IEICE TRANS. COMMUN., VOL.E99–B, NO.5 MAY 2016

INVITED PAPER *Special Section on Internet Architectures and Management Methods that Enable Flexible and Secure Deployment of Network Services*

Real-Time Streaming Data Delivery over Named Data Networking

Peter GUSEV[†a)], **Zhehao WANG**[†], **Jeff BURKE**[†], **Lixia ZHANG**[††], ***Nonmembers***, **Takahiro YONEDA**[†††], **Ryota OHNISHI**[†††], ***and*** **Eiichi MURAMOTO**[†††], ***Members***

SUMMARY Named Data Networking (NDN) is a proposed future Internet architecture that shifts the fundamental abstraction of the network from host-to-host communication to request-response for named, signed data–an information dissemination focused approach. This paper describes a general design for receiver-driven, real-time streaming data (RTSD) applications over the current NDN implementation that aims to take advantage of the architecture's unique affordances. It is based on experimental development and testing of running code for real-time video conferencing, a positional tracking system for interactive multimedia, and a distributed control system for live performance. The design includes initial approaches to minimizing latency, managing buffer size and Interest retransmission, and adapting retrieval to maximize bandwidth and control congestion. Initial implementations of these approaches are evaluated for functionality and performance results, and the potential for future research in this area, and improved performance as new features of the architecture become available, is discussed.

key words: *named data networking, information centric networking, videoconferencing, real-time, low-latency, congestion control*

1. Introduction

Supporting real-time streaming data (RTSD) dissemination with low latency is an important capability for modern networks. Videoconferencing, teleconferencing, live media broadcasting (i.e., future television and radio), streaming of sensor data, and control systems are examples of applications that require data to be distributed as it is produced (*real-time*) with minimal delay (*low-latency*). This paper discusses issues related to designing and building these types of applications, using Named Data Networking (NDN), a proposed future Internet architecture, with benefits unique to the capabilities of the architecture. The advantages and challenges of using NDN and other information-centric networking (ICN) architectures for content distribution and, more recently, supporting the Internet of Things (IoT) have been widely discussed. However, how these architectures can support RTSD applications is relatively unexplored.

From the perspective of application developers, this paper describes a general approach to receiver-driven RTSD that emerged from the development of several applications using NDN, including 1) a real-time videoconferencing application, as shown in Fig. 1; 2) a positional tracking data provider; and 3) a simple multimedia control system. These applications were built using the current open source NDN implementation and the NDN project team's testbed, which do not yet support all envisioned architectural features, including hop-by-hop congestion control which is critical to RTSD applications. By providing an initial design and implementation examples based on the currently available toolset, this paper aims to promote discussion of design issues related to RTSD applications over NDN, and to inform the ongoing design and implementation of the architecture itself.

Fig. 1 NDN-RTC as deployed for real-time streaming of the NDN Community Meeting 2015*.

1.1 Challenges of Real-Time Streaming Data

Though the current Internet is widely used for real-time streaming (in web-based videoconferencing, for example), the IP architecture's host-to-host model complicates the deployment of scalable, ubiquitous, and secure real-time streaming, with data consumed and produced by many nodes with intermittent and/or multi-path connectivity. Furthermore, typical streaming security models over IP are based on the notion of host-to-host sessions, making them brittle in the face of the intermittent, multi-path connectivity encountered in modern wireless networks, and complicating

Manuscript received January 4, 2016.
Manuscript revised January 22, 2016.

[†]The authors are with the Center for Research in Engineering, Media and Performance, University of California, Los Angeles, USA.
[††]The author is with the Department of Computer Science, University of California, Los Angeles, USA.
[†††]The authors are with the Advanced Research Division, Panasonic Corporation, Kadoma-shi, 571-8501 Japan.
a) E-mail: peter@remap.ucla.edu

DOI: 10.1587/transcom.2015AMI0002

*https://www.caida.org/workshops/ndn/1509

important applications such as mobile conferencing or vehicular sensing[†]. Also, current RTSD approaches over IP tend to treat real-time and historical data differently, e.g., using sender-driven WebRTC for real-time communication and receiver-driven HTTP-DASH for recorded video, which complicates application development.

Looking forward to future applications, the architectural limitations mentioned above make it difficult to achieve streaming designs that enable consumer-side content selection with low latency and scalability, as will be required for important emerging use cases, such as navigable video and progressive, on-demand delivery of virtual and augmented reality content–both of which make ongoing content selection and adaptation based on user interaction.

This paper is organized as follows: Section 2 describes architectural affordances and generalized design objectives for RTSD over NDN; it also outlines related work. Section 3 proposes a receiver-driven approach for RTSD over NDN, which is expanded in Sect. 4 via specific application use cases. Initial evaluation results are discussed in Sect. 5. Finally, the paper is concluded and future work is presented in Sect. 6.

2. NDN Affordances & Opportunities

NDN shifts the "thin waist" of a network from the host-centric communication model of IP to a data-centric, information dissemination model. It is a prominent example of ICN [1]. Communication over NDN employs two types of packets: *Interest* and *Data*. An application wishing to consume data sends Interests for named Data packets to retrieve from the network. Each forwarding node that receives these Interests forwards them to the next hop based on a longest-prefix match against its name-based Forwarding Information Base (FIB). It also stores the incoming interface from which it received the Interest in a Pending Interest Table (PIT), where it also aggregates duplicate Interests. When an Interest reaches a node with matching Data, whether an original producer, in-path cache, or any other source, the hop-by-hop state in the PIT is used to return the matching Data–along the original path back to the requesting node(s). Each Data packet is signed, providing an important building block for data-centric security. For a complete description of the architecture, please see [2], [3].

Within a research context, NDN has been applied to applications in live and prerecorded video streaming [4], lighting control [5], streaming sensor data [6], "big data" distribution in science [7], file sharing [8], [9], mobile health [10], building management [11], IoT [12], vehicular networking [13], and other areas.

NDN also addresses many of the significant challenges of designing and deploying large-scale RTSD applications that are encountered over IP. The following affordances motivated the research team's initial application development:

- **Consumer scalability.** Through its intrinsic multicast support, as a result of architectural features such as Interest aggregation and Data caching, NDN can enable RTSD applications to scale in the number of consumers based on network capacity rather than producer capacity, if the work required at the producer for each new consumer is appropriately limited by the application design.[††] Leveraging this benefit, constrained producers, such as mobile phones, could provide real-time data streams to a large number of consumers, provided there was efficient multicast delivery and sufficient cache capacity upstream of the producers.
- **Producer scalability.** Redundancy and scalability of producers can be achieved by leveraging name-based forwarding to distribute Interests to one or more producers transparently. For example, multiple nodes can capture and stream the same live video feed and listen for the same Interests, enabling hot failover. No action at the consumer side is required to switch from a normal to failover operation. More generally, different hosts can provide different subsets of content (e.g., different bitrates) transparently to the requesters, even in real-time streaming.
- **Random access.** Through name engineering, NDN-based RTSD applications can also support efficient random access to content–in time and other dimensions. For example, a namespace design expressing both temporal and spatial characteristics of data can enable augmented reality applications that must navigate real-time streams in both time and space. Related techniques are preliminarily explored in [15].
- **Per-packet verification.** NDN's per-packet signatures enable applications to achieve these advantages (above) while still being able to verify trust in individual data objects based on their signatures, using schemes such as [16]. By performing trust chain verification asynchronously and caching the results, applications can offer fast-start options without the equivalent of TLS/SSL session overhead, while HMACs and other techniques can provide computationally efficient alternatives to asymmetric cryptography for confidentiality.
- **Storage friendliness with access control.** It is important to note that host-to-host session semantics and security can also be achieved as a special case in NDN. However, by using broadcast encryption or similar techniques, even access-controlled content can be stored (and cached) efficiently, as discussed in preliminary work such as [17]. Furthermore, data in RTSD can be made available for historical access through techniques such as including a timestamp component in the namespace at design time. Then, objects can simply be stored in a repository when they are generated, and accessed by consumers based on the time component in the name.

[†]While these issues are not unique to RTSD, real-time applications' requirements make it difficult to solve these problems effectively at the application layer.

[††]This was demonstrated in [14] for streaming video playout over NDN using one NDNVideo [4] producer to supply 1000 consumers connected to the NDN testbed.

2.1 Related Work

The research team's approach has been informed by previous work over both ICN and IP architectures, the most relevant of which is described here.

2.1.1 RTSD over ICN

One of the earliest attempts of RTC over an ICN network was made in [18]. This work described the general feasibility of low-latency communication over ICN and its proposed advantages over IP in terms of security and scalability. Some important insights into RTSD application design were discussed, namely, a) *constructible names*, which allow consumers to generate exact Interest names for the future data based on packet names of previously received data, and b) Interest pipelines to minimize delay. These ideas were further elaborated in the NDN team's Audio Conference Tool (ACT) [19], which also explored rendezvous protocols for managing participants in a conference. ACT uses a *data delivery* namespace for audio transmission and a *multicast namespace* to support conference announcement and bootstrapping. The first fully functional videoconferencing application was NDN-RTC [20], later versions of which are discussed in Sect. 4. It demonstrated that high-definition audio-video multiparty conferencing is viable on the current NDN testbed, and provided a platform to explore challenges of minimizing latency, Interest expression patterns, error correction, and buffer control.

2.1.2 IP-Based Streaming

Media streaming, for both conversational and playout applications, is a well-studied area for the IP architecture. However, current IP-based streaming work, such as the buffer-based rate adaptation of [21], assumes a host-to-host unicast approach to streaming that is not applicable to a general NDN design, especially given the aim to leverage the architecture's unique affordances. In the interest of space, all such approaches are not surveyed here. Of more direct relevance are previous receiver-driven approaches developed to leverage multicast, such as Receiver-driven Layered Multicast (RLM), introduced in [22], extended in a variety of other work, and surveyed later in [23]. In RLM, a sender employs layered coding and sends different data layers to separate multicast groups. This allows heterogeneous receivers to subscribe selectively to specific groups in order to utilize bandwidth efficiently. While receiving one stream, receivers conduct *join experiments* that try to subscribe to a higher bitrate stream, switching to that stream if they are successful. Because congestion detection is challenged by interference between receivers' join experiments, a *shared learning* technique was introduced in which receivers collectively share knowledge about successes and failures of join experiments and synchronize future attempts. However, this approach increased convergence time as the number of receivers grew. In [24], receivers coordinate through the use of *synchronization points*, special packets in a sender's data stream. Neither approach significantly reduced complexity; together, they suggest the need for congestion management support at the network layer to enable successful multicast distribution.

2.1.3 Congestion Control

The RLM approach ultimately faced difficulties due to the lack of network-layer congestion support, instead trying to solve congestion problems on an end-to-end basis. This suggests that **network-supported congestion control** is a key affordance of the NDN architecture for RTSD applications. As described in [25], the NDN architecture aims to provide hop-by-hop flow control, which makes receiver-driven multicast approaches viable. Network-layer approaches also provide an opportunity to support fair use of resources by concurrent consumers, a long-standing design objective in the Internet architecture [26].

At the time the applications were developed, the open source implementation of NDN did not yet provide these mechanisms. In order to provide running software suitable for experimentation, the design presented here follows an end-to-end approach that will be revised as new network-layer mechanisms become available. Effort on congestion control is ongoing by the NDN team, and related work such as [27] proposes specific mechanisms for how routers can control the transmission quantity of the Interests according to the utilization of the link. In the network-supported case, a stream consumer can respond to congestion and other varying network conditions more effectively using feedback from the network such as congestion NACKs.

Before such support is available, the designs described in this paper perform receiver adaptation based solely on local estimates of available bandwidth. Related work includes [28], [29], which present consumer-driven congestion control mechanisms that control the number of Interests in the network based on AIMD [30]. However, these methods cannot maintain high throughput and low-latency transmission at the same time [31]. Therefore, another approach to estimating available network bandwidth on the consumer side is proposed in Sect. 3.5.1.

3. Designing RTSD over NDN

RTSD applications require protocols to **minimize latency** for retrieval of newly produced data, provide both **jitter tolerance** and **loss tolerance**, and **adapt to available bandwidth** to meet the applications' throughput objectives and avoid congestion. Furthermore, to leverage NDN as previously discussed, the design follows a **receiver-driven** approach with **storage-friendly** data naming.

To incorporate the affordances of NDN, the research shifts the perspective of application design from one of communication to one of information distribution, as suggested by the NDN architects. Ideally, the NDN network should be

considered a "black box" that makes a best effort to deliver Data packets matching the Interests expressed to it. Thus, the design of RTSD **producers** must determine:

1. what name should be given to each new data object;
2. how each object should be signed and, optionally, encrypted, and how corresponding keys should be made available;
3. how the data generation pattern should be modified based on current network conditions and/or client needs.

The design of RTSD **consumers** must determine:

4. the names of data that must be fetched;
5. what keys are needed to verify and decrypt the retrieved data;
6. what Interest expression pattern should be used to meet performance objectives under the current producer and network conditions.

This paper addresses Questions #1, #4, and #6, which enables development of and testing fully functional RTSD applications over the existing NDN implementation. Section 3.2 covers #1 and #4 by generalizing application namespace and providing insights into the namespace design process. Sections 3.3 and 3.4 address #6, discussing the asynchronous processes of Interest expression and Data buffering, to address how to achieve low-latency retrieval and loss tolerance. Section 3.5 discusses bandwidth adaptation.

Without producer adaptation (#3), the range of available network conditions is assumed or manually adapted based on testbed conditions. While per-packet verification is included in the driver applications, a security design addressing Questions #2 and #5 fully is left for future work, which is expected to incorporate the recently published approaches for schematized trust [16] and name-based access control [17].

In the remainder of this section, a specific design approach to address these requirements is described, and answers to these design questions are provided. The discussion assumes familiarity with the NDN architecture (readers are encouraged to review the references provided above). Note that the design is based on the *current* NDN architecture and its open source implementation [32], the implications of which are discussed further below.

3.1 Application Architecture

The design answers the questions above, using the general application architecture given in Fig. 2.

A producer generates data in the form of Application Data Units (*ADU generation*), marshals it into named Data packets (*Segmenter*), and stores them in an application-level cache (*Cache*), which responds to incoming Interests. The local Pending Interest Table (*PIT*) stores unanswered Interests until data is produced or they time out.

The consumer's *Interest Pipeline* issues Interests to the network for data as needed. The *Data Buffer* module receives

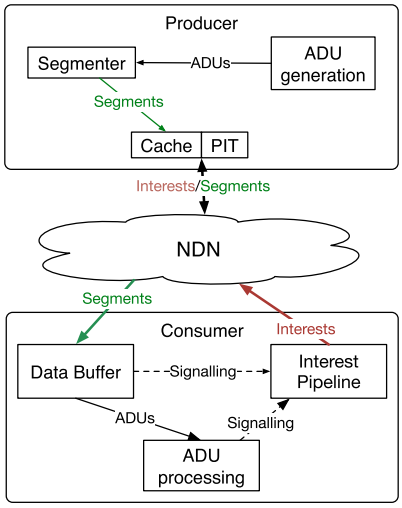


Fig. 2 NDN consumer and producer conceptual design.

Data packets and reassembles the ADUs, passing them to *ADU processing* for use by the application. To ensure RTSD objectives are met, the consumer monitors network performance by examining when Data packets return relative to the Interest that requested them, tracking timeouts and, in the future, handling congestion NACKs, in order to adjust to network conditions. It must also accommodate jitter in delivery timing via the buffer and packet losses via recovery techniques, such as Forward Error Correction (FEC) or retransmitted Interests.

In the design, Data processing and Interest expression are asynchronous, coordinated through signaling between the *Data Buffer* and *Interest Pipeline* components. Further signaling between *ADU processing* and the *Interest Pipeline* provides high-level control Interest expression (initiation, pause, restart, etc.).

3.2 Namespace Design for RTSD

NDN application protocol design starts with a namespace. Effective namespace designs describe the data in a way that makes sense to the application *and* considers how this data can be efficiently fetched[†]. The namespaces described in subsequent sections were iteratively developed through design, implementation and testing. Where possible, they aim to *describe the data* being made available by the producer, rather than *naming messages* as done in protocol design for host-to-host communication in IP. This avoids unnecessary session semantics and promotes storage friendliness, while not precluding name or data encryption for security. In NDN, only data that is requested traverses the network. Therefore, a superset of the data needed for any one access pattern can be published without penalty. This enables producers to name the data they have, and for consumer applications to employ the best access pattern for their needs.

[†]Further, names must take into account forwarding and security

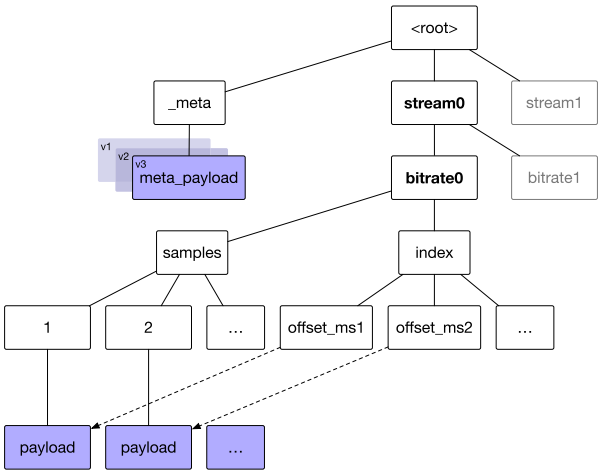


Fig. 3 Generalized RTSD namespace design.

A typical RTSD application namespace is illustrated in Fig. 3. Consumers start with well-known prefixes and then discover the specific names that they need in order to fetch data from the stream. This is done at several stages of retrieval. First, a consumer obtains a root prefix corresponding to a particular data source or context using an application-specific mechanism. For example, the producer prefix `/com/company/rtc/alice` could be supplied by an end-user, search engine, or local discovery mechanism[†]. Once the producer prefix is identified, a metadata object with a well-known name is fetched for that prefix, using Interest selectors to obtain the latest version. The corresponding object, e.g., `/<root>/_meta/<version>`, is an application-specific manifest describing the available streams and options available under that prefix. In a typical design, the producer may have several data streams (`stream0, stream1` in Fig. 3)–for example, for audio and video or for distinct sensors connected to an acquisition device. Each stream may have several quality levels, shown as bitrates for simplicity in Fig. 3. Both relations are represented as siblings in the name tree.

Application conventions, along with metadata from the manifest, are then used to construct the longer prefix where samples are published, such as `/com/company/rtc/alice/main_camera/high_quality/samples/`. The consumer then employs the techniques described in Sect. 3.3.2 to fetch the latest data available under current network conditions from that stream prefix.

The producer supports data access using multiple methods. For example, because a sequentially numbered sample namespace does not allow consumers of archived data to retrieve data easily for a certain time, a producer could provide an `index` namespace with child objects, corresponding to timestamps, that provide pointers into the segment namespace, as done in NDNVideo [34] and shown in Fig. 3. Furthermore, given the latency requirements, RTSD producers should name sample-level data to facilitate consumers quickly beginning to render for a given stream, which may have different implications for different data types. For example, the NDN-RTC real-time video conferencing application, described in Sect. 4, uses a namespace design that distinguishes between key and delta frames, to enable consumers to issue different numbers of simultaneous Interests depending on the frame type.

3.3 Interest Expression Control

To yield a continuous stream of data, once it knows the names to retrieve, an RTSD consumer must issue multiple simultaneous Interests. It tracks the number of *in-flight Interests*, those issued but not answered, with the **Interest pipeline** mechanism. Three important observations can be made. 1) As long as the Interest lifetime is sufficiently long, there is no penalty for Interests that arrive before data is produced. 2) Data is assumed to be generated at a known sampling rate, available to the consumer via metadata or per-packet timestamps, which enables the consumer to compare the delay in production to the delay in arrival without clock synchronization. 3) Pipeline size, the time period covered by in-flight Interests multiplied by segment size, corresponds to the bandwidth-delay product of the data stream being fetched.

The Interest pipeline is characterized by its size in Interests, λ_p, and the number of interests that the *Interest pipepline* must issue, λ_s to keep λ_p interests in flight. At initialization, $\lambda_s = \lambda_p$. As the consumer controls λ_p, it can adjust the number and expression pattern of in-flight Interests by such techniques as Interest *bursting* and *withholding*, as shown in Fig. 4.

3.3.1 Consumer-Producer Synchronization

By definition, RTSD applications must retrieve the latest data as soon as it is generated, so consumers should issue Interests early enough so that they arrive upon data's production or earlier[††]. The design contains a producer-side PIT that holds arrived Interests and is checked immediately when data is produced (see Fig. 2), which is discussed further in Sect. 4.2.

The notion of RTT for NDN networks is generalized by introducing **data retrieval delay** (DRD), the consumer-observed time between when an Interest is expressed and the corresponding Data packet is received. For example, in the case of two RTSD consumers fetching the latest data from one producer (see Fig. 5), only one consumer's Interest reached the producer; the other consumer's ($C2$) Interests are aggregated in the PIT of the router (R) and retrieve data from R's cache. Nevertheless, both consumers $C1$ and $C2$

concerns, though these issues are not addressed in this paper.

[†]NDN provides mechanisms that can be used for dynamic namespace discovery, such as the NDN sync primitive [33], which we employ in several of our implementations.

[††]Note that while intermediate hops may aggregate duplicate Interests from multiple consumers, at least one will reach the producer for each data segment.

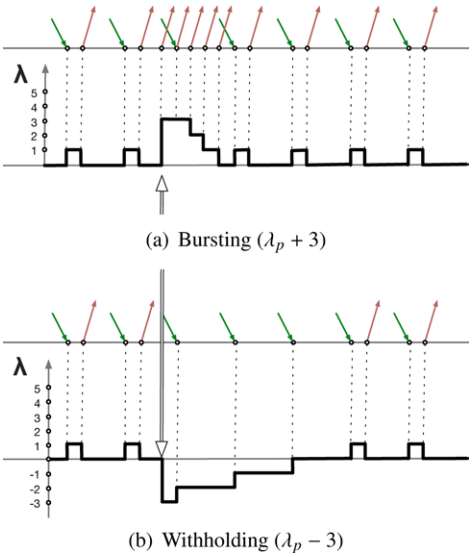


Fig. 4 Interest expression pattern adjustments.

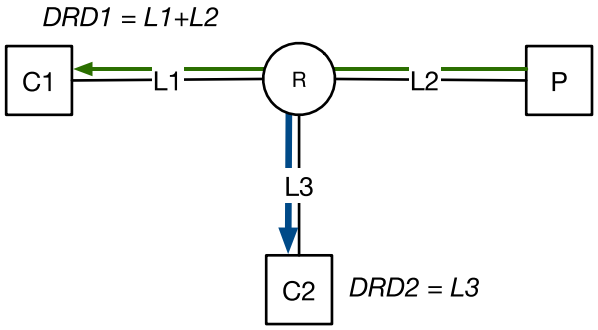


Fig. 5 Data Retrieval Delays in 1-to-2 RTSD fetching scenario: $C2$ experiences smaller DRD value when it starts fetching after the $C1$ and receives cached data from the R.

receive the latest data with minimum latency, even though the DRD values for them may differ.

Further, the delay between an Interest's arrival and the availability of corresponding data is defined as its **generation delay**, d_{gen}, which impacts the effective DRD measured by the consumer: $DRD' = DRD + d_{gen}$. Conceptually, d_{gen} should be kept small, to avoid arriving after the Data is produced, which would increase the playback latency for the consumer.

3.3.2 Retrieving the Latest Data

An RTSD consumer must achieve the goals of keeping DRD' minimal while fetching the latest data as soon as possible. Several designs are possible. For example, interests for uniquely named Data could be issued, requiring the producer to create new packets on the fly, but this limits scalability severely, especially if done for every packet. Or, Data can be named sequentially, and consumers can issue Interests with "rightmost child" and "fresh" selectors, using Data packet

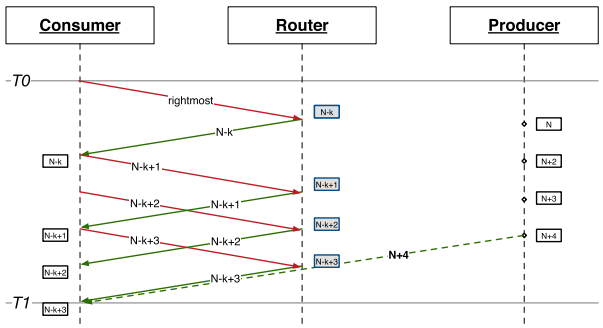


Fig. 6 Increased latency experienced at the consumer as a result of cache presence.

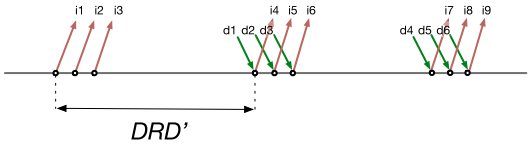
freshness values to expire old data[†]. However, in practice, this is insufficient. For example, the rightmost child selector provides the latest child as known by each hop, potentially returning data older than would be available if the Interest propagated further, as shown in Fig. 6. In high-rate RTSD publishers, many Interests and Data need to be in-flight simultaneously, and using selectors alone may not return each available segment reliably; further, selectors are not the architecturally preferred method for retrieving sequential data.

Thus, the consumer should stream real-time data from the producer by using *exact, sequential names*. The design combines knowledge of the producer sample rate with the use of selectors at bootstrapping to determine the latest sequence number, and then issues exact name requests for streaming. Observe that whether an in-path cache or producer responds, the consumer is not able to achieve playback latency lower than the fastest path between the consumer and the producer. In the scenario illustrated by the Fig. 5, Consumer $C2$ achieves playback latency no less than $L2 + L3$, even though effective DRD is shorter ($L3$).

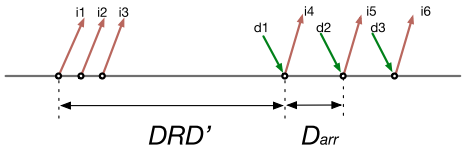
The approach is summarized as follows: 1) The producer names data sequentially, and provides the sampling rate–the data generation rate–in stream-level or segment-level metadata. 2) To bootstrap real-time retrieval, a consumer issues Interests with rightmost child selectors to get latest samples as known by the nearby router, and learn sequence numbers and sampling rate from the metadata of the received packets. After that, it expresses Interests with exact names at a rate faster than the sampling rate. If Data packets arrive at the same rate Interests are issued, the data in these packets must be from router caches. 3) When Data packets arrive at the same rate as the sampling rate, the consumer assumes it is receiving the data that is being generated in real-time. Thus, the consumer can determine data freshness by comparing the consecutive data packet arrival pattern with the Interest expression pattern used to retrieve the data and the producer's sampling rate (see Fig. 7).

Real-time data is detected by observing that *it arrives no faster than the producer-defined sampling rate*, while

[†]See [35] for more information on selectors and other details presented here.



(a) Bursty arrival of stale data reflects the Interest expression pattern and indicates that the data is not the latest.



(b) Periodic arrival of the latest data reflects publishing pattern and sampling rate.

Fig. 7 Getting the latest data: arrival patterns for the stale and the latest data.

stale data can be retrieved at a higher rate. To differentiate the real-time data from stale data, the consumer performs *stability estimation*. **Inter-arrival delay**, D_{arr}, is monitored for consecutive samples, and once it is stabilized around the *sample period*, T, the data flow is considered stable, and the consumer is retrieving real-time data. An example is provided in Sect. 4.1.3.

3.3.3 Latency Minimization

To implement the approach described in the previous section, the consumer tracks the producer's sample period, T (calculated using the sampling rate provided in the packet metadata) and the estimated DRD, DRD_{est}, defining network **Interest demand** as follows:

$$\lambda_d = \lceil \frac{DRD_{est}}{T} \rceil \tag{1}$$

where DRD_{est} is calculated using effective DRD (DRD') and generation delay, received in the packet metadata: $DRD_{est} = DRD' - d_{gen}$. Interest demand defines the minimal number of in-flight Interests necessary for the consumer to retrieve real-time data with steady flow. The consumer adjusts the Interest pipeline size, λ_p, to match with the current Interest demand. As can be seen from Eq. (1), Interest demand may change depending on network conditions (measured by DRD_{est}) or the producer data generation pattern (sample period T). Therefore, the consumer needs to track the current Interest demand for the network and adjust its Interest pipeline size, λ_p, accordingly. Having too many Interests in the pipeline will lead a majority of them to time out while waiting for data. Having too few will cause the consumer to fall behind in fetching real-time data.

The current design uses a two-phase process to determine the Interest pipeline size for the current network conditions. During the *chasing* phase, the consumer aims to fast forward to the latest data the network can offer by issuing bursts of Interests (see Fig. 4(a)) until the consumer detects the arrival of data at the target sampling rate. At this point, the consumer is fetching data with minimal delay, although the value of λ_p may be larger than necessary at this time. During the *adjusting* phase, the consumer reduces λ_p (Interest withholding), shortening the pipeline until stale data begins to arrive, at which point the previous value of λ_p is restored. More generally, the consumer attempts to set λ_p as follows:

$$\begin{aligned} \underset{\lambda_d}{\text{minimize}} \quad & DRD'(\lambda_p) \\ \text{subject to} \quad & E(D_{arr}(t)|I_{exp}(t)) = T \end{aligned} \tag{2}$$

where $D_{arr}(t)$ describes the pattern of data arrival at the consumer over time, and $I_{exp}(t)$ describes the Interest expression pattern. At the completion of bootstrapping, after the adjusting phase, the effective DRD' will be close to the actual network DRD. Thus, the consumer will receive the latest data with minimal latency by maintaining λ_p in-flight Interests.

3.4 Network Jitter Mitigation and Loss Recovery

The RTSD consumer must also provide a buffer for Data packets, which, as in IP streaming, aims to mitigate the impact of network latency variations and, potentially, packet loss on application performance. Sizing of the buffer must balance the target latency, acceptability of dropped samples, and volatility of network performance in an application-specific manner. The current buffer design supports two packet loss recovery mechanisms–Interest retransmission and Forward Error Correction (FEC)–and leaves the exact buffer sizing up to the individual application.

The buffer design is shown in Fig. 8. The total buffer size (in milliseconds), S, is the sum of the *playable size*, S_{pl}, and the *reserved size*, S_r. These contain the assembled, "ready-to-be-rendered" content and sufficient space to accommodate the Interest pipeline, $\lambda_p * T$, respectively.

The consumer issues Interests to keep S_{pl} as close as possible to the target playable size, $S_{tpl} = \alpha * jitter + \beta * DRD_{est}$. The first component accommodates the jitter [36] observed over a long observation window, by using $\alpha \approx 1$ as a conservative parameter. The second component is NDN-specific: it allows the consumer to avoid buffer underflows in case of sudden data path changes, by adjusting β. However, larger values of β increase overall playback latency. These values need to be tuned based on application-specific tradeoffs.

Packet loss detection is realized by setting a deadline for packet arrival after expressing an Interest. Once a consumer detects loss, it can take one of the two recovery strategies: packet recovery using FEC parity data or Interest retransmission. At the retransmission checkpoint (P_{RTX} in Fig. 8), the consumer checks multi-segment samples for completeness and attempts to recover incomplete samples using FEC data if it is available. Otherwise, Interest retransmission is performed.

The retransmission pointer is $P_{RTX} = \gamma * DRD_{est}$,

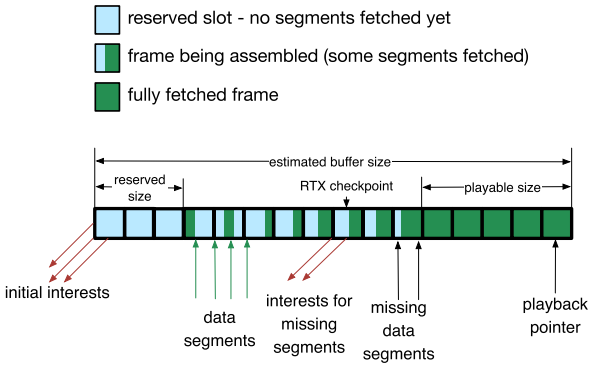


Fig. 8 Data buffering and retransmission design for NDN RTSD applications.

where $\gamma \in (0, min(1, \beta)]$ can be adjusted by the consumer depending on its understanding of current network conditions. The most conservative case, when $\gamma = 1$, covers worst-case scenarios, when the original Interest was lost on the way to the data source. Note that in future optimizations, late-arriving FEC data could be applied all the way up until playback, and retransmission could be triggered earlier.

3.5 Adaptation for Bandwidth Maximization and Congestion Control

For some RTSD applications, such as live video streaming and interactive videoconferencing, maximizing throughput during communication is important. At the same time, to maintain real-time communication, it is crucial to avoid growing packet queues in the routers and consequent packet losses. Therefore, the rate adaptation mechanism, which adapts to the actual available network bandwidth, is critical.

3.5.1 Congestion and Data Source Change Detection

An adaptive rate control mechanism for real-time video streaming in NDN [31] was proposed previously. As a method for the consumer to maintain real-time communication and best available throughput, the mechanism controls the sending rate of the Interests in order to control throughput of Data packets according to available network bandwidth. This means that the time interval between consequent Interest expressions is adaptively adjusted to the appropriate value, by estimating available bandwidth using two indicators. The first is the variation of DRD', and the second is the packet loss ratio. In order to avoid increasing the queuing delay in the router, the mechanism quickly decreases the Interest sending rate if indicators yield worsened conditions. During normal conditions, the mechanism increases the Interest rate to keep good throughput.

However, in multi-consumer scenarios, sudden changes in the data retrieval path may occur and cause changes of effective DRD. In order to distinguish between retrieval path changes and network congestion, the algorithm keeps track of the average DRD values for equal adjacent time intervals. By comparing these values, the consumer is able to make an educated decision about changed conditions. When the retrieval path changes, the difference of average DRD values in adjacent intervals will be conspicuously larger than in the case of network congestion.

4. Real-Time Streaming Data Use Cases

We developed, evaluated, and used three NDN RTSD applications implementing aspects of the design described in the previous section. The primary driver was NDN-RTC, a videoconferencing library designed to provide real-time, conversational multimedia communication over NDN. Also, we developed NDN-opt, an NDN data publisher for an open source, scalable multi-person tracking system and Ananke, a real-time control system for live multimedia performance.

4.1 NDN-RTC: Real-Time Videoconferencing

NDN-RTC is a real-time videoconferencing application using NDN [20]; here, we relate the specifics of that application design to the more general approach of the previous section.

4.1.1 Producer

The implementation of NDN-RTC producer is straightforward. It encodes acquired media data, slices it into smaller packets where necessary, FEC data, names the packet, and adds them to an application-level cache that handles Interests asynchronously.

Namespace design. The NDN-RTC namespace (see Fig. 9) follows the approach described in Sect. 3.2. A hierarchy of streams and different bitrates is named at the *media stream* and *media thread* levels. Different frame types, *Key* and *Delta*, are published as siblings on the *packet_type* level. Sequential frame numbering (level *frame*) uses NDN naming conventions for segment numbering [37] and enables consumers to establish a pipeline of Interests to use exact names to fetch data that is yet to be produced. FEC data is published at the *data_type* level.

4.1.2 Consumer

Conceptually, the consumer must: 1) choose the most appropriate stream from those provided by the producer; 2) fetch and reassemble incoming segments into frames, reordering if necessary; 3) mitigate the impact of jitter and packet loss; and 4) decode frames and play them out. The NDN-RTC consumer implements these features across three modules that directly follow the general design described in Fig. 2. The **Interest Pipeline** holds in-flight Interests for future data. Its properties are adjusted using feedback from the **Data Buffer** module, which reorders and assembles incoming Data segments into ADUs, e.g., frames, and initiates retransmissions when necessary. The variable λ_s, described in Sect. 3.3, is used for signaling between the two asynchronous components. Once ADUs are assembled, they are passed via a

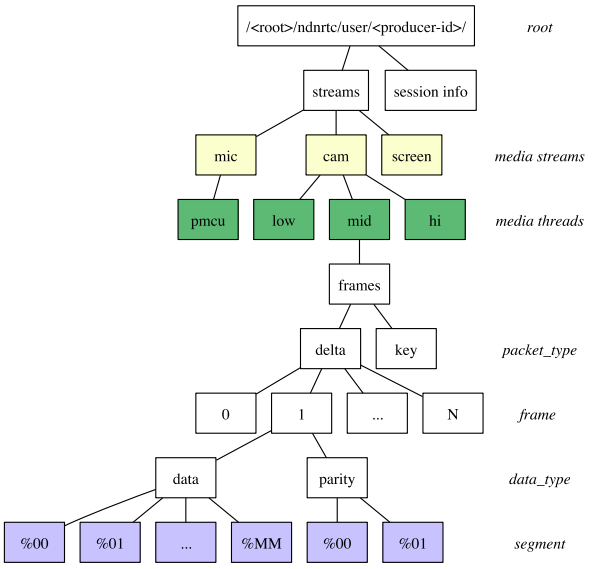


Fig. 9 NDN-RTC namespace.

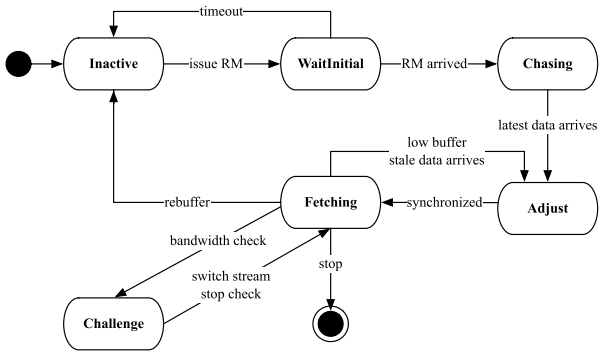


Fig. 10 NDN-RTC consumer state diagram.

queue to the **ADU Processing**, where in NDN-RTC handles video and audio decoding and rendering.

Consumer state machine. The NDN-RTC consumer implements the state machine shown in Fig. 10. Each media fetching thread starts in the *Inactive* state until the user selects a stream to fetch. Then, to initiate bootstrapping, an Interest with the "rightmost child" selector is issued as part of the *issue RM* transition on the state diagram. The consumer stays in the *WaitInitial* state until the data arrives, or it times out and is reissued. NDN-RTC extracts necessary metadata from the object and initiates the chasing phase represented by the *Chasing* state on the diagram. Arrival of the latest data triggers the transition to the *Adjust* state, described below, where frame rendering first begins to deliver data to the end user.

Once the adjustment phase has been completed and the Interest pipeline size, λ_p, corresponds to the current Interest demand, λ_d, the consumer switches to the *Fetching* state. During *Fetching*, the consumer monitors the buffer level and the stability estimator, transitioning to *Adjusting* mode

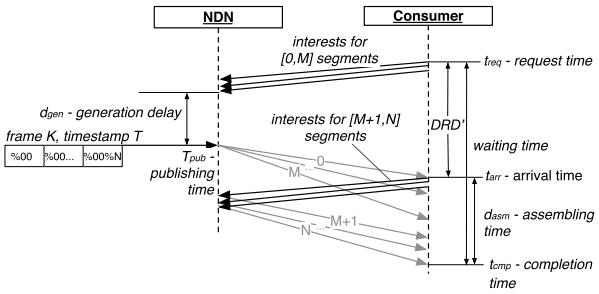


Fig. 11 Video frame fetching timeline in NDN-RTC.

in either of two cases: a) low buffer level or b) stale data begins to arrive. This approach helps the consumer to react to changing network conditions. Periodically, the consumer enters the *Challenge* state in order to start requesting data from higher bitrate streams for bandwidth estimation and rate control.

Frame fetching. A primary extension of NDN-RTC from the general design is to handle multi-segment objects. The consumer issues a bundle of Interests for each frame as illustrated in Fig. 11. The number of segments in a given frame is not known ahead of time. Thus, the consumer maintains estimators M (specifically, M_{key} and M_{delta}) for this value for the different frame types. The actual number of segments comprising a given frame, N, is delivered in each segment as metadata. If $N > M$, the consumer issues more Interests; if $N < M$, some Interests may go unanswered. (The latter is preferred over the additional round-trip penalty.)

4.1.3 Latency Minimization and Interest Expression Control

Interest pipeline. During bootstrapping, the consumer aims to catch up with the producer's data production and fetch the latest data. It initializes λ_p using current Interest demand as in Eq. (1), and transitions through the *Chasing* and *Adjust* states as described in the previous section. The following equations are used for adjusting the Interest pipeline:

- *Interest bursting:* $\lambda_p^i = 2 * \lambda_p^{i-1}$
- *Interest withholding:* $\lambda_p^i = \frac{3}{4}\lambda_p^{i-1}$

After each Interest pipeline adjustment, the consumer waits for some period of time, called the *detection period*, to detect the impact of the previous action by measuring the network's output (data arrival pattern and DRD'). No new Interest pipeline adjustments are made until the detection period is expired.

Latest data detection. The consumer monitors packet inter-arrival delay, D_{arr}, during the chasing phase in order to differentiate the latest data from stale data. As discussed in the previous section, we observe that 1) stale data arrives in bursts, following Interest bursting pattern, thus D_{arr} value fluctuates widely; 2) latest data arrival is more stable, D_{arr} fluctuations are minimal; 3) during latest data arrival, D_{arr} stabilizes around producer's sample period, T. To detect

latest data, the NDN-RTC consumer estimates D_{arr} attenuation during the chasing phase by comparing average values of D_{arr} over two consecutive non-overlapping windows (attenuation windows). Once D_{arr} has stabilized, its average value is compared to producer's sample period T in order to confirm that the data arrives at producer's sampling rate. The following stability estimator is used:

$$\hat{s} = \begin{cases} 1, & \text{if } (\frac{m_1}{m_2} \leq \theta_1) \text{ and } (1 - \frac{|m_1 - T|}{T} \leq \theta_2). \\ 0, & \text{otherwise.} \end{cases} \quad (3)$$

where:

$$m_1 = \frac{1}{N} \sum_{k=i-N}^{i} D^k_{arr} \qquad m_2 = \frac{1}{N} \sum_{k=i-2N}^{i-N} D^k_{arr}$$

m_1 is a running average of the last N values of D_{arr}, and m_2 is a running average of the N values of D_{arr} before that. The attenuation coefficient, θ_1, similarity, θ_2, and attenuation window, N, are pre-defined constants. The first condition of the stability estimator in Eq. (3) checks for D_{arr} attenuation, while the second condition checks that D_{arr} stabilized around the producer's sampling period, T. The latter is a simple way to check whether arriving data packets follow the producer's publishing pattern, thus representing the latest data; more sophisticated estimators could be used in the future. The state change from *Chasing* to Adjust is made after estimator reports latest data arrival for K consecutive times. The next section discusses actual values of θ_1, θ_2, N and how this estimator performs in practice.

4.1.4 Buffer Management and Loss Recovery

The NDN-RTC consumer sizes the buffer and handles loss recovery according to the ideas explained in Sect. 3.4 with $\alpha = 1$ and conservative parameters, $\beta = 2$ and $\gamma = 1$[†]. The consumer checks whether all segments for a frame have been received at the retransmission checkpoint. If not, it attempts to recover it using FEC data. If it fails, the consumer retransmits Interests for the missing segments. With $\beta = 2$ and $\gamma = 1$, the consumer is guaranteed to have at least DRD_{est} milliseconds to receive the missing segment.

4.1.5 Adaptive Rate Control

To achieve the rate control described in Sect. 3.5, the producer generates and delivers multiple bitrate video streams. Additionally, the consumer shifts to the *Challenge* state from the *Fetching* state, and estimates an available network bandwidth with a variation of DRD. Then, the consumer selects the appropriate bitrate of the video stream and fetches it. The consumer employs the following basic design:

- In the *Challenge* state, the consumer fetches two video streams of different bitrates at the same time. One is "current media," which the consumer plays back, and the other is "additional media," which is a higher bitrate than "current media."
- According to the algorithm proposed in [31], the consumer estimates available bandwidth while referencing DRD of two media steams, and controls the sending rate of Interest packets based on the bandwidth estimation. However, the consumer only controls the Interest packet for "additional media," because the expression timing of the Interest packet for "current media" should not be changed.
- If a bandwidth estimate is above the bitrate of "additional media," the consumer stops fetching "current media," and switches to fetching "additional media" only.
- Bandwidth estimation is an ongoing process, so the consumer is able to detect when available bandwidth is less than the bitrate of "current media." In this case, it instantly switches to fetch a lower bitrate media stream.

The current design assumes that different media streams ("current" and "additional" media) follow the same path in the network and originate from the same producer. To cover other cases, the algorithm should be upgraded to perform per-stream bandwidth estimation measurements.

4.2 OpenPTrack: Real-Time Positional Tracking

OpenPTrack is an open source positional tracking system that uses networks of 3D imagers (such as the Microsoft Kinect) to detect and track people in real-time [38]. It is designed for use in interactive applications in education, arts, and culture. To drive interactivity, it must provide real-time streaming positional data to one or more network clients. Currently, the system supports UDP, WebSockets, and NDN for dissemination. NDN's receiver-driven, connectionless approach is a good fit for this application, and its storage friendliness provides an easy path towards storage and retrieval of historical data in future applications. UCLA REMAP developed a straightforward RTSD publisher and consumer prototype for OpenPTrack, NDN-opt, in 2014-15, based on what had been learned from NDN-RTC (Fig. 12). It was used in lieu of other middleware to support six interactive undergraduate thesis projects exhibited at the end of 2015 (Fig. 13). These browser-based projects were written in HTML5/Javascript, and successfully used a Javascript-based NDN-opt consumer library to retrieve positional tracking data from a C++ publisher module compiled into OpenPTrack.

In this project, we employed RTSD ideas developed for NDN-RTC to achieve low-latency data delivery. In developing this RTSD application, the need in application-level cache and PIT was reinforced. Additionaly, an approach for dynamic namespaces was demonstrated.

4.2.1 Producer

As in NDN-RTC, the NDN-opt producer is straightforward.

[†]These values were established empirically to provide reasonable quality of experience in the face of loss, over many test runs on both the NDN testbed and in isolated tests.

Fig. 12 A three-imager OpenPTrack system being demonstrated live at UCLA's Royce Hall; real-time tracks are shown in the projection.

Fig. 13 Student interactive media project using NDN-opt to provide real-time person tracking data to a browser-based immersive experience via NDN.

Via the host framework for OpenPTrack, the Robot Operating System (ROS), the NDN-OPT publisher receives a callback when new person tracking data is generated. Tracking data is expressed as NDN objects with Javascript Object Notation (JSON) payloads, containing timestamps and location coordinates, named sequentially. Data is produced at approximately 60 Hz.

Figure 14 illustrates the data namespace. While streams and bitrates in an NDN-RTC do not change during operation, the NDN-opt namespace is dynamic and may add or remove uniquely identified track components during operation, as people enter and leave the tracked space. The *track hint* child in the namespace provides the consumer with a list (manifest) of current tracks under a well-known name. Tracking hint objects provide the IDs of currently active tracks, and are published periodically with the producer's current timestamp. Track IDs extracted from *hint_payload* can be used to express Interests for actual tracking data under the *tracks* namespace.

Note that just as NDN-RTC, which sometimes must request additional segments for unusually large frames, consumption is also a two-step process here. Track hints provide a list of streams dynamically for the consumer to retrieve.

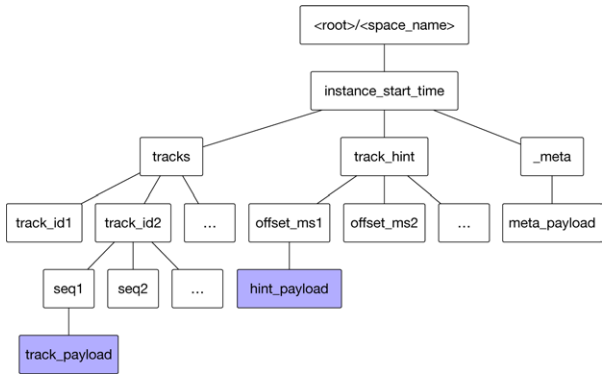


Fig. 14 NDN-opt namespace.

4.2.2 Consumer

The consumer aims to acquire tracking data with minimal delay once it learns the track ID from the hint packets. Hint data fetching is accomplished by having the consumer express Interest with everything older than or equal to the timestamp of the last hint excluded and the right-most-child selector.

As described in the previous section, the consumer aims to deliver Interests before the corresponding Data is produced. This application-level PIT holds interests received at the application level, since there is no mechanism in the current NDN implementation for the forwarder to notify the application that newly produced data had been previously solicited. The authors' first implementation did not have an application-level PIT, and data bursts were often observed on the consumer side. Fewer bursts were observed after introducing an application-level PIT, which reduced the percentage of data arrival intervals that were larger than 100 ,ms. Since the application-level PIT makes application-level caches more usable, it is now implemented in the NDN Common Client Libraries [39]. In the future, the forwarder implementation may provide a similar mechanism and application API directly.

As in the NDN-RTC design, an Interest pipeline that maintains a series of Interests for the next few Data packets is maintained. The pipeline issues an Interest for the next sequence number when data is received and the sequence number is within the current pipeline. NDN-opt uses a loss recovery approach based on retransmission only, as the preliminary version could tolerate larger latencies (100–200 milliseconds).

4.3 Ananke: Distributed Control System for Live Performance

Another example of RTSD is a distributed media-cue control system for an interactive performance, *Los Atlantis*, which was rehearsed and performed at the UCLA School of Theater, Film and Television from April-June 2015 (Fig. 15).

We used this opportunity to demonstrate the viability

Fig. 15 Performance environment controlled by Ananke via NDN, with TouchDesigner in the foreground.

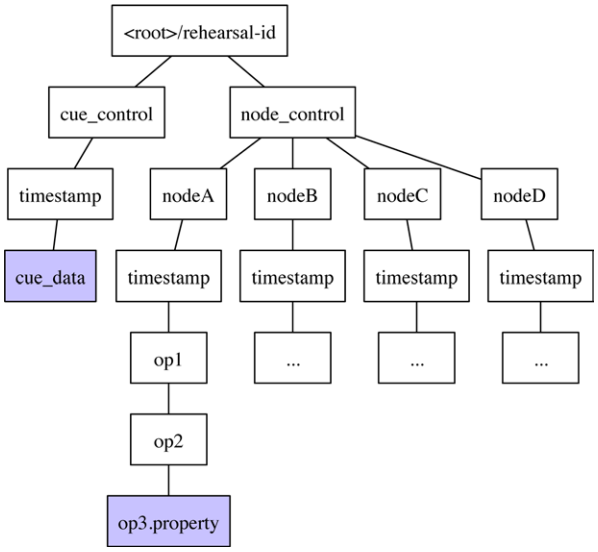


Fig. 16 Ananke namespace.

of RTSD techniques to emulate push-style control.

The specifics of the performance required multiple independent nodes, represented by a rolling tower with a laptop and a projector, which could receive cues to initiate or suppress video projection processing. A media cue was represented as a Data packet, and each node ran an NDN consumer application, which maintained outstanding Interests for the next cue. Namespace designed during this project, allowed for easy and flexible one-to-many control and customization of individual nodes.

During each individual performance of *Los Atlantis*, the actors guided the audience to experience unique, evolving sets of video and audio content, based on choices made by the specific audience (both in the performance space and experiencing the performance online). Unlike a more traditional performance, wherein cues can be called as a reaction to choreographed movements and scripted spoken words, the dynamic nature of *Los Atlantis* was well-suited for NDN.

NDN's name-based forwarding enabled a straightforward mapping between network communication and the projection control software (TouchDesigner), which represents data flow processing internally as a hierarchically structured set of named operators. It was trivial to expose this namespace to network manipulation via NDN, which allowed for easy, customizable and scalable control over the group of nodes. This was crucial for the performance, which continued to change during the rehearsal process and run, and relied on continuously shifting relationships with media content stored online.

As can be seen from Fig. 16, a *cue_control* sub-namespace was used to publish cues. A timestamp name component allowed consumers to exclude older cues and fetch only the latest data. A parallel sub-namespace provided access points to the projection software tuning. Every component (i.e. *nodeA*, *nodeB*, etc.) can have a hierarchical name identical to TouchDesigner's operator name, with a last component added to denote the property of the operator. The payload carries actual value for the property. By injecting a *timestamp* component in the hierarchy, consumers at different nodes were able to fetch only the latest changes posted by the producer. With the support of data storage, this enabled a powerful "re-play" mechanism for the entire process.

5. Evaluation

This section describes initial evaluation results of the design concepts described above, as implemented in the NDN-RTC library and tested with experimental runs of the *ndncon* tool [40] (Fig. 17), a desktop application built on top of NDN-RTC [20]. Unless noted, all tests use the NDN Platform version 0.4 on MacOS X (Intel Core i7, 16 GB RAM), with UDP tunnels.

5.1 Consumer/Producer

Two tests demonstrated basic consumer and producer scalability. During the first, two consumers and a producer were configured in the topology illustrated by Fig. 18(a). The producer was started, and then each consumer was launched in succession. As can be seen from the results (see Fig. 19(a)), even though the bandwidth through Face 2 increased when consumer $C2$ started to fetch the stream at approximately 18 seconds, Face 1 throughput stayed unchanged. During the second test, hot failover was demonstrated between two routers[†]. The topology shown in Fig. 18(b) was used, and measurements were made of throughput over each node's faces. The consumer was configured to forward Interests to both routers. Approximately 100 seconds after the producer and consumer were both started, router $R1$ was shut down. As seen in Fig. 19(b), Data packets continued to flow from router $R2$; the transition did not cause any interruption in

[†]Synchronization of multiple producers is not yet available in the implementation.

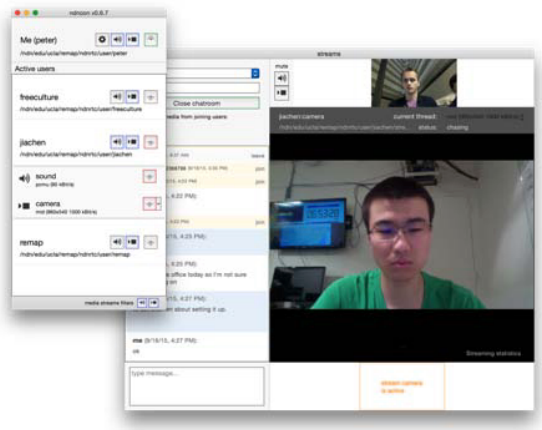
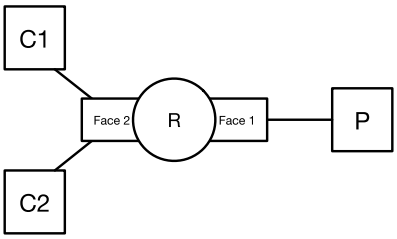
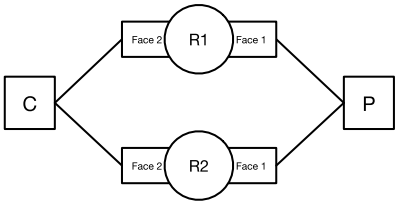


Fig. 17 Screenshot of *ndncon* - NDN-RTC demonstration application; real-time remote video in the foreground.



(a) Consumer scalability test topology.



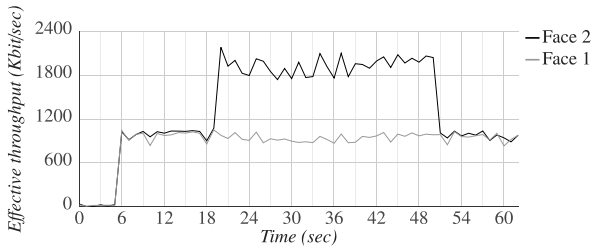
(b) Failover test topology.

Fig. 18 Network topologies for evaluating consumer scalability and failover.

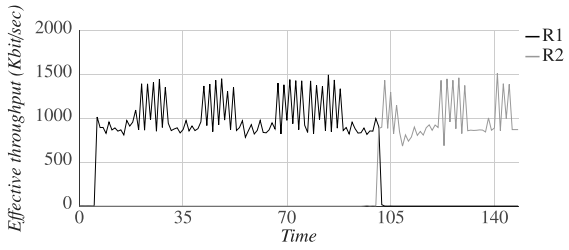
playback or packet loss at the consumer.

5.2 RTSD Performance Evaluation

Additional tests were conducted to evaluate NDN-RTC's latency minimization, loss recovery, and bandwidth utilization under simple conditions. For these tests, the topology, shown in Fig. 20, was used: two clients (consumer-producer), $CP1, CP2$, connected to a router, R. Except where noted, each consumer-producer pair established bidirectional audio-video communication at a 1000 Kbps video bitrate and a 100 Kbps audio bitrate, with video T=1/30 sec and audio T=1/50 sec. The network shaping tool, **tc** was



(a) Throughput at Face 1 and Face 2 for consumer scalability topology.



(b) Throughput at R1 and R2 for hot failover topology.

Fig. 19 Results of test runs on topologies in Fig. 18.

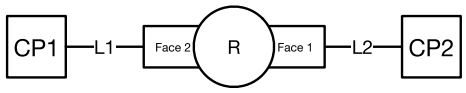


Fig. 20 Basic evaluation topology.

used to vary latency, loss ratio, and bandwidth on both links.

5.2.1 Latency Minimization

Interest pipeline adjustment. Six tests, with different starting pipeline sizes, were conducted to learn more about the bootstrapping process. The duration of the chasing period was measured along with other parameters. Table 1 summarizes these tests. As described in Sect. 3.3.3, in the adjusting phase, the consumer decreases λ_p until it starts acquiring stale data, at which point it restores the last value of λ_p and switches to the *Fetching* phase. This "Backoff period" of receiving stale data during the adjusting phase was measured for each condition and shown in the table. During bootstrapping, the consumer adjusts its pipeline size, λ_p, to match the current Interest demand value (for the tested network conditions of round-trip latency 100 ms and producer rate 30 FPS, Interest demand corresponds to 4). Thus, the initial value, λ_p^0, directly affects how fast the consumer is able to switch to the *Fetching* phase, sets its pipeline size, λ_p^{final}, and starts rendering media for the user. As λ_p is adjusted, the effective DRD' measured by the consumer changes as well. By the end of the bootstrapping, the resulting DRD' should be close to the actual network DRD value. Figure 21 illustrates the sample bootstrapping process for Test Run #1; in it, first two white areas represent chasing and back-off

Table 1 Bootstrapping test results.

Test run	Varied parameter	Results			
	λ_p^0	Resulting DRD'	Chasing period (ms)	Backoff period (ms)	λ_p^{final}
1	30	110	700	1600	4
2	15	110	700	1500	4
3	5	140	1100	0	5
4	4	110	2000	400	4
5	3	110	2500	2000	4
6	2	110	3800	0	4

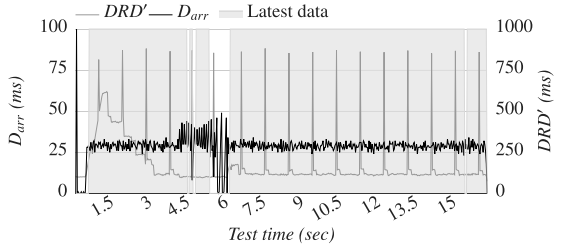


Fig. 21 D_{arr} and DRD' during consumer bootstrapping process; the grey area shows the detection of the latest data arrival.

periods respectively.

As can be seen from the results, larger initial sizes of the Interest pipeline yielded more aggressive consumer behavior and, thus, shorter chasing periods. Initializing the pipeline with less than the actual Interest demand required more time for the consumer to bootstrap. The best result, with shortest chasing and back-off durations, as well as best DRD' accuracy, was achieved when λ_p^0 was initialized with the current Interest demand estimation, as in Eq. (1).

Latest data detection. The chasing phase must be completed before data can be rendered for the user; thus, its duration directly affects QoE [41] and should be kept as small as possible. A series of experiments was conducted in order to explore how parameters of the stability estimator introduced in Sect. 4.1.3 impact the duration of the chasing period. Three groups of parameters for the implemented estimator were defined (given in Eq. (3)), representing different thresholds of what is considered "stable" enough to terminate chasing:

- *Low:* $\theta_1 = 0.6, \theta_2 = 0.5, N = 3, K = 4$
- *Medium:* $\theta_1 = 0.3, \theta_2 = 0.7, N = 10, K = 4$
- *High:* $\theta_1 = 0.1, \theta_2 = 0.95, N = 30, K = 4$

In order to simulate different network conditions, latency was varied on links $L1$ and $L2$, and the chasing period was measured for one of the consumers. Figure 22 shows how the estimator with the highest precision required longer chasing periods–i.e., more time to detect the latest data arrival–whereas lower precision provided faster chasing, and thus shorter playback startup times. However, the low precision estimator was observed to yield false latest data detection in more complex (multi-hop) topologies. Overall, the chasing period duration increases as link latency increases,

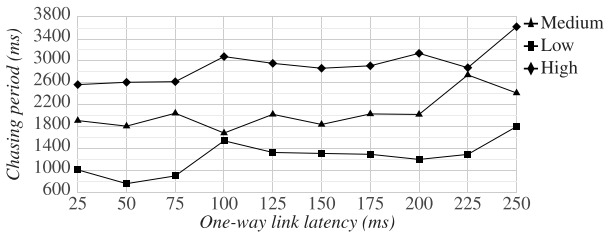


Fig. 22 Chasing period duration measured for different stability estimator parameters, for various link latencies.

at least according to these tests.

5.2.2 Loss Recovery

The authors compared packet loss recovery for 1000 and 8000 byte data payloads, corresponding to the approximate amount of application data that can be expected to fit in the MTU of the underlying transport (1500 bytes for UDP on the Internet) and in the maximum NDN segment size (8800 bytes). All of the previous tests used a maximum payload size of 1000 bytes, with the intention of enabling each packet to fit in a single MTU. The benefit from larger segment sizes should be to decrease the number of Interests and Data packets and name overhead. The maximum segment size directly impacted the consumer's ability to recover packets. As can be seen from Fig. 23, a consumer with a larger segment size suffered more under lossy link conditions, and was unable to start playback at all for 5% and 10% loss ratios. Figure 24 shows the performance of FEC and retransmission for packet loss recovery under different conditions for the two segment sizes. This revealed an FEC implementation issue. For an 8000 byte payload and 1000 Kbit/s video stream, most frames fit into a single segment. The FEC algorithm as implemented provided a parity segment for that frame that was unsuitable for recovery[†]. Thus, the consumer resorted to Interest retransmissions. For such situations, two corrections may be made in the future: 1) to handle one-segment media samples as special cases, and publish a copy of media samples under the parity namespace, or 2) to use a more advanced FEC approach across frames.

The poor performance independently of FEC in the 8000 byte payload configuration is likely caused by the following: For segments larger than MTU of the underlying transport, the NFD forwarder performs fragmentation and assembly, and uses retransmissions for lost packets. However, NFD's default retransmission timers are large relative to latency in this application and likely not suitable for RTSD–causing the losses observed at the consumer, as complete segments are rarely delivered on-time. In the future, this could be addressed by using small segments or providing required retransmission behavior at a lower layer of the stack.

[†]Currently, NDN-RTC uses a basic implementation of Reed-Solomon Forward Error Correction applied per frame.

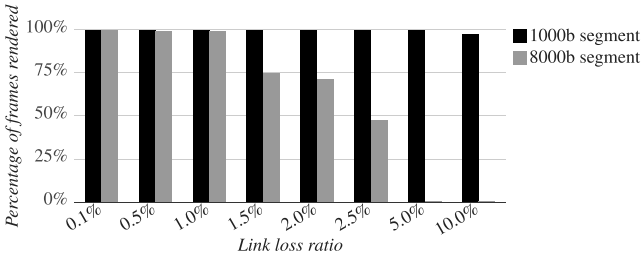
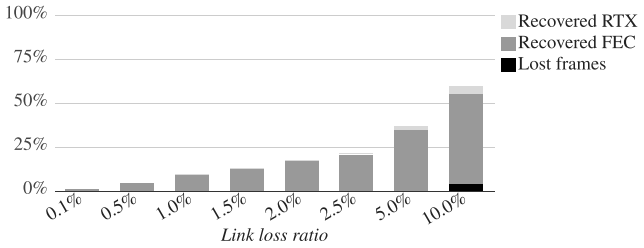
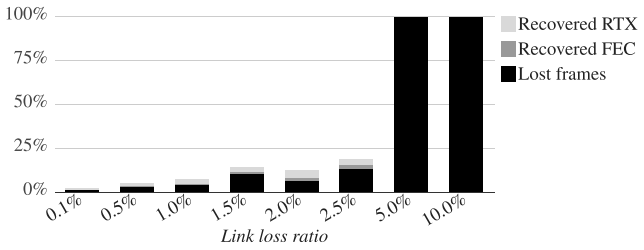


Fig. 23 Percentage of rendered frames to requested frames depending on loss link ratio per 1000 and 8000 bytes of payload.



(a) Maximum payload set to 1000 bytes.



(b) Maximum payload set to 8000 bytes.

Fig. 24 Percentage of lost frames, frames recovered using FEC and retransmission schemes to the total number of rendered frames.

5.2.3 Bandwidth Utilization

In order to show the basic feasibility of a rate adaptation mechanism in NDN-RTC, average throughputs were measured on the consumer side, and the network bandwidth efficiency of a bottleneck link was evaluated. In this evaluation, a bottleneck link was simulated using a network emulator with a limited throughput of 4000 Kbit/s on the link $L2$ (see Fig. 20). Tests were unidirectional with $CP1$ as a producer and $CP2$ as a consumer. $CP1$ produced three bitrate video streams (200 Kbit/s, 1000 Kbit/s and 3000 Kbit/s), with a segment size 1000 bytes, and 20% of FEC data. $CP2$ bootstrapped with fetching the lowest bitrate stream.

Figure 25 shows the result of the average throughputs for $CP2$. In this figure, $CP2$ shifted to the *Challenge* state at Second 2, and started to estimate an available network bandwidth. From Second 2 to Second 8, $CP2$ fetched both the lowest bitrate media stream as "current media" and the middle bitrate media stream as "additional media" at the same time. In this network condition, available bandwidth

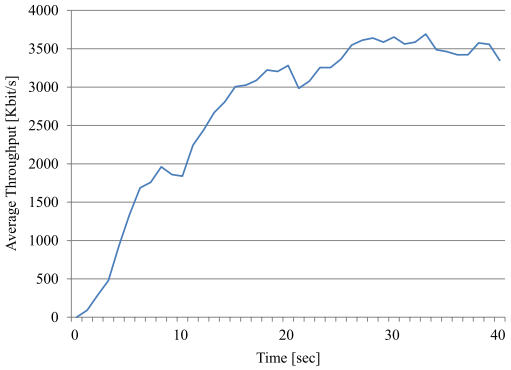


Fig. 25 Average throughput (each link delay is 0 ms, bandwidth of $L2$ is 4000 Kbit/s).

Table 2 Bandwidth utilization depending on network latency.

Round-trip link delay (ms)	Bandwidth utilization (%)
0	81.3
50	73.0
100	68.4
150	63.8

was enough for fetching the middle bitrate media stream, and $CP2$ started fetching middle bitrate media stream around Second 8. After that, $CP2$ started to probe bandwidth for a higher bitrate stream. Eventually, $CP2$ kept throughput around 3500 Kbit/s and continued to fetch the middle bitrate stream.

Next, bandwidth utilization was evaluated depending on link delay, which was varied from 0 ms to 150 ms. Table 2 summarizes these results. As can be seen, bandwidth efficiency was affected by increasing network latency. This is because the ARC algorithm reaction grows proportionally to the network delay, and errors in DRD estimation vary greatly. Thus, the consumer becomes more conservative and underutilizes the link.

6. Conclusions and Future Work

This paper describes an initial design for receiver-driven, real-time streaming data dissemination over NDN. It is based on experimental development and testing of running code for real-time video conferencing, data dissemination from a positional tracking system, and a distributed control system for live performance, each of which has been deployed in real-world experiments. It also describes specific evaluations of functionality and performance. The design includes initial approaches to namespace definition, minimizing latency, managing buffer size and lost recovery, and adaptive retrieval to maximize bandwidth and control congestion.

The design efforts explored dimensions that were expected to be important in NDN RTSD applications, including 1) the ability to operate without direct producer-consumer coordination, through loosely coupled, asynchronous Interest expression and Data buffering; 2) packet loss mitigation

by both forward error correction and retransmission; and 3) congestion control and rate adaptation based on end-to-end measurement, which should be improved significantly when NDN's hop-by-hop congestion control implementation become available.

Significant future work remains, which can be summarized by returning to the design questions presented earlier.

1. *How should the producer name each new data object?* Further work on namespace design is needed to explore how to express other types of data, such as geographically or spatially organized information and layered coding of media.
2. *How should the producer sign and, optionally, encrypt each data object, and make the corresponding keys available?* An important next phase is to use a schematized trust model for per-packet signing [16] and extend NDN-RTC to include encryption support by using name-based access control [17]. Applying these newly developed security solutions to specific applications has been proven the best way to gain deeper an understanding on their usability and naming requirements.
3. *How can the producer modify its data generation pattern based on current network conditions and/or consumer needs?* While a pure receiver-driven approach to RTSD has been pursued so far, producers in some applications may have the ability to adjust their data generation based on feedback. One question is what is the best way to gather consumer feedback, whether one can derive adequate feedback indirectly from observing Interests arrival patterns, or more explicit negotiation strategies may be needed.
4. *What is the best way for the consumer to learn the names of data that it wants?* Each stage of namespace discovery provides open challenges, including development of the best mechanisms to discover streams available within a certain application context and to request the latest data available from the network in low-latency, high-rate data streams. What application and network-layer features are necessary to prioritize certain named data objects (such as a base layer in scalable video coding) over others also remains an open question.
5. *What keys must the consumer fetch to verify and decrypt the retrieved data?* Recent work in schematized trust and name-based access control can provide standard mechanisms for each consumer to request the keys that it needs based on the data it wishes to verify and/or decrypt. However, specific security designs for RTSD applications remain to be developed.
6. *What Interest expression pattern should be used to meet the consumer's performance objectives under the current network conditions?* Given the receiver-driven nature of the design, many areas of future work fall under this question, including error correction versus retransmission tradeoffs in practical applications and stream prioritization. Notably, given that the NDN architecture is still under development, there is a unique opportunity to coordinate application-level approaches to both congestion control and latency minimization with planned and potential future architectural capabilities, such as hop-by-hop flow control.

This work aimed to demonstrate that NDN provides fundamental advantages over IP for RTSD applications. In the design, no specialized infrastructure is required to scale the number of simultaneous producers or consumers for a given namespace, and NDN's intrinsic multicast support provides a significant gain in bandwidth efficiency. Furthermore, as demonstrated by the rate adaptation approach, random access to different versions of the same stream is achieved simply by changing the namespace being accessed. Finally, supporting historical data access to RTSD streams is trivial, by placing data objects in persistent storage as they are created, and using an index namespace to enable fetching to start from any arbitrary point in time. While there remains much to be done, NDN offers great promise for supporting future real-time applications.

Acknowledgements

This work is supported by the U.S. National Science Foundation (award CNS-1345318 and others). Initial work on NDN-RTC was also supported by the Cisco University Research Program. The authors thank Jiachen Wang for his assistance in testing NDN-RTC and Dave Oran for continued feedback on this research. Daisuke Ando implemented the FEC algorithm for NDN-RTC.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A survey of information-centric networking," IEEE Commun. Mag., vol.50, no.7, pp.26–36, 2012.
[2] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named data networking," ACM SIGCOMM Comp. Commun. Rev., vol.44, no.3, pp.66–73, 2014.
[3] V. Jacobson, D.K. Smetters, J.D. Thornton, M.F. Plass, N.H. Briggs, and R.L. Braynard, "Networking named content," Proc. 5th International Conference on Emerging Networking Experiments and Technologies, CoNEXT'09, pp.1–12, 2009.
[4] D. Kulinski, J. Burke, and L. Zhang, "Video streaming over named data networking," IEEE COMSOC MMTC E-letter, vol.89, no.4, 2013.
[5] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, "Securing instrumented environments over content-centric networking: The case of lighting control and NDN," 2013 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp.394–398, 2013.
[6] J. Burke, P. Gasti, N. Nathan, and G. Tsudik, "Secure sensing over named data networking," 2014 IEEE 13th International Symposium on Network Computing and Applications, pp.175–180, 2014.
[7] C. Fan, S. Shannigrahi, S. DiBenedetto, C. Olschanowsky, C. Papadopoulos, and H. Newman, "Managing scientific data with named data networking," Proc. Fifth International Workshop on Network-Aware Data Management, NDM'15, pp.1–7, 2015.
[8] A. Afanasyev, Z. Zhu, Y. Yu, L. Wang, and L. Zhang, "The story of chronoshare, or how ndn brought distributed secure file sharing back," NDN, Technical Report, NDN-0029, 2015.

[9] W. Shang, Z. Wen, Q. Ding, A. Afanasyev, and L. Zhang, "Ndnfs: An ndn-friendly file system," NDN, Technical Report, NDN-0027, 2014.
[10] K.C. Claffy, J. Polterock, A. Afanasyev, J. Burke, and L. Zhang, "The first named data networking community meeting," ACM SIGCOMM Computer Communication Review, vol.45, no.2, pp.32–37, 2015.
[11] W. Shang, Q. Ding, A. Marianantoni, J. Burke, and L. Zhang, "Securing building management systems using named data networking," IEEE Netw., vol.28, no.3, pp.50–56, 2014.
[12] M. Amadeo, C. Campolo, A. Iera, and A. Molinaro, "Named data networking for IoT: An architectural perspective," 2014 European Conference on Networks and Communications (EuCNC), pp.1–5, 2014.
[13] G. Grassi, D. Pesavento, G. Pau, R. Vuyyuru, R. Wakikawa, and L. Zhang, "VANET via named data networking," 2014 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS), pp.410–415, 2014.
[14] P. Crowley and J. DeHart, "Named data networking presentation and demonstration," China-America Frontiers of Engineering Symposium, National Academy of Engineering, China, May 2013.
[15] Z. Wang, Z. Qu, and J. Burke, "Project matryoshka: Ndn multiplayer online game (poster)," NDN Community Meeting, Los Angeles, CA, Sept. 2014.
[16] Y. Yu, A. Afanasyev, D. Clark, K. Claffy, V. Jacobson, and L. Zhang, "Schematizing trust in named data networking," Proc. 2nd International Conference on Information-Centric Networking, ICN'15, pp.177–186, 2015.
[17] Y. Yu, A. Afanasyev, and L. Zhang, "Name-based access control," Tech. Rep. NDN-0034, NDN Project, Oct. 2015.
[18] 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," Proc. 2009 Workshop on Re-Architecting the Internet, ReArch'09, pp.1–6, 2009.
[19] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang, "ACT: audio conference tool over named data networking," Proc. ACM SIGCOMM Workshop on Information-Centric Networking, ICN'11, pp.68–73, 2011.
[20] P. Gusev and J. Burke, "NDN-RTC: Real-time videoconferencing over named data networking," Proc. 2nd International Conference on Information-Centric Networking, ICN'15, pp.117–126, 2015.
[21] T.-Y. Huang, R. Johari, N. McKeown, M. Trunnell, and M. Watson, "A buffer-based approach to rate adaptation," Proc. 2014 ACM Conference on SIGCOMM, SIGCOMM'14, pp.187–198, 2014.
[22] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," ACM SIGCOMM Comput. Commun. Rev., vol.26, no.4, pp.117–130, 1996.
[23] B. Li and J. Liu, "Multirate video multicast over the Internet: An overview," IEEE Netw., vol.17, no.1, pp.24–29, 2003.
[24] L. Vicisano, J. Crowcroft, and L. Rizzo, "TCP-like congestion control for layered multicast data transfer," Proc. IEEE INFOCOM'98, the Conference on Computer Communications, Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, Gateway to the 21st Century, pp.996–1003, 1998.
[25] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," Comput. Commun., vol.36, no.7, pp.779–791, 2013.
[26] V. Jacobson, "Congestion avoidance and control," ACM SIGCOMM Comput. Commun. Rev., vol.18, no.4, pp.314–329, Aug. 1988.
[27] G. Carofiglio, M. Gallo, and L. Muscariello, "Joint hop-by-hop and receiver-driven interest control protocol for content-centric networks," Proc. Second Edition of the ICN Workshop on Information-Centric Networking, ICN'12, pp.37–42, 2012.
[28] G. Carofiglio, M. Gallo, and L. Muscariello, "ICP: Design and evaluation of an interest control protocol for content-centric networking," 2012 Proc. IEEE INFOCOM Workshops, pp.304–309, 2012.
[29] S. Salsano, A. Detti, M. Cancellieri, M. Pomposini, and N. Blefari-Melazzi, "Transport-layer issues in information centric networks," Proc. Second Edition of the ICN Workshop on Information-Centric Networking, ICN'12, pp.19–24, 2012.
[30] D.-M. Chiu and R. Jain, "Analysis of the increase and decrease algorithms for congestion avoidance in computer networks," Comput. Netw. ISDN Syst., vol.17, no.1, pp.1–14, June 1989.
[31] T. Yoneda, R. Ohnishi, E. Muramoto, and J. Burke, "Consumer-driven adaptive rate control for real-time video streaming in named data networking," Internet Conference 2015, IC2015, pp.23–32, Oct. 2015.
[32] "NFD: Named data networking platform," http://named-data.net/codebase/platform/, Jan. 2016.
[33] Z. Zhu and A. Afanasyev, "Let's ChronoSync: Decentralized dataset state synchronization in named data networking," 2013 21st IEEE International Conference on Network Protocols (ICNP), pp.1–10, 2013.
[34] D. Kulinski and J. Burke, "NDNVideo: Random-access live and pre-recorded streaming using ndn," Tech. Rep., UCLA, Sept. 2012.
[35] "NDN packet format specification 0.2-alpha-3 documentation," http://named-data.net/doc/ndn-tlv/index.html, Jan. 2016.
[36] "ETSI TR 101 329-7: Telecommunications and Internet protocol harmonization over networks (TIPHON) release 3; End-to-end quality of Service in TIPHON systems; Part 7: Design guide for elements of a TIPHON connection from an end-to-end speech transmission performance point of view," Tech. Rep., European Telecommunications Standards Institute, 2002.
[37] Y. Yu, A. Afanasyev, Z. Zhu, and L. Zhang, "Ndn technical memo: Naming conventions," Tech. Rep., UCLA, July 2014.
[38] M. Munaro, A. Horn, R. Illum, J. Burke, and R.B. Rusu, "OpenPTrack: People tracking for heterogeneous networks of color-depth cameras," In IAS-13 Workshop Proceedings: 1st Intl. Workshop on 3D Robot Perception with Point Cloud Library, Padova, Italy, pp.235–247, July 2014.
[39] J. Thompson and J. Burke, "Ndn common client libraries," NDN, Technical Report, NDN-0024, Revision 1, Sept. 2014.
[40] "NdnCon GiHub Repository," https://github.com/remap/ndncon, Sept. 2014.
[41] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang, "Understanding the impact of video quality on user engagement," Commun. ACM, vol.56, no.3, pp.91–99, 2013.

Peter Gusev joined the UCLA Center for Research in Engineering, Media and Performance (REMAP) in July 2013 as a staff researcher. He focuses primarily on Named Data Networking (NDN), and is leading the development of NDN-RTC, a real-time conferencing library, and *ndncon*, a real-time conferencing application. In addition to NDN, Peter has contributed to a wide range of interactive audiovisual projects. His experience includes mobile development, video streaming, embedded programming, computer vision, web design and interactive augmented and virtual reality projects. Peter holds masters' degrees from Bauman Moscow State Technical University (computer science) and Wroclaw University of Technology (business information systems). Center for Research in Engineering, Media and Performance, University of California, Los Angeles.

Zhehao Wang is a graduate student in Computer Science at UCLA and researcher at REMAP. He focuses on Named Data Networking as an application developer, but has also contributed to a number of other cross-disciplinary projects. His research interests include computer networking, computer systems and architecture, and algorithms. Zhehao Wang received his undergraduate degree in 2014. Center for Research in Engineering, Media and Performance, University of California, Los Angeles.

Jeff Burke is Assistant Dean for Technology and Innovation at the UCLA School of Theater, Film and Television (UCLA TFT). He holds an M.F.A. in Film, Television and Digital Media and an M.S. in Electrical Engineering from UCLA. He has produced, managed, programmed and designed experimental performances, short films, new genre art installations and new facility construction internationally for more than 15 years, incorporating research that explores the intersections of the built environment, computer networks, and storytelling. In 2004, Burke co-founded UCLA TFT's Center for Research in Engineering, Media and Performance (REMAP), a collaboration with the Henry Samueli School of Engineering and Applied Science, which combines research, artistic production and community engagement. From 2006–2012, he was area lead for participatory sensing at the NSF Center for Embedded Networked Sensing. He is Co-PI and application team lead for the Named Data Networking (NDN) project. Center for Research in Engineering, Media and Performance, University of California, Los Angeles.

Lixia Zhang is the Jonathan B. Postel Professor of Computer Science at the University of California, Los Angeles. She received a PhD degree in computer science from MIT and joined Xerox Palo Alto Research Center as a member of research staff. Her work at Xerox PARC included analysis of TCP traffic dynamics, reliable multicast, and designs of Internet integrated services support. Professor Zhang joined the faculty of UCLA Computer Science Department in 1995. Her research at UCLA started with Adaptive Web Caching (AWC), the design of a global scale web caching system, funded by DARPA and the Internet Distance Map Service funded by NSF. From 1998 to 2010 much of her group's research focus was on the resiliency and security issues in the global routing system and Domain Name System (DNS), and the system challenges in deploying cryptographic protections in global scale open systems such as the Internet. Since 2010, Lixia Zhang has been leading a multi-campus research project on the development of a new Internet architecture called Named Data Networking (NDN) funded by NSF. Department of Computer Science, University of California, Los Angeles.

Takahiro Yoneda received a B.S in Science and Technology from Ritsumeikan University in 2000 and an M.S. in Information Science from Nara Institute of Science and Technology in 2002. In 2002, he joined in Matsushita Electric Industrial Co. Ltd., currently known as the Panasonic Corporation. He is engaged in research for network congestion control, and development of video conference systems. Advanced Research Division, Panasonic Corporation, Osaka, Kadoma city.

Ryota Ohnishi received a B.E. in Electrical and Electronic Engineering form Kyoto University in 2010 and an M.E. in Informatics from Kyoto University in 2012. In the same year, he joined the Panasonic Corporation, and since, has engaged in research and development of networking technology. Advanced Research Division, Panasonic Corporation, Osaka, Kadoma city.

Eiichi Muramoto joined Matsushita Electric Industrial Co. Ltd. in 1991. He received an M.S. degree from the School of Information Science at Japan's Advanced Institute of Science and Technology in 2000 (company dispatched). He is currently working on networking for the Panasonic Corporation. He is also a member of the WIDE project. Advanced Research Division, Panasonic Corporation, Osaka, Kadoma city.