Video Streaming over Named Data Networking

Derek Kulinski, UCLA Computer Science, kulinski@cs.ucla.edu
Jeff Burke, UCLA REMAP, jburke@remap.ucla.edu
Lixia Zhang, UCLA Computer Science, lixia@cs.ucla.edu

1. Introduction

Named Data Networking (NDN) is a proposed future Internet architecture that offers many advantages over TCP/IP [1], and holds significant promise for content distribution applications, such as video streaming. The TCP/IP architecture assigns IP addresses to hosts, making the Internet essentially a point-to-point communication system. NDN allows data consumers to retrieve desired content by directly using application-specified hierarchical data names, enabling a general-purpose distribution system. This approach, in combination with NDN's per-packet content signatures, permits any node in the network to cache named data packets and respond to requests for them. This is in sharp contrast to the current IP Internet, wherein a video producer sends data packets directly to every viewer, even when multiple viewers are watching the same video at the same time, and even when those consumers share the same upstream routers where the data could be easily cached.

Not only does NDN enable the use of storage in the network to cache popular data that are frequently requested by multiple users—reducing bandwidth and improving performance—it also enables video producers to easily provide a variety of other functions. For example, producers can assign meaningful names to video data (e.g., timecode frame indexes using a well-established naming convention), so video consumers can simply request specific video content by names to seek (rewind or fast forward). Because of such properties, video streaming can benefit significantly from NDN.

In fact, video streaming has already received considerable attention in the CCN/NDN research community. Early developments include the CCNx VLC plug-in [2] and GStreamer plug-in [3] as test applications. A number of more recent research efforts have produced additional results. For example, Xu et al. compared HTTP live streaming with a CCNx-based approach on Android [8]. Others considered how devices can collaborate to share bandwidth for the same video [9] and rate adaptation [10], as well as additional topics in the context of NDN.

This paper provides an overview of NDNVideo, a complete software solution developed at UCLA for video and audio streaming over NDN that serves as a representative example of how content-centric applications can be implemented in this new architecture. NDNVideo takes advantage of NDN's features to provide highly scalable, random-access video from live and pre-recorded sources using a straightforward data publisher and consumer model without connection negotiation or session semantics. The application was built using PyCCN, Python bindings that we created for the CCNx library and software router by PARC [4, 7].

2. Design Goals

NDNVideo was designed to support live and pre-recorded video streaming with frame-accurate random access, while integrating persistent storage of live content and, eventually, enabling synchronized playback across multiple consumers. Driver applications are web video streaming services as well as applications in digital signage, live events, and professional media production.

Our design goals included 1) quality consistent with current Internet video expectations; 2) simple, low-latency random access into streams, based on actual location in the video stream (by video frame), using a timecode-based namespace (i.e., the hour, minute, second, and frame of the stream); 3) consumer-side synchronization of streams using the timecode-based namespace (for multiple consumers and future digital signage, multi-camera, and interactive applications), and scalability without impact on the original source of video; and 4) on-the-fly archival of live streams, making them indistinguishable from pre-recorded streams for the purposes of most playback applications. Additionally, NDN's per-packet signatures on ContentObjects (data packets) provided a starting point for content verification and provenance in video applications, and we employ them in this application.

We use the GStreamer framework for capturing and rendering video. A GStreamer based application for video streaming was developed in [3], but treats video as an arbitrary binary stream of data, and consequently does not fully use the potential of the NDN architecture. Their approach is similar to streaming video by downloading a file over the HTTP protocol, as opposed to using protocols that leverage the content type, such as RTP. In NDNVideo, we develop a namespace and protocol specifically for streaming video and audio.

3. Architecture

Overview. The NDNVideo protocol has two types of participants: publisher and consumer. The streaming relationship is one-to-many (i.e. a single publisher publishes data that is received by many consumers). Unlike IP, NDN is pull-based, in which data consumers issue request packets, or "Interests" that indicate the

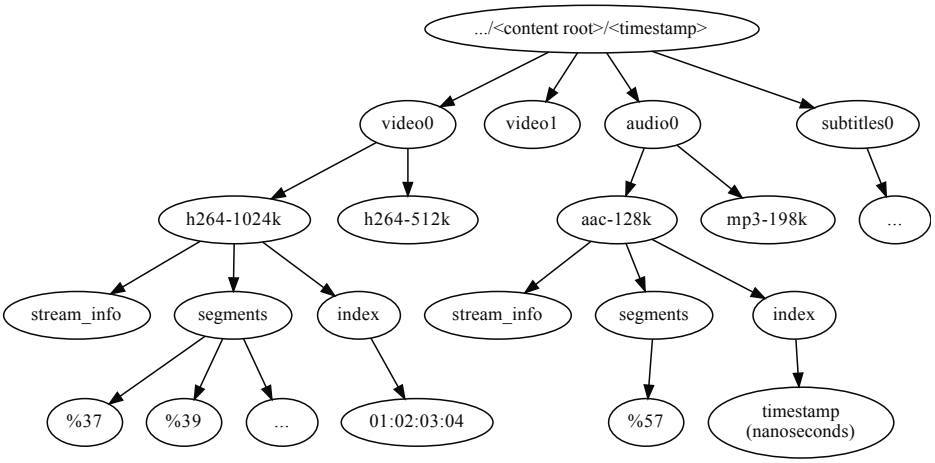

Figure 1. NDNVideo namespace, providing multiple encoding rates and both time and segment-based data access.

name of the content that they wish to receive. Any entity on the network may respond with a corresponding data packet. This request/response approach enables the video publisher and consumers to utilize the Interest-Data exchange of the NDN protocol directly. NDNVideo requires no direct interaction between consumer and producer; the publisher simply prepares data packets, signs them, and stores them in a repository (persistent store) for retrieval by consumers. Consumers no longer need to establish a session with the publisher, nor inform the publisher about their Quality of Service requirements. Instead, they are in full control of how much data they receive, and at what rate. If, for any reason, a consumer needs to upgrade or downgrade its bit rate, it can do so seamlessly by requesting data from a different namespace for the same timecode in the video.

Namespace. Like all NDN applications, a fundamental facet of NDNVideo's design is its namespace, illustrated in Figure 1. During packet preparation, the video producer segments the stream using semantically meaningful names, e.g. frames for video and samples for audio, organized by bitrates and encodings that can be enumerated by the consumer via a metadata file to list all children nodes in a namespace tree. Such structuring of data provides many benefits, one of which is enabling the publisher to uniquely name every frame and, in turn, allows the consumer to easily seek to a specific place in the stream. A key design goal was to provide this random-access via video timecode, in which video is indexed by frames (e.g., HH:MM:SS:FF). To allow more efficient playback after seeking, the data is also provided using consecutive segment names.

Encoding and Packet format. NDNVideo supports all of the many encoding formats supported by GStreamer, although our implementation focuses on H.264 video and MP3 audio. The NDNVideo publisher generates signed ContentObjects from an input video stream, which can be either live or pre-recorded, and places them immediately into a local repository—a persistent, disk-based storage that is another fundamental component in the NDN architecture. Consumers' data requests go to that repository or are otherwise satisfied by caches in the network.

NDNVideo handles large (video) and small (audio) samples by packing data in two layers. The inner layer contains all information necessary for the playback of a data buffer, such as timestamp, duration, and length in bytes. The timestamp tells the player at what point in time the buffer is supposed to be played, while the duration tells how long, making NDNVideo compatible with variable frame rate codecs. Then, multiple buffers are put inside one packet by trying to fill it completely. If a buffer does not fit into one packet, it is split into multiple packets. The outer layer of the packet contains two additional fields: offset and count. The count indicates how many buffers begin in a given packet, and the offset is used to tell where the first buffer starts. The offset is used only when there is a packet loss, and it lets the consumer quickly resume processing from the next available buffer.

Random access. A player seeks in NDNVideo by simply issuing an Interest in the index namespace corresponding to the desired timecode (e.g., "HH:MM:SS:FF"), with Interest "selector fields" set to return the nearest keyframe[1] ContentObject, which in turn maps timecode to segment number using a simple ASCII text payload. NDN's Interest selector fields enable a consumer to express requests that are more sophisticated than just name prefixes; they are described in detail in [5]. The *ChildSelector* is set to RIGHTMOST, instructing the responding node to return the last element it has in the index namespace, and *AnswerOriginKind* is set to NONE, to tell the

[1] The publisher indexes only keyframes, since the closest preceding keyframe is needed to properly decode a specific frame of the video.

library to fetch content from the network rather than any local cache. Finally, the *Exclusion Filter* is used to tell the network to return the latest index entry, just before the desired frame. (This requires timecode be expressed in a format that sorts properly according to the NDN architecture.) For example, to seek to 00:00:05:00, the Interest excludes all indexes after this point and, through the rightmost child selector, requests the next largest index. Thus, nodes on the network will respond with the nearest keyframe ContentObject they have, without any direct negotiation with the publisher.

5. Live Streaming

With NDN, the video publisher is far simpler than the corresponding one in IP; even for live streaming, it simply puts frames in a network-addressable content repository immediately after capture. Cooperation between publisher and consumer to maintain QoS (as done in RTP) is no longer needed. In the NDN case, the consumer knows exactly whether any data packet is lost, and there is no need to inform the publisher of how fast it needs to send the data, since it controls its own data fetch. However, some complexity is shifted to the stream consumer for certain cases, such as live streaming. The video consumer needs to pipeline Interest packets so that the data are fetched continuously as they are produced. However, it should not fetch the data too quickly and request segments that do not yet exist. To address this challenge, the player must determine what is the latest data and at which rate it should be requested.

When an NDN node receives an Interest, it does a longest-prefix match, also incorporating specified selectors, to see if it can be answered with data contained in its Content Store, only forwarding Interests to neighboring nodes when it cannot satisfy the request. Because different nodes can have different data in their Content Stores, the same request might result in different responses from different nodes. To accommodate this behavior, the consumer first issues Interests periodically during its video playback to determine the ever-increasing duration of the stream, by checking for the latest entry in the index namespace. It sets the exclusion filter in the Interest packet to only request Data packets with index name components *greater* than the last index it has seen. This forces the node providing the previous response to forward the Interest to its peers. Without this parameter, the consumer's Interest would retrieve previously received content cached in the network. Different nodes may respond with their own notion of what is "latest," so these Interests must be issued more than once. In the worst case, this approach may take N queries, where N is number of connecting nodes accessed, before converging on the correct "latest position."

Second, to enable the consumer to determine the rate at which it should issue Interests, in addition to the timestamp of the buffer, each individual packet also contains a local time at which the packet was generated. While the consumer may or may not have a clock in sync with the publisher, this information is still useful, because it can be used to calculate the time difference between packets. The consumer can then estimate the mean time interval and dynamically adjust the rate of Interests after starting with the latest segment of the live stream previously determined.

If the player does not receive data fast enough to play back at the correct rate, instead of pausing the playback, it skips to the most recent segment and continues playback from there. To do so, it calculates the local time at which given content is supposed to be played back, as well as determining whether to send the data to the decoder or not.

Compared to the complexities in live video stream seeking and pipelining, archival/recorded playback is more straightforward; the video's length is fixed. The only additional information needed for streaming from a pre-recorded file (or a live stream that is already completed) is a marker of the end of the stream. By convention, the publisher sets the value of the FinalBlockID field in the ContentObjects to the last segment number to signal the end of the stream. If the player uses multiple streams (e.g. audio and video), it stops playback after receiving the EOS (End of Stream) signal from all the streams.

6. Handling Packet Loss

Given the pull-based nature of NDN, NDNVideo lets the consumer be fully in charge of data it is receiving. Each data packet is named with the segment name plus segment numbers to make the data names predictable and enable the consumer to pipeline requests for the data. This is necessary to provide sufficient playback quality, especially when latency between the publisher and the consumer is high. If a single buffer is contained in multiple packets, the packet header information is used to put the buffer back together. In case of packet loss, the consumer can either request the same data again or issue an Interest for the next segment. If the segment is considered lost, the offset field is used to determine the point at which the next buffer starts. The code does not need to wait for the buffers that start at the beginning of packets (e.g., those of keyframes).

In an ideal case, the consumer will get responses to all the Interests it issues. Unfortunately, packets can be dropped due to network congestion or other causes. In order to provide seamless playback, it is important for the consumer to know how long to wait before assuming that an Interest or the corresponding data packet is lost. Interests for the data can then be quickly reissued or assumed unavailable, and the consumer can move on to the next segment. The NDNVideo

consumer adjusts its Interest timeout based on previous RTT values, smoothed using a low pass filter similar to what is defined in RFC 2988 for TCP.

7. Implementation and Testing

The implementation is written in Python, and uses the GStreamer [6] multimedia framework. It employs PARC's CCNx [4] implementation of the NDN architecture and our PyCCN [7] bindings. Both NDNVideo and PyCCN are open source (as are CCNx and GStreamer) and can be retrieved from GitHub.

After a variety of tests using pre-recorded files and live sources, the system was demonstrated "live" to a large audience in March 2012. A live, standard definition H.264-encoded stream (@ 1Mbit/sec) from a musical performance in the UCLA School of Theater, Film and Television's TV Studio #1 was published over the NDN testbed to our Washington University in St. Louis collaborators' demonstration for the GENI Engineering Conference in Los Angeles. Broadcast quality audio and video feeds from three cameras were mixed live and published to a CCN repo at UCLA. The WUSTL team displayed the video using the NDNVideo player. In this and other tests with standard definition, H.264 video, the streaming works well for end-users. We have recently added support for high-definition ("1080p") resolution. Additionally, we have deployed webcams connected to application servers at several geographic locations on the NDN testbed, which use NDNVideo.

8. Current issues and future work

The interval-based pipeline is being refined for deployment in the next series of demonstrations and tests. Additionally, the CCNx repository does not yet support deletion of specific data objects, which can be problematic for long-running live streams; this will be addressed in future versions.

Finally, we plan to enable the consumer to switch codec based on bandwidth or performance. For example, when the consumer detects that it cannot receive data at the desired rate, it could downgrade playback to a lower bit rate by simply changing a component of the Interest name it is requesting. An elegant way to do this would be to provide H.264 Scalable Video Coding (SVC) or a similar solution with enhancement layers expressed directly in the video namespace. We are exploring this solution.

9. Conclusion

We designed, implemented, and tested NDNVideo, a video streaming application that was conceived with NDN architecture in mind and demonstrates some immediate advantages of NDN. The protocol and namespace are equivalent for live and pre-recorded streams, requiring only additional logic at the consumer for live streaming. Reliable and rate-adaptive playback can be provided with no session semantics or negotiation necessary between the consumer and the producer. The approach leverages Content Stores in the network, which makes the video distribution more efficient. Video and audio streaming uses the intrinsic features of the architecture to scale without loading the publisher, and to provide efficient random-access, even to live streams. We believe that NDN-based streaming can enable a better user experience with less strain on the publisher when compared to TCP/IP, and that the approach could be used to enable serverless video publishing from resource-constrained mobile devices.

References

[1] Zhang, L., *et al*., "Named data networking (ndn) project," http://named-data.net/techreport/TR001ndn-proj.pdf, 2010.

[2] "CCNx vlc plugin," part of the CCNx package. [Online]. Available: https://github.com/ProjectCCNx/

[3] Letourneau, J. "CCNxGST - GStreamer plugin used to transport media traffic over a CCNx network." [Online]. Available: https://github.com/johnlet/gstreamer-ccnx

[4] "CCNx." [Online]. Available: http://www.ccnx.org

[5] CCNx Technical documentation. [Online]. http://www.ccnx.org/releases/latest/doc/index.html

[6] "GStreamer - open source multimedia framework." [Online]. Available: http://gstreamer.freedesktop.org

[7] "PyCCN - python bindings for CCNx." [Online]. Available: https://github.com/remap/PyCCN

[8] Xu, H., et al. "Live Streaming with Content Centric Networking." *IEEE 3rd Intl. Conf. on Networking and Distributed Computing (ICNDC), 2012*.

[9] Detti, A., et al. "Offloading cellular networks with Information-Centric Networking: The case of video streaming," *IEEE WOMOM 2012*, June, 2012.

[10] Han, B. et al. "AMVS-NDN: Adaptive Mobile Video Streaming and Sharing in Wireless Named Data Networking," *IEEE NOMEN 2013*, April 19, 2013.

Derek Kulinski is a Systems Engineer at Edmunds.com. He received his MS in Computer Science from UCLA, where he participated in research at the Center for Embedded Networked Sensing and on Named Data Networking.

Jeff Burke is the Director of Technology Research Initiatives for the UCLA School of Theater, Film and Television and the Executive Director of the UCLA Center for Research in Engineering, Media and Performance. Jeff is the application team lead for NDN.

Lixia Zhang is a professor in the UCLA Computer Science Department. She previously served as vice chair of ACM SIGCOMM, member of the editorial board for the IEEE/ACM Transactions on Networking, member of the Internet Architecture Board, and co-chair of the Routing Research Group under IRTF. She is a fellow of ACM and IEEE, and holds the UCLA Postel Chair in Computer Science.