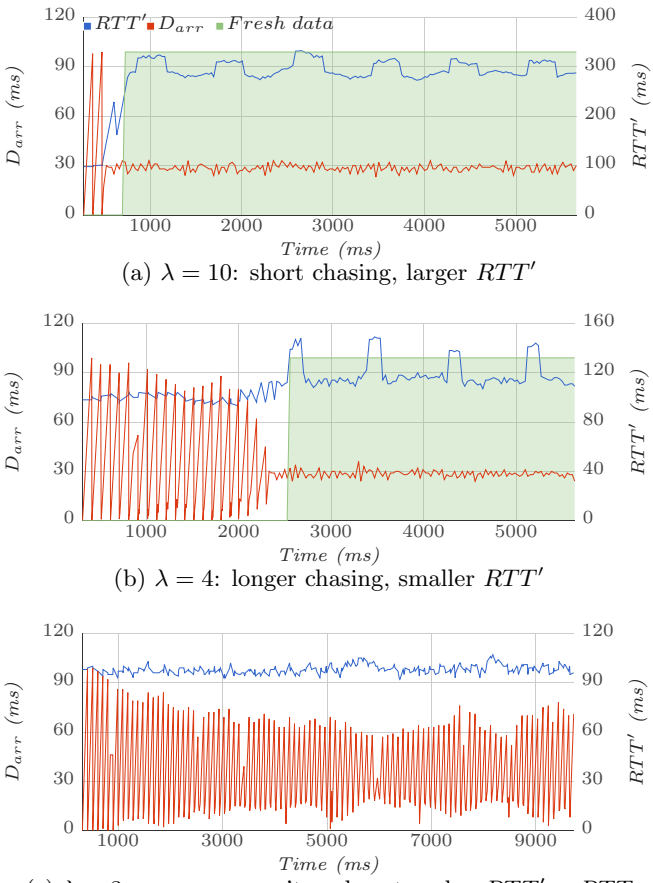


(a) "Interest demand", λ

(b) Interest bursting ($\lambda + 3$)

(c) Interest withholding ($\lambda - 3$)

Figure 7: Managing Interest expression



(a) $\lambda = 10$: short chasing, larger RTT'

(b) $\lambda = 4$: longer chasing, smaller RTT'

(c) $\lambda = 3$: consumer can't exchaust cache, $RTT' = RTT$

Figure 8: Larger λ decreases "chasing" phase, but increases RTT' for the same network configuration ($RTT \approx 100ms$)

2. If the consumer did not receive freshest data during the allocated time[10], it increases Interest demand: $\lambda = \lambda + 0.5\lambda_d; \lambda_d = \lambda_d + 0.5\lambda_d$.
3. Whenever the consumer receives data determined to be fresh (cache exhausted), it decreases Interest demand: $\lambda = \lambda - 0.5\lambda_d; \lambda_d = \lambda_d - 0.5\lambda_d$ and waits for one of two results: a) RTT' decreases and the consumer still receives the freshest data – repeat step 3; b) d_{arr} fluctuates unexpectedly, indicating cached data – restore previous value for λ_d, increase λ accordingly and stop any further adjustments as the consumer has achieved sufficient synchronization with the producer.

Note that λ is a counter of how many Interests can be issued more whereas λ_d represents the total number of outstanding Interests allowed. The value $\lambda_d - \lambda$ shows how many outstanding Interests consumer has issued at any given point in time. The way λ and λ_d are adjusted was determined empirically and may be a good topic for further research, along with how often to re-check that the consumer is obtaining the latest data through the steps above.

Bootstrapping begins with issuing an Interest with the enabled *RightMostChild* selector, in `delta` namespace for audio and `key` namespace for video (the video decoding process can start only with a key frame). Once an initial data segment of a sample with number S_{seed} has been received, the consumer initializes λ with initial value λ_d, and asks for the next sample data $S_{seed}+1$ in the appropriate namespace.

[10]In the current implementation, 1000ms.

Upon receiving the first segments of sample $S_{seed} + 1$, the consumer initiates the fetching process (described above) for all namespaces (`delta` and `key`, if available). The bootstrapping phase stops when the consumer finds the minimal value of λ, which still allows for receiving the most recent data, and the consumer switches to the playback mode.

Interest demand provides a manageable mechanism to speed up or slow down Interest expression, coupling the asynchronous Interest expression mechanism with the status of the playback buffer. An increase in λ value makes the consumer issue more Interests (Figure 7(b)), whereas any decrease in λ holds the consumer back from sending any new Interests (Figure 7(c)). Larger values of λ make the consumer reach a synchronized state with the producer more quickly. However, a larger value means a larger number of outstanding Interests and larger RTT' because of longer generation delays d_{gen} for each media sample. By adjusting the value of λ and observing inter-arrival delays d_{arr}, the consumer can find minimal RTT' value while still getting non-cached data, adapting towards a loose synchronization with the producer.

Playback. During playback, the consumer continues to observe RTT' and d_{arr}. Whenever d_{arr} indicates that no fresh data is being received, the consumer increases Interest

demand and starts the adjusting process over again to find minimal RTT' for the new conditions. Such an approach helps the consumers to adjust in cases when data may suddenly start to arrive from a different network hub which introduces new network RTT.

Interest batches. Practically, for video, the consumer controls expression of "batches" of Interests rather than individual Interests, because video frames are composed of several segments. λ is adjusted on a per-frame basis, rather than per-segment.

6. IMPLEMENTATION

NDN-RTC is implemented as a library written in C++, which is available at `https://github.com/remap/ndnrtc`. It provides a publisher API for publishing an arbitrary number of media streams (audio or video) and a consumer API with callbacks for rendering decoded video frames in a host application. NDN-RTC builds on functionality provided by other libraries. NDN-CPP [14] is used to access the NDN stack and to provide in-memory storage for the application. As discussed, the WebRTC framework [2] is used in two ways: 1) direct use of the video codec; 2) full incorporation of the audio pipeline, including echo cancellation. OpenFEC [1] is used for forward error correction support.

To demonstrate and evaluate the library, a desktop NDN videoconferencing application, *ndncon*, [3] was implemented on top of NDN-RTC. It provides a convenient user interface for publishing and fetching media streams, text chat, and organizing multi-party audio/video conferences. It was used, along with a command-line interface, for the evaluation below. The NDN-RTC library does not provide conference call setup functionality. This task was intentionally left out to be solved by applications that use it, and we are exploring it currently in *ndncon*. The MacOS X platform is currently supported; Linux build instructions will be added soon.

7. EVALUATION AND ITERATIVE REFINEMENT

Over the course of NDN-RTC development, numerous tests were run across the NDN testbed, as well as in isolated environments, to explore different library design patterns and implementations. These tests also helped us understand the nature of low-latency communication over NDN. We are still in the process of establishing well-defined metrics and test scenarios, but initial results generated refinements to our approach and are described below. There were several design iterations, and each introduced improvements in the overall quality of experience for the end-user, as well as in application efficiency related to bandwidth and computation. Each iteration tackled problems that were revealed during tests. These motivated namespace, application packet format and other revisions, which are reflected in the design detailed above.

7.1 Streaming performance

Separation of key and delta frame namespaces. Video streaming performance in early versions of NDN-RTC suffered from video "hiccups", even when being tested on trivial topologies. The cause of this problem turned out to be an inefficient frame fetching process. In early NDN-RTC versions, the difference in size, and thus segments, of key frames and delta frames was not reflected in the producer's

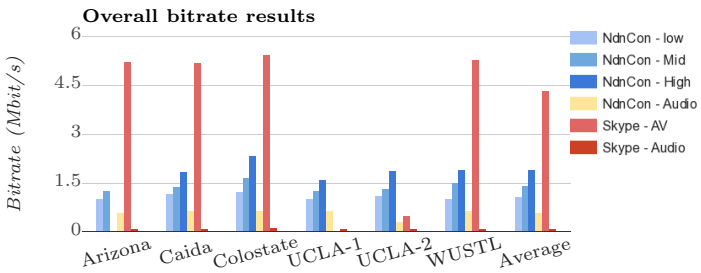


Figure 9: Two-peer conference tests compared to Skype

namespace, and consumers were forced to issue equal numbers of initial Interests (M), regardless of the frame type. This resulted in additional round trips of missing Interests and, consequently, larger assembling times (d_{asm}) for key frames that eventually led to missed playout deadlines and "hiccup" effects. Having a separate namespace for key frames enables consumers to maintain separate Interest pipelines per frame type and collect historical data on the average number of Interests required to retrieve one frame of each type in one round trip.

Audio sample bundling. Another set of tests targeting streaming performance was conducted over the existing testbed with a number of volunteers from the NDN community. Apart from monitoring application performance, we gathered user feedback and compared the experience with Skype. Each test was comprised of six runs of two-person, five-minute conference calls using *ndncon*: a) three runs of audio+video with low, medium and high video bandwidths settings (0.5, 0.7 and 1.5 Mbit/s accordingly); b) one run of audio-only conference; c) one run of Skype audio+video conference; d) one run of Skype audio-only conference. Tests were conducted between the UCLA REMAP hub and six other hubs. These tests covered both one-hop and multi-hop paths. As a main outcome of these tests, audio sample bundling was quickly introduced in NDN-RTC to reduce audio bandwidth (and the number of Interests on the consumer side), making it comparable to Skype audio bandwidths.[11] Figure 9 shows overall bitrate usage results before audio bundling was implemented. As expected, Skype adapted to use link capacity between peers, and delivered higher bitrate videos; leaving such adaptive rate control as our highest priority future work.

7.2 Consumer-Producer synchronization

Bootstrap behavior. In initial library versions based on the approach taken in NDNVideo, the consumer "chased" the producer's time-series data by exhausting cached data via issuing a large number of outstanding Interests. However, there was no mechanism to adjust Interest expression dynamically; the buffering mechanism dictated the Interests' lifetime: all Interests entering the buffer had a lifetime equal to half of the current buffer size. Thus, it was expected that data will arrive before the Interest times out. In these cases, the Interest is re-issued, even with a half-buffer length remaining to receive data before the playout dead-

[11] Currently, five audio samples are bundled together into one segment.

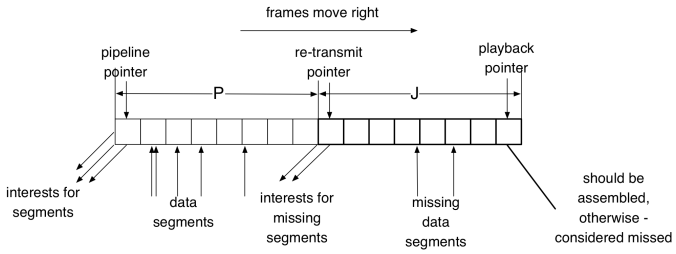


Figure 10: Frame buffer

line. This approach resulted in unavoidable Interest timeouts, in the cases when the consumer issued Interests far too early, before the actual data was produced. This was further complicated by forwarding strategies in the NDN Forwarding Daemon (NFD) that did not handle consumer-initiated retransmission over short periods. For two similar test runs (one-hop topology), the number of timed out Interests and re-transmissions varied greatly (either ≈1% or ≈50%). This problem was addressed by increasing the Interests' lifetimes[12] and the introduction of a new NFD re-transmission strategy that allowed early Interests re-transmissions. Additionally, the re-transmission checkpoint is now placed at a time estimated to be the effective *RTT* from the end of the buffer ($J = RTT$ on the Figure 10), which reflects a more accurate understanding of how data is received.

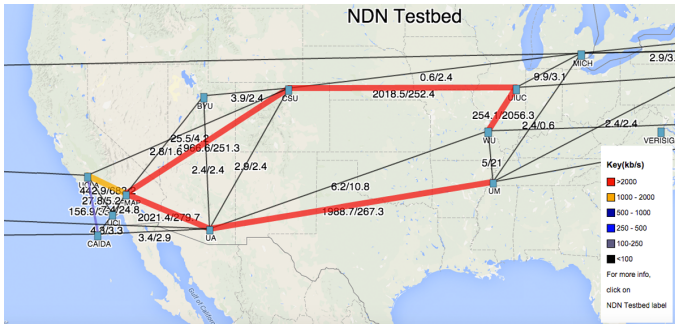
Moreover, the problem described above cannot occur if the consumer knows that it is issuing Interests too early. The chasing algorithm in older library versions was exhausting the network cache too aggressively; Interests were issued constantly until they filled up the buffer. With the introduction of the λ concept, the consumer exercises more precise control of the Interest expression as described previously. The number of outstanding Interests is controlled by a consumer and directly influences how fast consumer can "chase" the producer. Thus, the consumer is able to control the "agressiveness" of cache exhaustion and achieve a better synchronization state with the producer.

7.3 Multi-party use

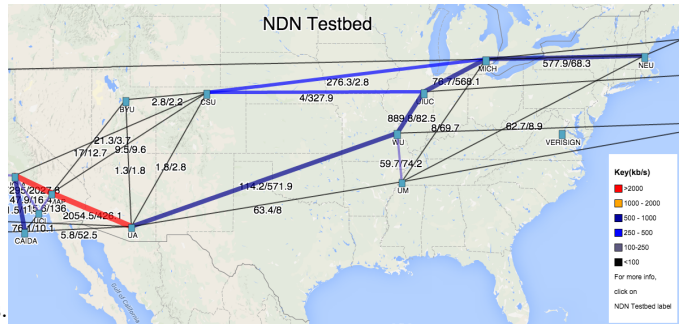
In another experiment, *ndncon* was used to stream an NDN seminar over the existing NDN testbed. An audio/video bridge was set up using third-party tools allowing captured screen and audio feeds from existing IP-based conferencing tools to be simulcast. (Screen broadcast is now supported natively in later versions of *ndncon*.) Figure 11(a) shows an example of instantaneous NDN testbed utilization during the one-hour conference call. It is estimated that media streams were consumed by five to eight people.

Other tests of multi-party conferencing ability included four peers, each publishing three video streams and one audio stream and fetching one video and one audio stream from each of the other participants. Participants were distributed across four NDN testbed hubs - UCLA, REMAP, CAIDA and WUSTL, as shown in Figure 11(b). Even though the user experience was satisfying and multi-peer conferences

[12]In fact, the dependence on Interests' lifetimes is not required anymore and every Interest is set to have 2000 ms lifetime.



(a) NDN testbed utilization during biweekly NDN seminar using *ndncon* for simulcast.



(b) NDN testbed utilization during 4-peer call between UCLA, REMAP, WUSTL and CAIDA hubs.

Figure 11: NDN testbed utilization during one-to-many and many-to-many scenarios.

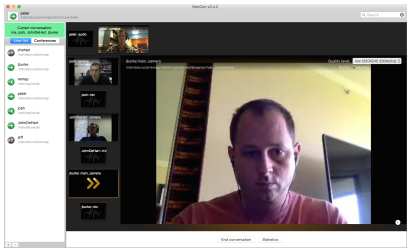


Figure 12: *ndncon* screenshot.

over NDN testbed have proven their viability, we plan more experimentation to explore quality of experience on a larger scale.

8. CONCLUSION AND FUTURE WORK

This paper presents the design, implementation, and initial experimental evaluation of NDN-RTC, a library intended to support experimentation in real-time communications over Named Data Networking. Our approach to this project has been experimentally driven so far, and has generated a functional low-latency streaming tool that we can now use as a platform for exploring important design challenges in real-time media over NDN. This is a rich area, and some of the future work that we have identified includes:

Scalability tests. NDN-RTC has shown that real-time communication using NDN is viable with the current open source implementation and on the current NDN testbed; we

are in the process of evaluating its performance in a variety of traffic scenarios and topologies. Our assumptions about the Interests and Data delivery patterns emerged from empirical observations of network behavior, and require more thorough experimentation in multi-peer scenarios, as well as simulation for much larger networks. The best schemes for 1) determining the "latest data" the network can provide at the correct rate and 2) congestion control remain open challenges that require collaboration between application developers, architecture researchers, and testbed operators.

Adaptive rate control. Multiple bitrate support is provided in the current design and implementation to lay the groundwork for adaptive rate control as a near-term effort, though for now the consuming application must manually select the best stream from bitrates offered by the producer. In ongoing co-development of an adaptive rate control solution, we are exploring if monitoring of d_{arr} and other approaches (described above) can address challenges of such adaptation over ICN, as suggested in papers such as [13].

Audio prioritization. For quality of experience in typical audio/videoconferencing applications, audio should be prioritized over video. This can be done at the application level but may also benefit from architectural support.

Scalable video coding. A more efficient way to relieve the producer from having to publish multiple copies of the same content at different bandwidths may be to use scalable video coding. By reflecting SVC layers in the namespace, the consumer will have more freedom for adapting media streams to the current network. Just as with audio, the SVC base layer may need to be prioritized; how to achieve this is an open question.

Inter-consumer synchronization. The absence of direct consumer-producer coordination shifted the complexity of "RTC-over-NDN" streaming to the consumer. A related requirement of modern videoconference not covered by this work is to ensure media playback sychronization across different consumers. This points more generally to the need for research on application-level time synchronization over NDN.

Encryption-based access control. The current NDN-RTC design supports basic content signing and verification. However, a prominent requirement of most videoconferencing is confidentiality, which can be supported in NDN through encryption-based access control. While encryption could limit the gains offered by caching, recent work exploring that application of advanced cryptographic techniques (such as attribute-based encryption to multimedia in ICN [11]) suggest new directions for meeting security requirements while leveraging key ICN features.

9. ACKNOWLEDGEMENTS

This project was partially supported by the National Science Foundation (award CNS-1345318 and others) and a grant from Cisco. The authors thank Lixia Zhang, Van Jacobson, and David Oran, as well as Eiichi Muramoto, Takahiro Yoneda, and Ryota Ohnishi, for their input and feedback. John DeHart, Josh Polterock, Jeff Thompson, Zhehao Wang and others on the NDN team provided invaluable testing of *ndncon*. The initial forward error correction approach in NDN-RTC was by Daisuke Ando.

10. REFERENCES

[1] OpenFEC Library. `http://openfec.org`.
[2] WebRTC Project. `http://www.webrtc.org`.
[3] NdnCon GiHub Repository. `https://github.com/remap/ndncon`, September 2014.
[4] P. Crowley. Named data networking: Presentation and demo. In *China-America Frontiers of Engineering Symposium*, Frontiers of Engineering, 2013.
[5] 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 *Proceedings of the 2009 workshop on Re-architecting the internet*, pages 1–6. ACM, 2009.
[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.
[7] A. Jangam, R. Ravindran, A. Chakraborti, X. Wan, and G. Wang. Realtime multi-party video conferencing service over information centric network. In *Proceedings of Workshop on Multimedia Streaming in Information Centric Networks (MUSIC) at ICME*, 2015.
[8] D. Kulinski and J. Burke. NDNVideo: random-access live and pre-recorded streaming using ndn. Technical report, UCLA, September 2012.
[9] S. Lederer, C. Mueller, and B. Rainer. Adaptive streaming over content centric networks in mobile networks using multiple links. *âĂę (ICC)*, 2013.
[10] I. Moiseenko and L. Zhang. Consumer-producer api for named data networking. In *Proceedings of the 1st international conference on Information-centric networking*, pages 177–178. ACM, 2014.
[11] J. P. Papanis, S. I. Papapanagiotou, A. S. Mousas, G. V. Lioudakis, D. I. Kaklamani, and I. S. Venieris. On the use of attribute-based encryption for multimedia content protection over information-centric networks. *Transactions on Emerging Telecommunications Technologies*, 25(4):422–435, 2014.
[12] D. Posch, C. Kreuzberger, and B. Rainer. Client starvation: a shortcoming of client-driven adaptive streaming in named data networking. *Proceedings of the 1st âĂę*, 2014.
[13] D. Posch, C. Kreuzberger, B. Rainer, and H. Hellwagner. Client starvation: a shortcoming of client-driven adaptive streaming in named data networking. In *Proceedings of the 1st international conference on Information-centric networking*, pages 183–184. ACM, 2014.
[14] J. Thompson and J. Burke. NDN Common Client Libraries. *NDN, Technical Report NDN-0007*, September 2012.
[15] L. Wang, I. Moiseenko, and L. Zhang. Ndnlive and ndntube: Live and prerecorded video streaming over ndn. Technical report, UCLA, 2015.
[16] Y. Yu. ChronoChat. `https://github.com/named-data/ChronoChat`.
[17] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang. Ndn technical memo: Naming conventions. Technical report, UCLA, July 2014.
[18] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang. Named data networking. Technical report, 2014.
[19] L. Zhang, D. Estrin, J. Burke, V. Jacobson, J. Thorton, D. K. Smetters, B. Zhang, G. Tsudik, K. Claffy, D. Krioukov, D. Massey, C. Papadopoulos, T. Abdelzaher, L. Wang, P. Crowley, and E. Yeh. Named data networking tech report 001. Technical report, 2010.
[20] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang. Act: audio conference tool over named data networking. pages 68–73, 2011.