Data-Centric Video for Mixed Reality

Peter Gusev, Jeff Thompson, Jeff Burke
UCLA REMAP
Los Angeles, USA
{peter, jefft0, jburke}@remap.ucla.edu

***Abstract*—Network video streaming abstractions tend to reproduce older paradigms of video dating back to analog broadcast. With IP video distribution becoming increasingly realistic for a variety of low-latency media applications, this paper looks ahead to a data-centric architecture for video that can provide a superset of features from existing abstractions, to support how video is increasingly being used: for non-linear retrieval, variable speed and spatially selective playback, machine analysis, and other new approaches. As a case study, the paper describes the use of the Named Data Networking (NDN) network architecture within an experimental theatrical work being developed at UCLA. The work, a new play, *Entropy Bound*, uses NDN to enable a hybrid design paradigm for real-time video that combines properties of streams, buses, and stores. This approach unifies real-time live and historical playback, and is used to support edge-assisted machine learning. The paper introduces the play and its requirements (as well as the NDN components applied and developed), discusses key design patterns enabled and explored and their influence on the application architecture, and describes what was learned through practical implementation in a real-world production setting. The paper intends to inform future experimentation with real-time media over information-centric networking and elaborate on the benefits and challenges of using NDN in practice for mixed reality applications today.**

***Index Terms*—NDN, Named Data Networking, ICN, Video**

I. Introduction

While video transport and live streaming over TCP/IP networks have become more efficient, ubiquitous and interoperable, solutions have been built almost entirely on abstractions based on a 20th-century broadcast video mindset that focuses on uniform, linear streams transmitted from video sources to receivers over the network. As the percentage of video consumed linearly by humans decreases, this broadcast stream abstraction is less appropriate. For example, in a mixed reality experience, video segments or other media may be retrieved dynamically based on activity detected by sensors in the environment for non-linear playback to interacting audiences. Or, they may be used in machine analysis, for example, to detect and track the motion of objects or people. Various semantic mismatches between contemporary technology's abstractions for moving images and the actual uses of those images creates overall system inefficiencies (especially for video that must be secured), limits innovation by making important types of access and manipulation difficult, and significantly complicates impending transitions to new video formats, including 360-degree, light-field, and volumetric capture. In this paper, we explore how a data-centric approach to video—in this case, built on the Named Data Networking (NDN) network architecture [1]—can support modern uses of video as well as future media formats.

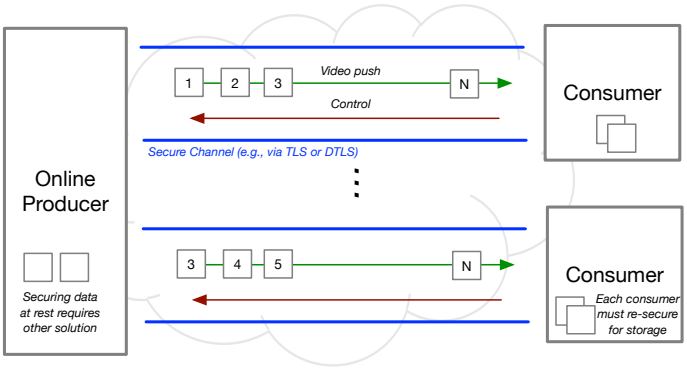

Fig. 1. Typical IP Video abstraction: File or topic granularity request/response; protocol control of content addressing.

Applications today store, analyze, and process video in a variety of ways, ranging from manipulations as simple as playing in reverse or at multiples of normal speed for human viewers, to applying machine learning (ML) selectively to specific areas of many feeds from spatially calibrated cameras. Despite such diversity, tools for retrieving and manipulating video across IP networks rely almost entirely on legacy abstractions, such as those shown in Figure 1. They are often optimized around linear playback of one stream at constant speed and uniform size. For applications requiring more granular access or different types of playback, the only accessible solutions involve transferring entire videos (or, if properly encoded, DASH segments [2]) or, for live video, acquiring and buffering whole frames or groups of pictures (GOPs), and then manipulating locally. In order to support this, applications must rely on custom solutions and/or advanced, less-supported options in contemporary protocols (e.g., Spatial Relationship Descriptors in DASH [3]). Custom solutions must also be used for multidimensional random access to video, such as selectivity in frame space, time, quality, or field of view (e.g., in 360-degree virtual reality). If one takes security into account, even fewer off-the-shelf solutions are available. NDN provides a platform on which to build a data-centric approach as shown in Figure 2, in which granular, individually secured objects making up a video stream can be accessed directly by name as needed by various applications without relying on a channel-based abstraction and taking advantage of NDN architectural benefits, including intrinsic multicast and

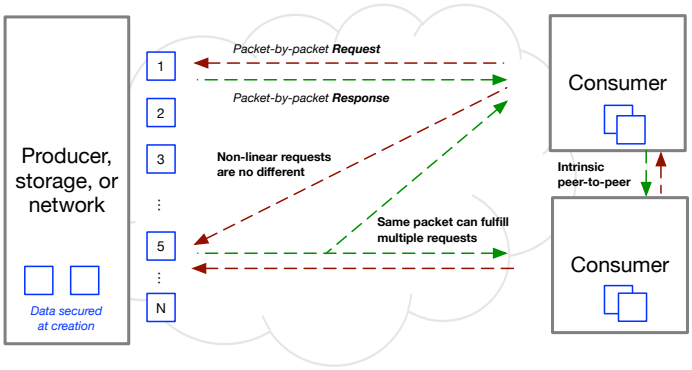

Fig. 2. NDN (Data-centric) Video abstraction: Packet granularity request/response; application control of protocol naming.

data-centric security.

The next section briefly introduces NDN and application research in video streaming, which is generalized to articulate a data-centric video architecture in Section III. Section IV describes the experimental mixed reality theater production *Entropy Bound* used as a case study, followed by a brief comparison with a hypothetical TCP/IP-based approach (Section V). In the remaining sections, we describe the details of the NDN-based implementation used to realize the case study, including new and revised libraries, and conclude with a brief discussion of what was learned and potential future work.

II. Background

A. Named Data Networking

NDN [1] is a proposed network architecture that forwards data directly based on application-defined names. NDN provides request-response semantics at the network layer that are similar to web semantics, but at packet granularity. It does this without requiring host addressing or name-to-address mappings, such as those provided by the Domain Name System (DNS). NDN makes named data the new "thin waist," or common interoperability layer. Each data packet is bound to its name by a cryptographic signature or similar mechanism. Stateful forwarding is used to route packets through the network without source and destination addresses [4]. With NDN, the common stack used by all nodes provides capabilities normally left to middleware or higher-level frameworks, thus simplifying application design and network management, and reducing the attack surface. NDN secures data directly at production, decoupling security from both middleboxes (e.g., CDNs) and the channels over which data are delivered. NDN's network layer forwarding of immutable data, using the application-assigned names, provides several benefits, a few of which we highlight here:

a) Intrinsic multicast: NDN is intrinsically multicast, aggregating duplicate requests at intermediate nodes and answering the same requests with the same data packets (within the limits supported by network caches), limiting load on the sources. Intrinsic multicast support enables developers to rethink traditional channel and file-based video models, and to embrace modular, distributed architectures without security compromises.

b) In-network storage: The standard packet format enables any node to store any data and, if desired, respond to requests for it. This is made possible by data immutability and data-centric security. While on-path caching has traditionally been a focus of NDN and information-centric networking research [5], this is just a subset of a broader range of in-network storage options enabled by NDN.

c) In-network processing: Similar to in-network storage, the use of standard packet formats and consistent naming conventions allows NDN to treat processing as a means to retrieve dynamically generated data seamlessly. Consumers use properly constructed names to request results of data processing, and the network delivers this request to the appropriate processing unit (e.g., closer to the data to be processed).

B. Data-centric video

In NDN, names define access to the data. Providing semantically rich naming on the content producer side enables consumers to access what data they want easily on a per-packet basis. Early work by our group in live and streaming video [6] explored benefits to application authors. In that work, we were particularly interested in the ability to seek randomly to any frame in an appropriately named stream simply by changing the name being requested. We also wanted to pursue a unified approach to playout of live and pre-recorded streams, which is closely connected to the approach taken here.

The NDN video playout and streaming applications described in [6], [7], provided random access to video according to frame number rather than a chunk of bytes—a departure from the approach taken in DASH. Providing unambiguously sequenced naming schemes for multimedia content enables consumers to start streaming media at any point in time and/or at a rate slower or faster than the original source, *with the first request packet they issue*. This also allows for seamless switching between historical and live content, on-the-fly editing and looping/reverse playback capabilities that are important for video analysis and storage. For example, for an application that wants random access to video based on UTC time—say to pull five seconds every Thursday at 17:00—data objects mapping frame numbers to UTC date/time can be stored with the video and used as packet redirects for the first retrieval, adding only one packet of fetching overhead.

C. NDN-RTC: Low-latency data-centric video

For live and playout applications with larger buffer sizes, it is possible to use HTTP (or QUIC) to approximate this name-based random access and host-independent behavior at the application layer if an application can tolerate higher latency and/or data chunk size. By operating at packet granularity, the NDN-based approach can support both low-latency applications as well as latency-tolerant (and disruption-tolerant) applications more efficiently. Furthermore, it provides a more effective approach for multiparty and channel-independent security, as well as disruption tolerance and other benefits

described below. More recently, we have pursued the above benefits for low-latency (or "soft real-time") streaming with subsecond buffer sizes and subframe granularity, not just playback with long buffer times. This work has resulted in NDN-RTC, an implementation of video and audio real-time communication (RTC) over NDN, as described in [7].

Real-time communication of audio, video, and other signals have been a regular area of study, from early work in secure voice communication [8]–[10] and instant messaging [11], which influenced NDN-RTC. The original NDN-RTC library was designed and developed to support multi-party audio/video conferencing use cases. The library provides functionality for low-latency streaming of audio and video, loss resilience through forward error correction, and synchronized audio/video playback and data verification—enabled by NDN. It uses the VP9 video encoder for video streaming and WebRTC's audio processing pipeline for the out-of-the-box echo cancellation mechanism. A number of applications were developed using NDN-RTC, including the GUI multi-party conferencing tool *ndncon* [12], headless clients for macOS and Ubuntu, and their containerized versions for easier and faster deployment [13].

From the onset of the NDN-RTC project, the library was designed with the requirement of no direct producer-consumer communication, enabling the number of consumers to scale based on network rather than producer capability. Data consumers request data from the network independently of whether it must come from a producer, a network cache or a historical repository. This shifts packet-by-packet decision-making to the consumer side of the application by making it responsible for deciding *what* data to request in order to meet its needs. Such an architectural paradigm shift, as illustrated in Figure 2, enables a number of appealing capabilities for real-time interactive distributed systems. Several—including seamless live-historical streaming and granular, source-independent data access—will be discussed later.

D. Edge-supported augmented reality

While NDN-RTC was originally developed for a new conferencing application, low-latency video streaming has many uses, which led us to generalize it into a library used in a variety of ways, including video transport in an edge-supported augmented reality (AR) application. AR is a prominent example of how cameras are used as sensors; video streams are analysed to generate device odometry and other scene information whether or not their POV is presented to the end user. For such applications, we are interested in how a data-centric approach can be used for the contextual data gathered by cameras. This is being explored in the ongoing "ICN-Enabled Secure Edge Networking with Augmented Reality" (ICE-AR) project [14], which uses decentralized augmented reality as its driver application.

In ICE-AR, using NDN-RTC, mobile devices continuously transfer users' *context* (POV video and available metadata—e.g., IMU data) to edge nodes running ML processes in real time. The raw video is processed to generate a semantic description of the current environment—the *deep context*. Different edge nodes provide various subsets of context extraction services. Deep context is then used by the mobile client to retrieve relevant content from the cloud content providers and overlay it on the POV video.

III. Data-centric Video Architecture

In this section, we describe a generalization of the above past work into a data-centric architecture for video. This high-level perspective is important but not sufficient for real-world systems. Details of supporting research in adaptive rate control, network-supported congestion control, and generalizing protocols for real-time data retrieval can be found in [15], [16], and [17], respectively.

A. Data-centric design patterns

We first identify a series of data-centric design patterns for application development, which are achieved at upper layers in, for example, many IP cloud applications, enabled at a lower layer by NDN, and inspired by our approach with NDN-RTC. Prior work [18] discusses the following patterns for the use case of tactical applications and systems—a specific case of the more general needs of dynamic and mobile environments: a) create host-independent behavior; b) embrace multicast; c) enable storage everywhere; d) communicate aggressively and opportunistically; e) share namespaces, not connections; f) secure data first. For the purpose of this paper, we focus on the specifics of *host-independent behavior* for video, and occasionally touch on how this is interwoven with the other patterns described above.

a) Data immutability: In NDN, when a data packet is published, it is named uniquely, and can not be altered as it is cryptographically bound to its name. Thus, a requester can get the data from any node and be sure it is correct. The implications of immutable data are discussed in [19]. In NDN, because individual packets are secured (signed and optionally encrypted) at creation, the immutable data can be distributed independently of the security of any particular channel.

b) Hierarchical naming: Though NDN data names can be any string of bits, forwarding is designed to take advantage of hierarchical structure, just as in IP. The same hierarchy can be used by the application to convey relationships between data and by the network for efficient forwarding. Requests (Interests) can carry incomplete names (prefixes) for data discovery purposes.

c) Application-level framing: Using the names, data is broken into Application Data Units (ADUs) that are represented directly on the network level, subject to the underlying physical layer packet limits. Application-level framing in the context of multicast is discussed in [20].

d) Receiver-driven multicast: In NDN, consumers request the ADUs they need as they need them (at packet granularity), a version of receiver-driven multicast [21]. Requesting independently verifiable immutable data via receiver-driven multicast enables requests to be aggregated and responses to be cached. Multicast behavior is provided implicitly in NDN through data caching and request aggregation.

e) Simultaneous access: Interest aggregation and data caching on NDN nodes enable "Reverse CDN" (rCDN) behavior [22], where live data leaves the producer once, and gets multicast to any consumer that requested it.

B. Data-centric video concepts

The above patterns, applied to low-latency video dissemination, help us articulate several underlying concepts for the proposed data-centric video architecture.

1) Granular random data access: Real-time video delivery in distributed applications usually means that data is transferred between hosts as a stream or a channel; a collection of node-to-node channels in a chain might be called a pipeline. Pipelines illuminate a key concern about sender-driven approaches: They start at the beginning of a chain, and provide little agency at the receiving end, which is especially important in scenarios with many heterogeneous receivers reading from different points in the pipeline. Moreover, pipelines contribute to limitations and brittleness of applications: 1) both communicating parties must be present in the network to establish communication; 2) exceptional situations due to link failures require re-synchronization between parties for recovery; 3) data, streamed over the pipeline, is not accessible until it has reached its destination.

In data-centric design, streaming is achieved by consuming ADUs at a certain rate. For instance, a video stream over NDN can be represented as a sequence of uniquely named frames that can be fetched independently. This shifts the responsibility of *what* is fetched onto data consumers. Consumers are decoupled from data producers, and act independently, without direct producer-consumer synchronization. *Data immutability* and *application-level framing* allow accessing data randomly and at application-defined granularity levels. Consumers are able to fetch only those elements that suit their needs, by requesting a complete set or a subset of data generated by the producer. An example NDN naming for packets of typical video streams in time, space, and quality (for example, as in enhancement layers of scalable video coding) is shown in Figure 3. Different applications could use different naming schemes and/or be standardized within domains.

2) Transparent storage: In data-centric design, it is no longer important *where* data is coming from, as long as it has an expected, verifiable origin, is *immutable*, and matches the data request, i.e. is *named uniquely*. Data producers simply put data into network-accessible (possibly in-memory) storage intended to respond to requests and "forget" about it; delivery is handled upon request, by the network. Consumers send name requests to the network, receive data back, and are able to verify its origin. This data may also be fetched, stored and re-provided by third parties later. Because of NDN data is immutable, independent persistent storage nodes may fetch data from the producer and store it as is on the wire—as named, signed data packets. Storage nodes serve data by satisfying incoming Interests from the network.

Once the data has been published, the only difference between stored and live media is that the latter is being

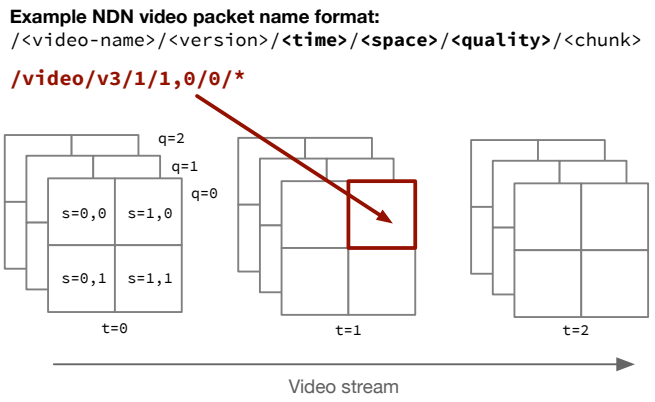

Fig. 3. Multidimensional access to video via named data.

produced as requests are in flight. Consumers designed in a host-independent, data-centric way can use the same retrieval strategy for recorded and live data, as long as they have a mechanism to know the name of the most recent data. (One such mechanism is described in the implementation section.)

Data storage in this architecture is "transparent" in the sense that individual nodes do not need to a) store produced data explicitly as long as persistent storage nodes fetch this data, b) discover and synchronize with producers in order to retrieve data, or c) access and process stored vs. live data differently. Transparent storage helps to make data-centric applications more scalable and disruption tolerant, compared to their host-centric alternatives.

C. Resulting approach for data-centric video

From the above, we observe that a data-centric video abstraction has qualities of a message **bus**, a video **stream**, and a key-value **store**. Video consists of ADUs, like "immutable grains" of video that, once produced, can be stored and accessed at any time and order across the full spectrum of granularity, including in a streaming fashion. Nodes of an interactive system communicate through this hybrid architecture by requesting data, whether it is live or historical. Production and consumption operate in an asynchronous, granular, and distributed fashion, where each node's function can be scaled independently and performed on its own timing, as long as the required data input is available or expected to be generated in the nearest future. Such a system is data-driven. No tightly coupled full-frame streaming pipelines need to be established between communicating parties in order for it to function. Instead, a receiver-driven approach pulls just the data needed where it needs to go (processing along the way if necessary); the availability and specific need for data at each component drives efficient operation of the system. Data persistency is enabled by introducing storage nodes that enable ubiquitous and seamless access to live and historical data.

IV. Case Study: *Entropy Bound*

Mixed reality integrates digital media and physical space through a combination of sensors, actuators, and processing

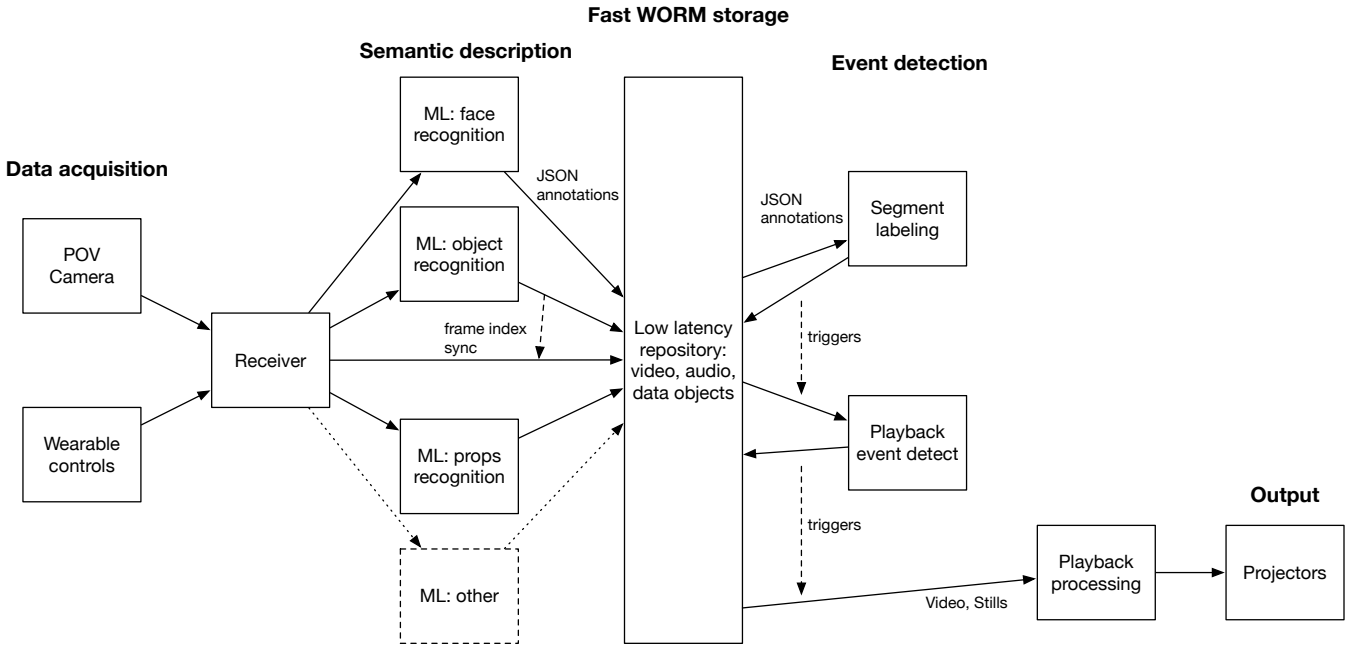

Fig. 4. Data-centric system architecture for the *Entropy Bound* case study.

components. Just as in AR, video is used as both *content* and *context*, often with granular access needs. Further, MR can be supported by offloading computations of media I/O or user devices for real-time analysis to provide deeper, device-independent contextual data, what we call *deep context*. For example, an MR environment may provide ML-powered services to describe scenes semantically, observed by the user's device. We envision mixed reality as an ecosystem that includes end user devices, local deep context providers, and cloud content providers, which offer MR content in exchange for the user's context. An MR ecosystem of this kind is a distributed system with independently operated, horizontally scalable components, and requires intensive data exchanges between individual nodes. Data-centric video over NDN offers unique benefits for such systems. To drive our design of data-centric solutions for ICE-AR, we have developed prototype mobile AR applications that demonstrate context-content exchange. We have also developed workstation-based MR applications that explore data-centric video architecture and use NDN directly over Layer 2, without the power, bandwidth and computational constraints of mobile devices. We focus on one such MR application here.

An example that serves as our case study is the experimental theater piece *Entropy Bound*, which uses machine learning to learn from and influence the dialogue and actions of the lead character through media—progressively, as it is rehearsed and performed.[1] In this work, a fictional brain implant is realized through machine learning components that analyze, store, and recall video from a wireless point-of-view camera worn by the lead character, augmenting performance space with projections seen by the audience in real time. This application has much in common with our vision for augmented reality: A wearable device offers real-time video to edge nodes for ML processing, storage and recall. To fulfill its role in the story, the implemented system has to simulate machine *intelligence*, be *responsive* to the input, and maintain *memory* of past interactions. We enumerate these in more technical terms as follows:

a) Modular machine learning components: A number of machine learning algorithms (e.g., object recognition and scene segmentation) must be run in parallel to generate descriptive and structural metadata for the POV camera feed from the lead performer. The resulting metadata must be stored and accessed by other nodes of the system. Figure 4 shows high-level architecture of the *Entropy Bound* system designed for NDN. As input data is acquired from the sensors (POV camera and wearable controls), it is published over NDN. A *Receiver* node packetizes data according to the application logic and assigns each packet a unique name. This data is placed into in-memory cache of the producer and made immediately available over the network. It is fetched asynchronously by a multitude of nodes for ML processing, storage, analysis and decision-making.

b) Historical video as a first-class entity: One of the premises of the show is that the fictional "implant" is assisting its user to remember things, and triggers memories that are deemed relevant to the present moment. Thus, the system must be capable of storing sensor data (including video and generated data) and making it accessible for later recall during

[1]This experimental comedy follows a highly determined, at times obsessive-compulsive urban professional who has suffered a traumatic brain injury, and is now living life with a first-generation digital brain implant. [23]

Fig. 5. Photo from the UCLA workshop of the new play *Entropy Bound* by Jeff Burke and Jared J. Stein.

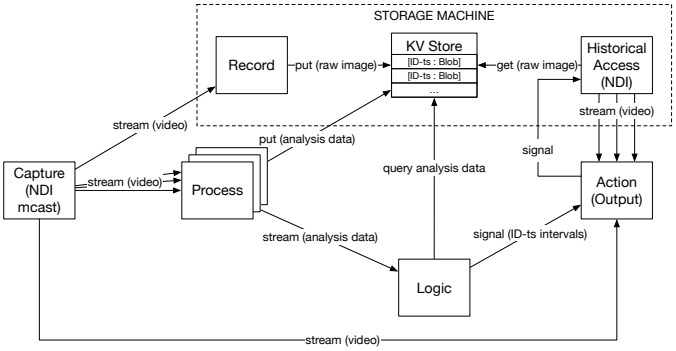

Fig. 6. Hypothetical high-level system architecture for *Entropy Bound* using a IP-based approach.

the show run or rehearsal. Since any memory can be evoked at any moment, the data must be available for recall immediately after storage. The *Fast WORM* (Write Once Ready Many) storage is central to the system.

c) Soft real-time responsiveness: Responsiveness is crucial to presenting the show's ideas to the audience in an efficient and playful way. This creates real-time processing requirements for the system. The main logic of the show is realized through the *Segment labeling* and *Playback event detect* modules which operate on real-time data and produce decision-making data on-the-fly for the *Playback processing* node, responsible for mixed reality projections into the stage space for the audience. For this case study, we focus on detecting scene boundaries of the POV video based on generated semantic annotations, and retrieving semantically similar scenes ("memories" of the fictional implant) from the storage for playback.

d) Prototyping and designer-friendly approach: The real code has to be running and evolving alongside the rehearsals and, eventually, production runtime. This requires support for fast prototyping and "experiment-friendly" plug-and-play system architecture. This leads to a system concept for the show that focuses on a data-centric multi-node design, which can be extended and modified quickly with new nodes. We also had an explicit goal to develop tools and approaches that aid iteration. Both are described in more detail later.

V. Comparison with TCP/IP

For comparison, we briefly present a hypothetical design for how we would have approached this case study over IP.

In a hypothetical high-level design for the system over IP, we assume using NDI [24] for streaming video over the network and a key-value store (such as RocksDB) for storage. NDI provides low-latency, multicast streaming of HD quality video. From the *Capture* node, video is multicasted to multiple nodes that carry *Record*, *Process* and *Action* functions. The *Record* node stores video in the KV *Storage*. A key naming scheme must be designed for the KV *Storage* to address video frames uniquely based on timestamps, streams, and independent show runs, as well as to cross-reference metadata with the image data it relates to. For example, one possible key naming scheme could look like the following:

- $< run\_id >:< timestamp >: video \rightarrow [image]$
- $< run\_id >:< timestamp >: ml :< ml\_id > \rightarrow [data]$
- $< run\_id >:< timestamp >: decision :< idx > \rightarrow [data]$

This way, by knowing the key prefix formed by show run number run_id, and video frame timestamp $timestamp$, one can retrieve image data and all accompanying metadata for this frame.

Processing nodes (which can be added and removed at runtime) put analysis data into the KV Storage. It is also streamed (via some UDP messaging mechanism) to the *Logic* node that makes show decisions. The *Logic* node performs queries on the *Storage* for the previously recorded historical data. The decisions made are then signaled to the *Action* node as a list of important intervals of relevant historical data. Finally, the *Historical Access* node, after receiving signals from the *Action* node, performs fetching of historical raw image data from the *Storage*, and sets up temporary NDI streams for consumption by the *Action* node.

This design illustrates challenging choices one has to make while designing such a system for IP. Fundamentally, there must be a hard-coded or software-brokered mapping relationship between IP addresses of nodes performing various processing, storage, and I/O functions. This is not trivial for networks in which those components are often added/removed regularly, and becomes even more challenging in other scenarios with mobile devices. Additionally, video storage becomes a system's bottleneck and introduces a number of problems. Since encoded NDI stream elements are not immutable and secured, unifying live and pre-recorded streams presents challenges. In practice, to be "re-streamed" at a later time, they have to be decoded, and raw image data must be stored, then resent upon request, which poses scalability challenges and adds delay. To minimize read-write access latency, the *Record* and *Historical Access* nodes must be deployed close to the actual KV-Store. Finally, even though, to the application, repeated historical playback may seem like the same bits

of data are accessed, this assumption does not hold for the underlying network infrastructure. The protocol is conceived of as creating a channel, rather than delivering individually usable elements of video. In practice, originally encoded live data, decoded for storage, has to be encoded and streamed again for historical access. On the network, these streams are represented as newly generated data packets, unrelated to the stored or live data, which results in excessive bandwidth usage and redundant computations for the system.

While we assume that video streams (both live and historical) can be discovered and accessed by multiple nodes through, for example, mDNS and UDP multicast, there is still the issue of synchronization and signaling support between system components that needs to be addressed. For brevity, we have not discussed signaling and synchronization of processing here, but these tend to require additional protocols and middleware components in comparison with the approach we take with NDN in Section VI-A1. In the IP approach, the need to manage hosts and channels creates a semantic disconnect between the network abstraction and the application architecture. We believe a data-centric paradigm for video can close this gap and increase scalability, robustness, and simplicity of deployed mixed reality and other video-heavy systems.

VI. Implementation

The data-centric video system built for *Entropy Bound* starts with existing NDN tools and libraries, namely: a) NDN Forwarding Daemon (NFD) for enabling NDN connectivity; b) NDN Common Client Libraries (NDN-CCL) for Interest-Data exchange; and c) NDN-RTC library for low-latency video dissemination, described in Section II-C.[2]

A. The application's named data

1) Messaging and general data exchange: For messaging and general exchange of non-video objects, we standardized a format for publishing objects of arbitrary type and size that can be efficiently verified. These enable a "learn first–fetch later" approach: Data structure can be learned without fetching the full object, as required by the data-centric video architecture principle of granular random data access. The resulting "Generalized Object" (*GObj*) and "Generalized Object Stream" (*GObjStream*) name schema (see Figure 7) were implemented in NDN-CNL, described below, and used in the case study for exchanging various data between participating nodes—from frame-level annotations, represented as JSON dictionaries, to still images and text data captured from system console outputs. Finally, the $\_manifest$ packet is used to verify unsigned data segments efficiently. The *GObjStream* abstraction provides a convenient API for publishing continuous streams of *GObj*, which complies with the Real-time Data Retrieval (RDR) protocol [17].

2) Data-centric video format: The above design efforts in namespace generalizations presented an opportunity to redesign the original NDN-RTC namespace. A new namespace

[2]See named-data.net/codebase/platform/ for code and details.

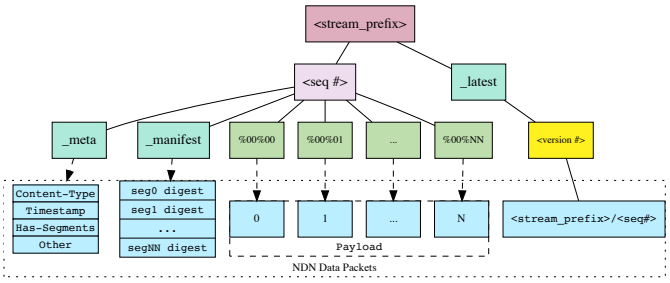

Fig. 7. Generalized Object Stream Namespace

(see Figure 8) provides better support for granular random access and interoperability between heterogeneous consumers. As shown, frame objects conform to the generalized object namespace by providing $\_meta$, $\_manifest$ and data segment packets. FEC data is preserved by using a dedicated $\_parity$ name component. Frames are numbered sequentially, and $\_latest$ component provides a pointer to the latest published frame for consumer bootstrapping using RDR protocol.

Generalizing the namespace allows reuse of code that already implements formats and fetching patterns like *GObj* and *GObjStream* to fetch NDN-RTC data. Compliance with these namespaces will enable data consumption by generalized consumers such as repos or inspection tools, with no further changes in the format or the tool. The new namespace also separates payload data and metadata by publishing it under different names,[3] allowing consumers to "learn first–fetch later." Additional data (like $\_gop$ or $\_parity$) is used by specialized NDN-RTC consumers, or can be optionally fetched by general clients at will. By providing semantically expressive, granular and standardized naming for the data, the data-centric approach enables a high level of flexibility and interoperability.

B. Meeting application requirements

Building on top of NDN-RTC enabled us to start with a data-centric video approach that met the requirement of real-time responsiveness. Below, we discuss how we provided a designer-friendly approach, unified historical and live streaming, and synchronized edge-based machine learning.

1) Designer-friendly approach with NDN-CNL and Touch-NDN: As part of the ICE-AR project, we aim to develop APIs with higher-level abstractions than packet-level request-response, to encourage data-centric design, which we tested in this case study. The NDN Common Name Library (NDN-CNL) offers a new API that is discussed in more detail in [25]. It aims to provide applications with an interface that emphasizes the importance of data naming, and offers a thin layer of abstraction that reduces or eliminates the need to write code for common packet level behaviors.

NDN-CNL is built around the central abstraction of a *Namespace*, which represents one or more data packets pub-

[3]For previous versions of NDN-RTC, metadata was embedded with the encoded frame payload.

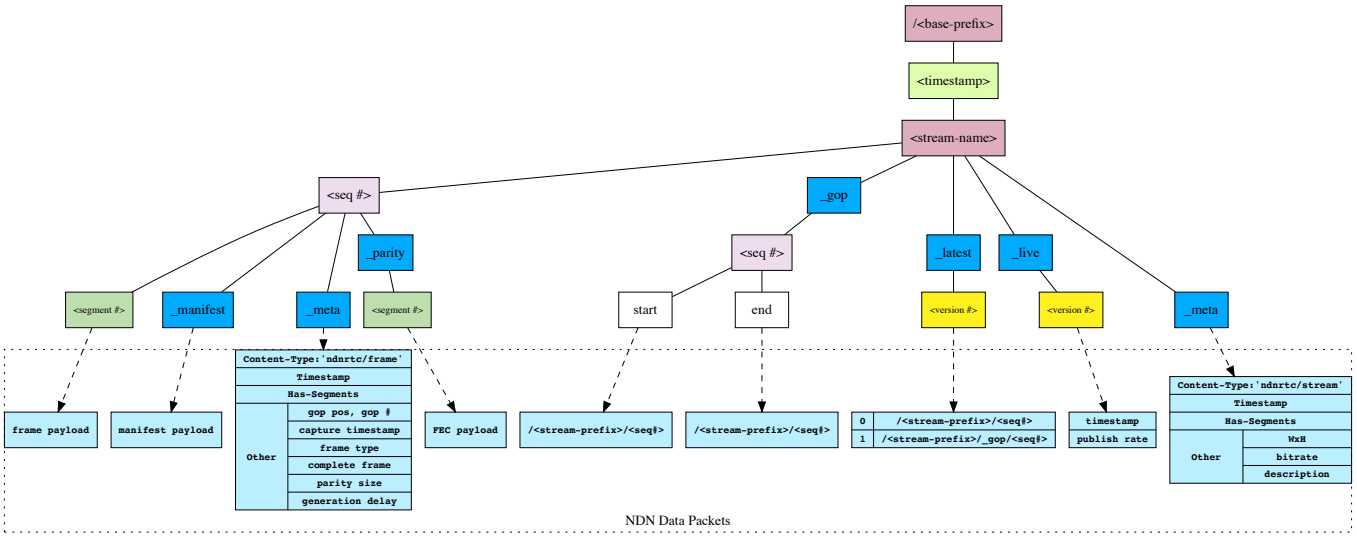
/<base-prefix>
<timestamp>
<stream-name>
<seq #>
_gop
_parity
<seq #>
_latest
_live
<segment #>
_manifest
_meta
<segment #>
start
end
<version #>
<version #>
_meta
Content-Type:'ndnrtc/frame'
Timestamp
Has-Segments
Other
gop pos, gop #
capture timestamp
frame type
complete frame
parity size
generation delay
frame payload
manifest payload
FEC payload
/<stream-prefix>/<seq#>
/<stream-prefix>/<seq#>
0 /<stream-prefix>/<seq#>
1 /<stream-prefix>/_gop/<seq#>
timestamp
publish rate
Content-Type:'ndnrtc/stream'
Timestamp
Has-Segments
Other
WxH
bitrate
description
NDN Data Packets

Fig. 8. NDN-RTC namespace

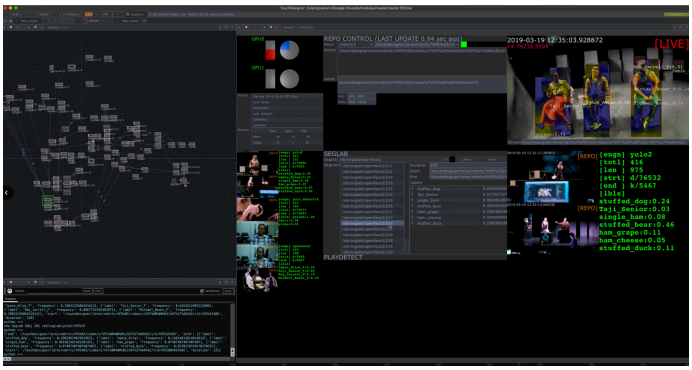

Fig. 9. TouchDesigner integration for NDN.

lished under an arbitrary hierarchical name. The actual logic of how data is published or fetched is separated into classes built on a *NamespaceHandler* base class. Applications act on and receive events associated with a hierarchical namespace through processing handlers that are associated with specific name prefixes and implement common functions. Each namespace node is controlled by the same state machine that includes hooks for data serialization/deserialization, signing/verification, encryption/decryption, and so on. This allows client code to work with data-centric video and messages in a way more aligned with the application logic, rather than at the level of request-response packets, without loss of benefit. For example, a *SegmentedContent* handler allows serializing and packetizing large objects into smaller, named segments. On the consumer side, the resulting de-packetized, deserialized object is asynchronously delivered to the client code.

While the NDN-CNL provides a high-level programming abstraction, it does not provide any tools for dealing visually with video streams and other data. This limits both the user base and the speed of prototyping. We integrated NDN support (via the CCL and CNL) into Derivative's TouchDesigner, a powerful real-time media processing environment, through a set of Python modules and C++ plug-ins that we call the "TouchNDN" framework. (See Figure 9.) The ability to program and experiment with this framework interactively while simultaneously building the actual system introduced "data-centric" networking design principles to the developers, previously unfamiliar with the concept, and shortened the overall development cycles. TouchDesigner enables rapid prototyping through its visual programming approach, where each data processing node can be inspected and adjusted in place to achieve the desired output. The environment employs a data-centric paradigm for processing: Each node operates on data and produces some output that can be fed to another node. Nodes, or operators, are named and can be nested, which results in hierarchical naming. We believe that integrating with TouchDesigner can allow us to stimulate wider and faster adoption of a data-centric networking approach.

2) Unified historical and live streaming: Through NDN's use of *in-network storage* for any data object, discussed in the data-centric video architecture principle of transparent storage, we aimed to unify the approach and interface for accessing live and recorded data. In NDN's receiver-driven model, the only difference between live and recorded data is that for the former, an application issues Interests before the data may be available, so it has to know the names to use. A combination of application-level framing—in which applications, rather than the network decide how to packetize their data—and application-defined, network-forwarded data naming in NDN allows storage to be implemented consistently on any network-attached node. By simply storing secured, application-generated packets, nodes respond to Interests for that data when appropriate. For the *Entropy Bound* system, *fast-repo* [26], a persistent data storage, was implemented using RocksDB [27] as a back-end key-value store. It allows for

efficient, simultaneous storage and retrieval. The application was updated with unified mechanisms for streaming live and historical data from this repo, as well as granular access—fetching individual frames by name. Once the NDN name for a stream or a frame is obtained, the following video can be streamed and played back using internal, or user-defined, external timing. This allows for straightforward, data-centric access to the historical and live data, and was used by a multitude of nodes across the system.

3) Synchronizing video processing: Incorporating and updating ML modules was straightforward: Nodes are configured with the name prefixes of video to process at runtime. We leverage the presence of an explicit frame index name component for video synchronization between independent nodes in the case study system. For real-time semantic video annotations, ML processing nodes generate per-frame metadata that is published under a name derived from the video frame name. For example, for a video frame named:

$$/ < run\_id > / < input\_id > / < frame\_idx >$$

all recognized objects will be published under:

$$/ < run\_id > / < input\_id > / < frame\_idx > /objects$$

This way, nodes that are interested in semantic analysis may fetch the latest annotations without fetching the actual video, or fetch frames selectively, based on received annotation. By making $frame\_idx$ predictable, i.e. sequentially increasing, multiple nodes can synchronize on the latest available data by sending requests for future video frames.

C. System deployment

Our initial implementation focused on a local area network scenario, though many elements could use cloud or MEC configurations (See Figure 10 for a system diagram).

POV video from the wearable camera is delivered wirelessly to the *Capture* machine, where it is compressed and published using NDN-RTC. This video is simultaneously consumed over NDN by multiple machines for storage, ML processing, system control and monitoring. The *Edge* node runs three containerized ML processes for faces, objects, and show props recognition. Resulting per-frame annotations are published as JSON dictionaries using *GObjStream* over NDN. The annotations stream is fetched for detecting scene changes (segmenting) that are stored in a local DB, which is also continuously queried for the top N scenes, most similar to the currently observed frame. This data, published over NDN, was then used by a number of control and monitoring nodes to query the *Repo* for retrieving and displaying actual frames, corresponding to the query results.

The system described was deployed and experimented with during the *Entropy Bound* workshop in December 2018, where it was co-developed alongside the performance and in use regularly during rehearsals. All the nodes were connected to the same LAN segment, and used NDN directly over the Layer 2 network (Ethernet). To minimize manual network configuration, multicast faces were used. Therefore, each node was receiving every other node's requests and responses.

VII. Discussion

A. Hybrid abstraction: Bus, stream, store

Based on our comparison with the hypothetical TCP/IP approach and our experience with the case study, the application was simplified considerably by using a data-centric video architecture compared to the host-centric alternative. Granular random data access and persistency, assumed early in the process, allowed us to focus design efforts directly on the application functionality. System deployment was simplified, as the network itself provided the functions of a message bus and historical data access. Finally, data-centricity allowed for unified data access mechanisms. Code that was designed for live streaming of named data was generalized to access historical data with minimal effort.

Different parts of the system treated video as a message bus, decomposable streams, and a store. We leveraged NDN's intrinsic multicast to enable many consumers of various live video streams, with only a single copy of each moving over the network. This allowed us to think of each stream as a "video bus" that could be tapped by many nodes. At the same time, by updating our codebase to use the same namespaces and network objects for both live and historical playback, we could also tap the bus at different points "in time." Together, these approaches allowed the team to view the video as both a bus and a store at the same time, a very effective hybrid approach. NDN enabled our implementation to be easily expanded with ML processing modules as needed. This effectively enabled edge processing for the real-time data where processing results become immediately and implicitly available over the message bus for other modules' consumption.

B. Naming challenges

Like recent work in mobile health using NDN [28], more sophisticated applications generate more complex namespaces and raise new challenges in managing them. Some specific challenges clarified through the case study include:

a) Developer scaffolding for complex namespaces: Our original namespace design was quite complex as it tried to support many data retrieval patterns. Two practical challenges forced us to simplify the namespaces for the *Entropy Bound* workshop. First, describing even a hard-coded namespace and associated conventions across the code of multiple distributed processes requires some combination of tedious manual coding or asynchronous communication of name hierarchies across various modules with intermittent connectivity. We are exploring how to address the first challenge via a `protobuf`-style language. For more dynamic communication of names, we plan to use namespace synchronization, which is described in the next section as future work. The second challenge is that many dynamic components required shared state across multiple application instances; for this, we plan to add a pub-sub style message bus for this in the next iteration.

b) Instance naming: Due to data immutability, producers must create their data with a name that is unique not only while they are running, but for its expected lifetime on the

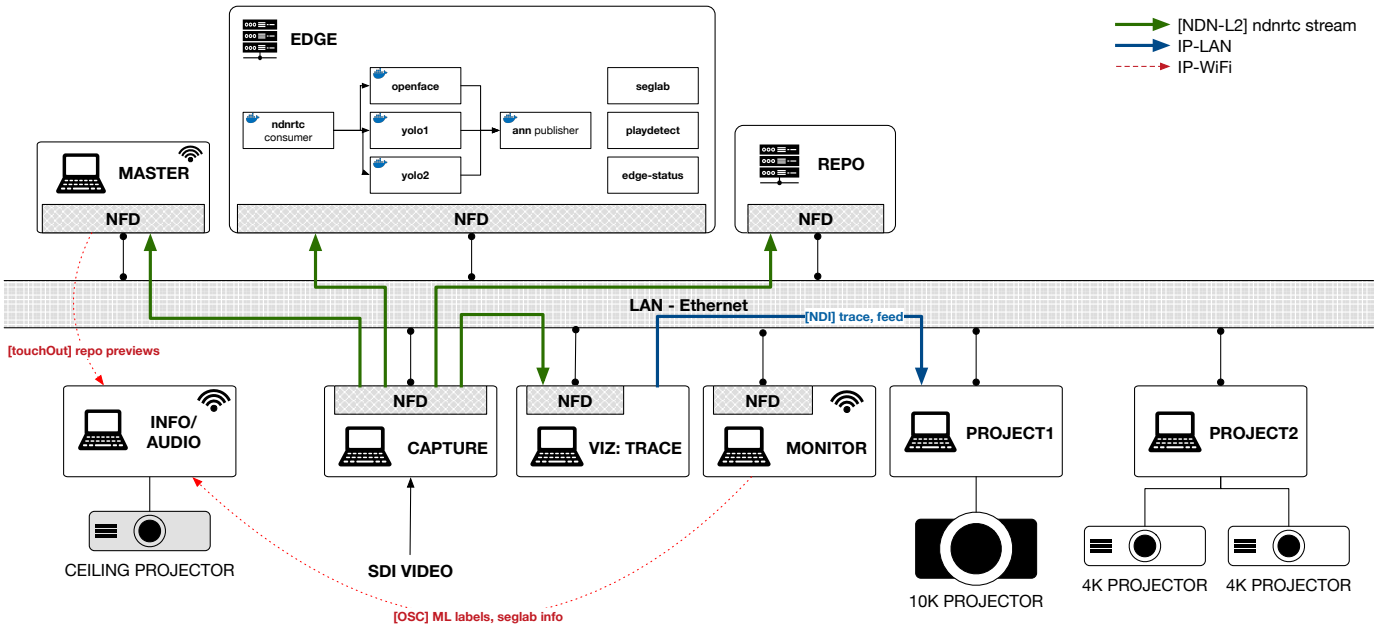

Fig. 10. Entropy Bound deployment diagram: Ubuntu-based processing and storage machines: *Edge*, *Repo*; macOS-based, TouchDesigner machines: *Capture*, *Master*, *Viz:Trace*, *Monitor*; supporting and projection machines: *Info/Audio*, *Project1*, *Project2*.

network and in persistent storage. Our approach used unique application instance identifiers to differentiate data produced between sessions. For example, in NDN-RTC, this is handled by inserting a $< timestamp >$ component into the name:

$$/ < base > / < timestamp > / < stream > / < seq >$$

This way, the producer always names data uniquely, independent of sequence numbers used. Instance naming, however, presents new challenges for discovering live and historical data. Initially, it can be solved by enabling the producer to answer special "rendezvous Interests" with a full name for the data. Eventually, we plan to address this challenge using name synchronization, where multiple parties can send namespaces up to a specific level in the name hierarchy.

VIII. Conclusion

The data-centric video architecture explored here emerged from our goals of exploring unique approaches, benefits, and challenges of using NDN for video in support of augmented reality, and using the resulting toolset to support other ongoing projects, such as *Entropy Bound*. With this approach, we aimed to motivate the design from an emerging area (edge-supported AR) while also applying it in applications that we are already building. Through the latter, as described in the previous sections, we have identified some potential new abstractions and ways of thinking, and identified limitations in available tooling that we plan to address in the future. In addition to addressing specific challenges identified in the previous section, other future work includes:

a) Multidimensional random access: Our initial implementation supports granular access in time, but as shown in Figure 3, there is a conceptually straightforward extension to providing access to spatial regions and enhancement layers by name. In bandwidth-intensive 3D data sets or very high-resolution video, spatial selectivity appears to be an increasingly important capability. Of course, if the media is encoded semantically, using a scene graph or object-oriented description [29], names could potentially be used to make efficient decisions about retrieval prioritization according to evolving scene semantics.

b) Resource discovery: For both semantically organized media in the future and current video formats, we have not addressed how resources are discovered and name schema determined. Name discovery through application-level name synchronization, manifests, name enumeration protocols, and other techniques, as well as name lookup and indirection services are all active areas of NDN research. For a first step in this direction, we plan to integrate depth-limited namespace synchronization [30] to enable consumers to retrieve the name structure for a given prefix to sufficient depth, to determine what data is available for fetching. They may individually proceed to fetch the actual data, as needed.

c) Security: The case study integrated only basic cryptographic signing and verification operations. However, our design is compatible with data-centric security approaches (e.g., schematized trust, name-based acces control, and other approaches [31]–[33]) while preserving granular access to media and other features.

Even considering these three areas of future work—increased consumer-side control over what media elements are needed on a packet-by-packet basis; how to use names to find the data needed; and data-centric security that preserves granular content access—we believe that a data-centric approach to networked video offers a powerful replacement

for existing abstractions that is better suited to mixed and augmented reality applications and emerging media formats.

ACKNOWLEDGEMENTS

We are grateful for the support of the NDN project team, including Lixia Zhang and UCLA IRL. The initial development of *Entropy Bound* was supported by a Google Focused Award and the workshop held at the UCLA Department of Theater; we thank all of the cast, crew, and production staff involved. Portions of this work were supported by NSF Award Nos. CNS-1719403 and CNS-1629922, the Intel ICN-WEN program, and Cisco.

REFERENCES

[1] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, kc claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM Computer Communication Review*, July 2014.

[2] I. Sodagar, "The mpeg-dash standard for multimedia streaming over the internet," *IEEE MultiMedia*, vol. 18, no. 4, pp. 62–67, 2011.

[3] O. A. Niamut, E. Thomas, L. D'Acunto, C. Concolato, F. Denoual, and S. Y. Lim, "Mpeg dash srd: spatial relationship description," in *Proceedings of the 7th International Conference on Multimedia Systems*. ACM, 2016, p. 5.

[4] C. Yi, A. Afanasyev, I. Moiseenko, L. Wang, B. Zhang, and L. Zhang, "A case for stateful forwarding plane," *Computer Communications: ICN Special Issue*, vol. 36, no. 7, pp. 779–791, April 2013.

[5] M. Zhang, H. Luo, and H. Zhang, "A survey of caching mechanisms in information-centric networking," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 3, pp. 1473–1499, 2015.

[6] D. Kulinski, J. Burke, and L. Zhang, "Video streaming over named data networking," *IEEE COMSOC MMTC E-Letter*, vol. 8, no. 4, pp. 6–10, 2013.

[7] P. Gusev and J. Burke, "Ndn-rtc: Real-time videoconferencing over named data networking," in *Proceedings of the 2nd ACM Conference on Information-Centric Networking*. ACM, 2015, pp. 117–126.

[8] 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*. ACM, 2009, pp. 1–6.

[9] Z. Zhu, S. Wang, X. Yang, V. Jacobson, and L. Zhang, "Act: audio conference tool over named data networking," in *Proceedings of the ACM SIGCOMM workshop on Information-centric networking*. ACM, 2011, pp. 68–73.

[10] Z. Zhu, J. Burke, L. Zhang, P. Gasti, Y. Lu, and V. Jacobson, "A new approach to securing audio conference tools," in *Proceedings of the 7th Asian Internet Engineering Conference*, ser. AINTEC'11, 2011.

[11] Z. Zhu, A. Afanasyev, Y. Yu, and L. Zhang, "Chronochat: a serverless multi-user instant message application over ndn (poster)," in *NDN Community Meeting*, Los Angeles, CA, September 2014.

[12] "https://github.com/remap/ndncon."

[13] "https://github.com/remap/ndnrtc."

[14] J. Burke, "Browsing an Augmented Reality with Named Data Networking," in *2017 26th International Conference on Computer Communication and Networks (ICCCN)*, July 2017.

[15] P. Gusev, Z. Wang, J. Burke, L. Zhang, E. Muramoto, R. Ohnishi, and T. Yoneda, "Real-time streaming data delivery over Named Data Networking," *IEICE Transactions*, May 2016.

[16] K. Schneider, C. Yi, B. Zhang, and L. Zhang, "A practical congestion control scheme for Named Data Networking," in *Proc. of ACM ICN*, 2016.

[17] S. Mastorakis, P. Gusev, A. Afanasyev, and L. Zhang, "Real-time data retrieval in named data networking," in *2018 1st IEEE International Conference on Hot Information-Centric Networking (HotICN)*. IEEE, 2018, pp. 61–66.

[18] J. Burke, A. Afanasyev, T. Refaei, and L. Zhang, "Ndn impact on tactical application development," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 640–646.

[19] P. Helland, "Immutability changes everything." *ACM Queue*, vol. 13, no. 9, p. 40, 2015.

[20] S. Floyd, V. Jacobson, S. McCanne, C.-G. Liu, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," *ACM SIGCOMM Computer Communication Review*, vol. 25, no. 4, pp. 342–356, 1995.

[21] S. McCanne, V. Jacobson, and M. Vetterli, "Receiver-driven layered multicast," *ACM SIGCOMM Computer Communication Review*, vol. 26, no. 4, pp. 117–130, 1996.

[22] H. Moustafa, E. M. Schooler, and J. McCarthy, "Reverse cdn in fog computing: The lifecycle of video data in connected and autonomous vehicles," in *2017 IEEE Fog World Congress (FWC)*. IEEE, 2017, pp. 1–5.

[23] J. Burke and J. J. Stein, "Live performance and post-cinematic filmmaking," in *Co-Presence with the Camera special issue of Performance Matters*, L. Hunter, Ed., 2020.

[24] "https://www.newtek.com."

[25] J. Thompson, P. Gusev, and J. Burke, "Ndn-cnl: A hierarchical namespace api for named data networking," in *(in submission)*.

[26] "https://github.com/remap/fast-repo."

[27] "http://rocksdb.org/."

[28] H. Zhang, Z. Wang, C. Scherb, C. Marxer, J. Burke, L. Zhang, and C. Tschudin, "Sharing mhealth data via named data networking," in *Proc. of ACM ICN*, 2016.

[29] D. Williams, J. Wyver, and M. Glancy, "Evaluating the potential benefits of object¬-based broadcasting," *Retrieved December*, vol. 5, p. 2017, 2016.

[30] T. Li, W. Shang, A. Afanasyev, L. Wang, and L. Zhang, "A brief introduction to ndn dataset synchronization (ndn sync)," in *MILCOM 2018-2018 IEEE Military Communications Conference (MILCOM)*. IEEE, 2018, pp. 612–618.

[31] Y. Yu, A. Afanasyev, D. Clark, kc claffy, V. Jacobson, and L. Zhang, "Schematizing and automating trust in Named Data Networking," in *2nd ACM Conference on Information-Centric Networking (accepted)*, September 2015.

[32] C. A. Lee, Z. Zhang, Y. Tu, A. Afanasyev, and L. Zhang, "Supporting virtual organizations using attribute-based encryption in named data networking," in *2018 IEEE 4th International Conference on Collaboration and Internet Computing (CIC)*. IEEE, 2018, pp. 188–196.

[33] Y. Yu, A. Afanasyev, J. Seedorf, Z. Zhang, and L. Zhang, "NDN DeLorean: An authentication system for data archives in Named Data Networking," in *Proc. of ACM ICN*, 2017.