NDN, Technical Report NDN-0031, 2015. http://named-data.net/techreports.html
Revision 1: April 30, 2015

# NDNlive and NDNtube: Live and Prerecorded Video Streaming over NDN

Lijing Wang
Tsinghua University
wanglj11@mails.tsinghua.edu.cn

Ilya Moiseenko
UCLA
iliamo@cs.ucla.edu

Lixia Zhang
UCLA
lixia@cs.ucla.edu

## ABSTRACT

Named Data Networking offers significant advantages for media distribution applications: in-network caching and multicast capabilities. This technical report provides a detailed view on two video streaming applications: NDNlive and NDNtube. NDNlive is capable of streaming live video captured by the camera and handling network problems by dropping individual video or audio frames. NDNtube prototypes Youtube-like user experience by serving dynamically generated playlist and streaming the media in its original quality. Both applications were developed on top of Consumer / Producer API providing a convenient way of publishing and fetching ADUs of any size, and Gstreamer library providing media decoding/encoding functionality.

## 1. INTRODUCTION

The Named Data Networking (NDN) was proposed as a new Internet architecture that aims to overcome the weaknesses of the current host-based communication architecture in order to naturally accommodate emerging patterns of communication [1, 2, 3]. By naming data instead of their locations, NDN transforms data into a first class entity, which offers significant promise for content distribution applications, such as video playback application. NDN reduces network traffic by enabling routers to cache data packets. If multiple users request the same video file, the router can forward the same packet to them instead of requiring the video publisher to generate a new packet every time.

NDN applications work with Application Data Units (ADU) — units of information represented in a most suitable form for each given use-case (e.g. Application Level Framing [4]). For example, a multi-user game's ADUs are objects representing current user's status; for an intelligent home application, ADUs represent current sensor readings; and for a video playback application, data is typically handled in the unit of video frames.

NDN publisher applications publishes ADUs as a series of Data packets according to the design of the namespace. NDN consumer applications send Interest packets carrying application level names to request ADUs, and the network returns the requested Data packets following the path of the Interests.

Some of the existing techniques, such as MPEG-DASH partly adopt the principles of Application Level Framing by breaking multiplexed or unmultiplexed content into a sequence of small file segments of equal time duration [5]. File segments are later served over HTTP from the origin media servers or intermediate HTTP caching servers.

Since the file segments have a bigger granularity than individual video and audio frames, the user of MPEG-DASH video streaming application still experiences interruptions and abrupt changes in the media quality.

In this technical report, we talk about two video streaming NDN applications: NDNlive and NDNtube. NDNlive is capable of streaming live video captured by the camera and handling network problems by dropping individual video or audio frames. NDNtube prototypes the user experience of Youtube by providing dynamically generated playlist and streaming the media in its original quality (no frames are dropped). Both applications were developed on top of Consumer / Producer API [6] providing a convenient way of publishing and fetching ADUs of any size, and Gstreamer [7] library providing media decoding/encoding functionality (Section 2). Architecture and implementation details of both applications are described in Section 4 and 5. The prior work is discussed in Section 6, followed by the conclusion in Section 7.

## 2. BACKGROUND

In this section, we talk about how NDNlive and NDNtube make use of external NDN and media processing libraries and other major NDN components.

### 2.1 Consumer / Producer API

Consumer-Producer API [6] provides a generic programming interface to NDN communication protocols and architectural modules. A consumer context associates an NDN name prefix with various data fetching, transmission, and content verification parameters, and integrates processing of Interest and Data packets on the consumer side. A producer context associates an NDN name prefix with various packet framing, caching, content-based security, and namespace registration parameters, and integrates processing of Interest and Data packets on the producer side.

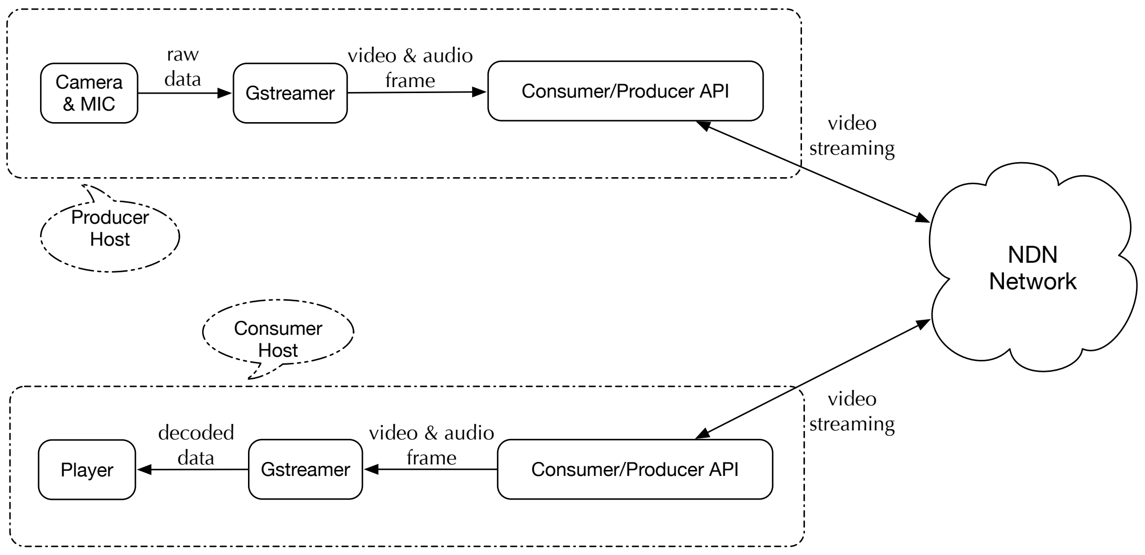

**Figure 1: NDNlive Architecture**

In NDNlive and NDNtube, the video publisher consists of multiple producers generating video and audio frames separately. The corresponding video players consist of multiple consumers sending Interests for the video and audio frames. Consumer / Producer API simplifies the application logic at both sides: media production and media consumption. Since video frames are too large to be encapsulated by a single Data packet, the media production pipeline has to include a content segmentation step in order to split the content into multiple Data packets. Producer API provides this segmentation functionality. At the same time, since a video frame cannot be retrieved by a single Interest packet, UDR and RDR protocols behind the Consumer API automatically pipeline Interest packets and solve other tasks related to the retrieval of the application frame. We will talk about the implementation details in Section 5.

## 2.2 Gstreamer

We use Gstreamer [7] to handle the media processing part.

In NDNlive, raw video images captured by the camera are transferred to the *Encoder_v* component and are encoded into *H264* format. Then the encoded video is passed to the *Parser_v* to be parsed into frames (*B, P or I frame*). The microphone captures the raw audio, which is passed to the *Encoder_a*. The encoder component encodes the raw audio into *AAC* format. The encoded audio stream is transferred to the *Parser_a* to be parsed into audio frames, which are passed to the Producer API for any possible segmentation. Video and audio data is retrieved frame by frame that are passed to the video *Decoder_v* and audio *Decoder_a* for the decoding into the format which the video *Player_v* or audio *Player_a* can play.

In NDNtube, the source for video and audio streams is an mp4 file containing *H264* video and *AAC* audio. Gstreamer opens the file and passes it to the *Demuxer* component to separate video and audio streams. Since the video file is already encoded, the media processing pipeline does not have an *Encoder* component in it. The encoded video or audio streams are separately pushed into the *Parser* to generate video and audio frames.

## 2.3 Repo

NDNlive streams the captured video and audio non-stop. Therefore, the media publisher just keeps producing new frames and does not care about the data it produced several minutes ago. The consumer is also interested only in recent video and audio frames. As long as the producer is attached to the NDN network, it will serve the incoming Interests.

NDNtube publishes the video only once — all Data packets corresponding to audio and video frames are permanent and never change after the initial publication. Since the same video could be requested multiple times by different users, it is reasonable to store the produced Data packets in a ‘database’ which is exposed to the requests from the network. Otherwise, every time a different user requests the same video, the corresponding video and audio Data packets would have to be republished (and signed) in case NDN cache has not been able to satisfy these Interests.

Repo-ng [8] is used as a permanent storage for the video and audio content. Repo-ng (repo-new generation) is an implementation of NDN persistent in-network storage, which exposes a Repo protocol [9] allowing write access to applications. Repo insertion is natively supported by the Producer API with *LOCAL_REPO* option (if repo is running on the local host) or *REMOTE_REPO_PREFIX* option to point to the right remote repo by its name prefix.

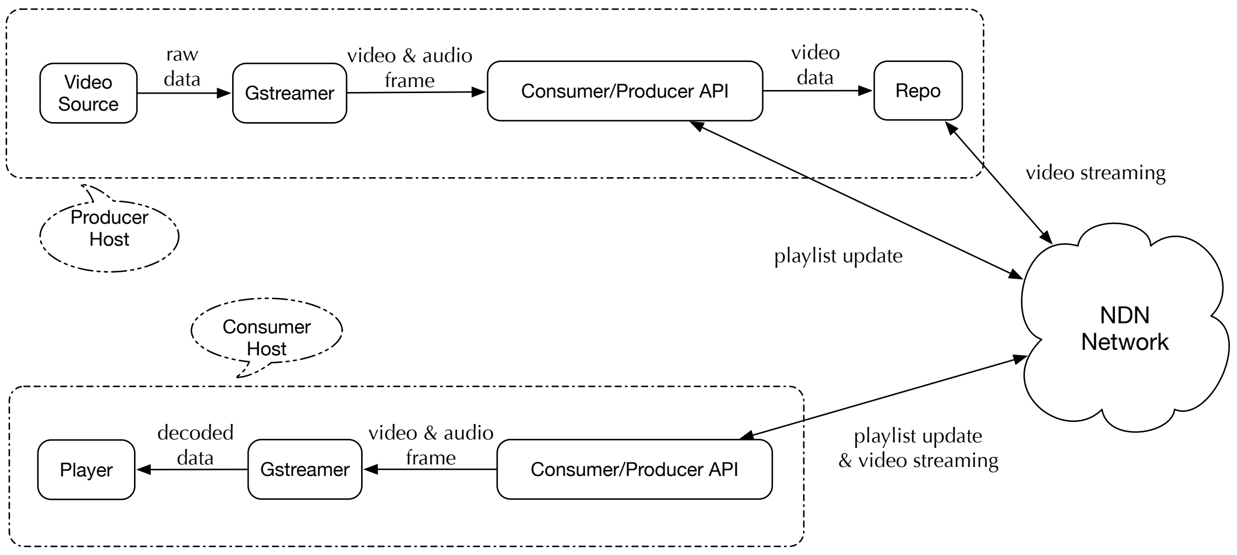

**Figure 2: NDNtube Architecture**

## 3. DESIGN GOALS

UCLA REMAP lab put efforts in building NDNVideo [10] that was one of the first showcases in support of the claim of the possibility of video streaming over NDN. The later changes in the packet format and the development of the new NDN forwarding daemon (NFD [11]) made this implementation of video streaming software obsolete.

NDNlive and NDNtube is the new effort of UCLA IRL lab to have similar functionality to NDNvideo compatible with the new packet format, new forwarder and new application libraries, such as Consumer / Producer API. The secondary goal of developing these application is to provide more complete examples for application developers learning the new Consumer / Producer API.

The following list contains the key features of NDNlive and NDNtube applications:

- *"Live and pre-recorded video&audio streaming to multiple users"*

  NDNlive provides the live video&audio streaming to multiple users and guarantees the fluency of the streaming. NDNtube prototypes a Youtube-like user experience: 1) selection of a media from the list of available media resources, and 2) smooth buffered playback.

- *"Random access based on actual location in the video"*

  The video stream is organized as two series of media frames: video and audio. The relationship between playback time and the frame number is non-ambiguous, because both applications use constant frame-rate encoding. In other words, the frame number can be easily calculated from the playback time information and the video and audio frame rate.

- *"Synchronized playback"*

  At the moment of data production, each frame is extracted in the form of *GstBuffer* [12], which contains playback timestamp information. When audio and video frames are retrieved at the consumer side, video and audio streams are synchronized by Gstreamer library automatically. Another way to achieve synchronization without Gstreamer library is to consider the direct relationship between frame-rate, playback time, and frame numbers.

- *"Connection-less and session-less streaming"*

  NDNlive and NDNtube consumers do not try to establish any persistent session or connection with the video publisher. Video and audio frames are fetched from either from in-network cache, or producer's cache, or the permanent storage (e.g. Repo), when producer goes offline.

- *"Content verification and provenance"*

  Every Data packet must be signed with an asymmetric key of the original publisher in order to reliable authenticate content. Video publishing pipeline outputs so many Data packets that the security component of ndn-cxx library becomes a bottleneck, because of its limited signing speed. In order to achieve the desired signing throughput, both NDNlive and NDNtube producers use Producer API with FAST_SIGNING option.

## 4. DESIGN

NDNlive and NDNtube consist of two big components: publisher and consumer (player). Both publisher and player make use of appropriate parts of Consumer / Producer API, which significantly simplifies the design of the system.

*NDNlive architecture.*

Figure 1 illustrates the architecture of NDNlive, which can be summarized as follows:

- *Publisher*

  NDNlive is a *live streaming* application; the publisher captures video from camera and audio from microphone and passes it to the Gstreamer to encode the raw data and extract individual video and audio frames. The video and audio frames are published to NDN network with Consumer / Producer API.

- *Player*

  The video player uses the UDR protocol of Consumer / Producer API protocol suite to generate Interest packets for specific video and audio frames, which are later passed to the Gstreamer for decoding purposes. The player application is responsible for timing the consumption of individual frames, i.e. pacing of *consume()* calls.

*NDNtube architecture.*

Figure 2 illustrates the architecture of NDNtube, which can be summarized as follows:

- *Publisher*

  NDNtube is a *pre-recorded media streaming* application, therefore the publisher works with existing video files stored on the disk. As we described in Section 2.3, the publisher reads the file from the disk, extracts video and audio frames from it and publishes these frames to the Repo. After that, the Repo takes over the duty of responding to the Interests requesting the frames.

- *Player*

  Comparing to NDNlive, NDNtube player has an additional functionality for displaying the list of the currently available video resources (e.g. playlist). In order to support this feature, NDNtube publisher keeps publishing the updated playlist every time a new video is added to the collection.

## 4.1 Namespace design

NDNlive and NDNtube separate video and audio streams from each other. Each stream consists of multiple frames and every single frame consists of multiple segments carrying unique names. The following examples provide the details of the naming schemes used in NDNlive and NDNtube applications.

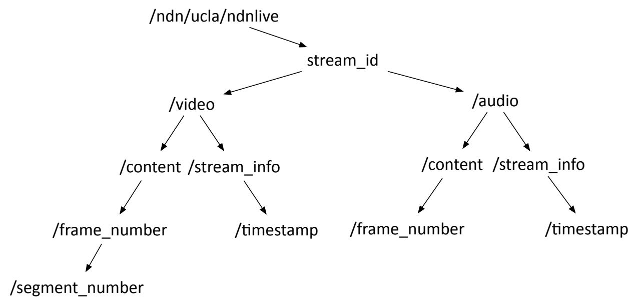

**Figure 3: NDNlive namespace**

*NDNlive namespace.*

The following name is a typical name of the Data segment that corresponds to a video frame.

"/ndn/ucla/NDNlive/stream-1/video/content/8/%00%00"

- **Routable Prefix:** "/ndn/ucla/NDNlive" is the routable prefix used by NFD forwarders to direct Interest packets towards NDNlive publisher.
- **Stream_ID:** "/stream-1" is a stream identifier used distinguish among live streams. Note, that stream ID could be a part of the routable prefix.
- **Video or Audio:** "/video" is a markup component to distinguish between video and audio streams.
- **Content or Stream_Info:** All frames go under "/content" prefix, and all stream information go under "/stream_info" prefix.
- **Frame number:** "/8" is frame number used to identify each individual video and audio frame.
- **Segment number:** "%00%00" is the segment number required to identify each individual Data segment, because most video frames are too large to fit in a single Data packet, and have to be broken into segments.

The following name is a typical name of the Data packet carrying the auxiliary information about the stream.

"/ndn/ucla/NDNlive/stream-1/video/stream_info/1428725107049"

Since remote participants join the live stream at different time and are not interested in watching the stream from its beginning, they must acquire the knowledge about the current frame numbers of the video and audio streams. At this preliminary step, the consumer requests the stream information object containing the current frame number of video and audio streams as well as other media encoding information. This information is kept up to date by the video publisher, which continuously publishes the new versions of this object. Each new version has a unique name with a different timestamp component in it (Figure 3).

*NDNtube namespace.*

NDNtube's namespace is mostly similar to NDNlive, with the following four differences:

1. *Playlist namespace branch*

   The user of NDNtube can select any video from the list of available ones (e.g. playlist). The typical name of the playlist is shown below.

   "/ndn/ucla/NDNtube/playlist/1428725107042"

   The playlist is identified by the timestamp name component, because it is updated every time the new file is added or the old file is removed from the collection of media resources. The consumer is interested in the latests version of it (e.g. rightmost).

2. *Video name*

   Video name must match one of the video names provided by the playlist. Semantically, the video name component replaces the stream ID component of the NDNlive application.

3. *Permanent stream information*

   In NDNtube, the information object carrying auxiliary video encoding information (e.g. final frame number, width, height, etc.) is published only once and is not updated after that, unlike the stream information in NDNlive's live streams. As a result, the name of the information object does not contain a timestamp component.

4. *Multi-segment audio frames*

   Since some mp4 video files that are added to the collection of media resources contain a high quality audio stream, the audio frames have to be broken into Data segments that have unique segment name component (Figure 4).

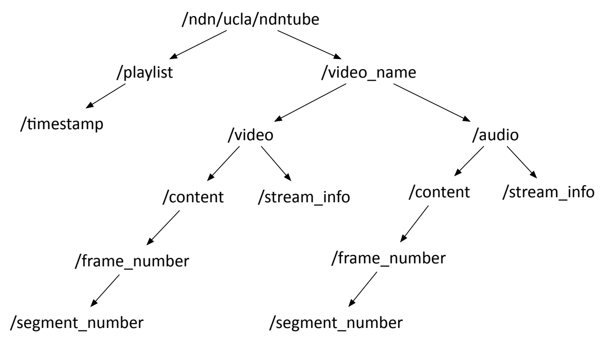

**Figure 4: NDNtube namespace**

## 5. IMPLEMENTATION

NDNlive and NDNtube are developed using Consumer / Producer API and Gstreamer 1.4.3 library.[1] The supported platforms are Mac OS X and Linux Ubuntu.[2]

### 5.1 NDNlive

As shown in Figure 1, NDNlive consists of two applications — one is running at the publisher's host and another one at the consumer's host. In this section, we go over implementation details of the publisher's application and then continue with consumer's application.

#### 5.1.1 Publisher

Publisher's application has four producers: video content producer, video stream information producer, audio content producer and audio stream information producer. Figure 5 shows the locations of the producers in the NDNlive namespace.

Two content_producers continuously publish video and audio frames by incrementally increasing the correpsonding frame numbers (Figure 3).

Two stream_info producers continuously publish up-to-date information about the live streaming media: current frame number, frame rate, video width and height, encoding format (Figure 3).

*Negative Acknowledgement.*

In some situations, the live stream publisher is not able to satisfy Interests with actual data (e.g. video and audio frames).

1. The consumer may sometimes miscalculate the pacing of video and audio frames and request the frame that does not exist at the moment (e.g. ahead of the production). The publisher can inform the consumer about this situation using *nack()* function of the Consumer / Producer API.

   As illustrated by the Algorithm 1, producer calls *nack()* function with *PRODUCER_*
   *DELAY* header containing the anticipated time value after which the data may become available.

2. Consumers join the live stream after the publisher. Since the publisher of the live stream stores only a limited number of the most recently produced audio and video frames, some consumers might request the frames that has already expired everywhere in the network. In this case, the publisher calls the *nack()* operation with textitNO-DATA header, which informs consumers that a particular video or audio frame is no longer available.

The pseudocode of NDNlive publisher application is provided in Algorithm 1.

[1]Other versions of Gstreamer may not be compatible

[2]Other Linux platforms are potentially supported

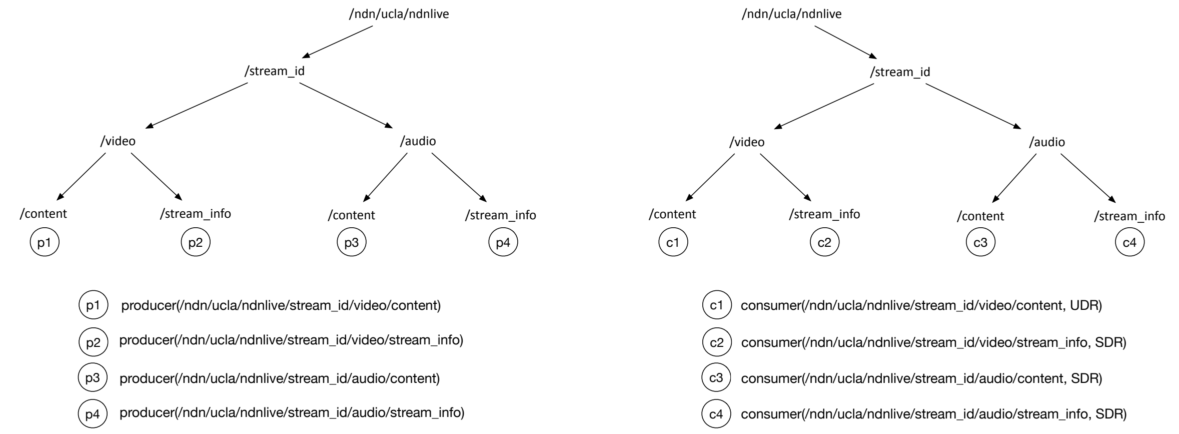
/ndn/ucla/ndnlive
/stream_id
/video
/audio
/content
/stream_info
p1
p2
p3
p4
p1 producer(/ndn/ucla/ndnlive/stream_id/video/content)
p2 producer(/ndn/ucla/ndnlive/stream_id/video/stream_info)
p3 producer(/ndn/ucla/ndnlive/stream_id/audio/content)
p4 producer(/ndn/ucla/ndnlive/stream_id/audio/stream_info)
c1
c2
c3
c4
c1 consumer(/ndn/ucla/ndnlive/stream_id/video/content, UDR)
c2 consumer(/ndn/ucla/ndnlive/stream_id/video/stream_info, SDR)
c3 consumer(/ndn/ucla/ndnlive/stream_id/audio/content, SDR)
c4 consumer(/ndn/ucla/ndnlive/stream_id/audio/stream_info, SDR)

**Figure 5: Locations of producers and consumers in the NDNlive namespace.**

### *5.1.2 Player*

NDNlive consumer must fetch the live stream information to set up the Gstreamer playing pipeline before it can request any of the audio or video frames. The application has four consumers: video content consumer, video stream information consumer, audio content consumer and audio stream information consumer.

#### *Data retrieval.*

Consumer / Producer API protocol suite offers three data retrieval protocols: SDR, UDR, RDR. In this section we describe how NDNlive consumer application uses SDR and UDR protocols. The pseudocode of the NDNlive consumer application is provided in the Algorithm 2.

1. *Content Retrieval*

   In the case of the live media streaming, the consumer application must continue retrieving video and audio frames at all times in order to keep up with the data production rate. All segments of each frame must be retrieved as fast as possible and the fetching process should not block other frames because of segment losses.

   NDNlive video content consumer uses **UDR** (*Unreliable Data Retrieval*) protocol for video frame retrieval. Since **UDR** pipelines Interests transmission and does not provide ordering, some Data segments may arrive out of order. NDNlive consumer application takes care of Data segment reassembly and drops the whole frame is any of its segments are lost.

   NDNlive audio content consumer uses **SDR** (*Simple Data Retrieval*) for audio frame retrieval. UDR does not pipeline Interest packets, which satisfies our requirements, since the audio frame is small enough to fit in just one Data segment.

2. *Stream Information Retrieval*

   Stream information is periodically updated by the video publisher, which essentially means creation of a new Data packet with a unique name (e.g. new timestamp name component). The consumer that is trying to join the live stream does not know the unique name of the latest stream information object, and therefore cannot use UDR or RDR protocols which assume such knowledge. A simple solution of this problem is to use **SDR** (*Simple Data Retrieval*) protocol with *Right_Most_Child* option set as TRUE. The protocol generates a single Interest packet with *RightmostChildSelector* which is capable of fetching the latest stream info object.

#### *Frame-to-frame interval.*

Consumer application should control the Interest sending speed. If it sends Interests too aggressively and the data is not yet produced by the publisher application, the playback may collapse. If it sends Interests too slowly, the playback may fall behind the video generation. NDNlive uses constant frame rate encoding, therefore for a video, which is encoded by 30 frames per second, the interval between frames is $1000/30 \approx 33.3$ millisecond. In other words, every 33 milliseconds the application calls *consume()* that attempts to fetch all segments of the frame as quickly as possible.

#### *Synchronization of video and audio.*

Since NDNlive is streaming video and audio separately, it is a vital problem to keep these streams synced. When video and audio frames are captured, they are timestamped by the Gstreamer. The time information is recorded in *GstBuffer* data structure containing the media data, and transferred along with every video or audio frame. When the consumer fetches the video or audio frames separately, the video and audio frames are pushed into the same *GstQueue*. Gstreamer extracts the timestamps present in the video and audio frames, and displays the content in synchronized mode.

**Algorithm 1** NDNlive producer

```
$h_v \leftarrow$ **producer**(/ndn/ucla/NDNlive/stream-1/video/
 content)
**setcontextopt**($h_v$, **cache_miss**, *ProcessInterest*)
**attach**($h_v$)
**while** *TRUE* **do**
    $Name\ suffix_v \leftarrow$ video frame number
    $content_v \leftarrow$ video frame captured from camera
    **produce**($h_v$, $Name\ suffix_v$, $content_v$)
**end while**

$h_a \leftarrow$ **producer**(/ndn/ucla/NDNlive/stream-1/audio/
 content)
**setcontextopt**($h_a$, **cache_miss**, *ProcessInterest*)
**attach**($h_a$)
**while** *TRUE* **do**
    $Name\ suffix_a \leftarrow$ audio frame number
    $content_a \leftarrow$ audio frame captured from mirophone
    **produce**($h_a$, $Name\ suffix_a$, $content_a$)
**end while**

**function** PROCESSINTEREST(Producer **h**, Interest **i**)
    **if** *NOT Ready* **then**
        $appNack \leftarrow$ **AppNack**($i$, **RETRY-AFTER**)
        **setdelay**($appNack$, $estimated\_time$)
        **nack**($h$, $appNack$)
    **end if**
    **if** *Out of Date* **then**
        $appNack \leftarrow$ **AppNack**($i$, **NO-DATA**)
        **nack**($h$, $appNack$)
    **end if**
**end function**
```

**Algorithm 2** NDNlive consumer

```
$h_v \leftarrow$ **consumer**(/ndn/ucla/NDNlive//stream-1/video/
 content, *UDR*)
**setcontextopt**($h_v$, **new_segment**, *ReassambleVideo*)

**while** *reaching Video_Interval* **do**
    $Name\ suffix_v \leftarrow$ video frame number
    **consume**($h_v$, $Name\ suffix_v$)
    $framenumber++$
**end while**
**function** REASSEMBLEVIDEO(Data **segment**)
    $content \leftarrow$ reassemble **segment**
    **if** *Final_Segment* **then**
        $video \leftarrow$ decode **content**
        Play $video$
    **end if**
**end function**

$h_a \leftarrow$ **consumer**(/ndn/ucla/NDNlive/stream-1/audio/
 content, *SDR*)
**setcontextopt**($h_a$, **new_content**, *ProcessAudio*)
**while** *reaching Audio_Interval* **do**
    $Name\ suffix_a \leftarrow$ audio frame number
    **consume**($h_a$, $Name\ suffix_a$)
    $framenumber++$
**end while**
**function** REASSEMBLEAUDIO(Data **content**)
    $audio \leftarrow$ decode **content**
    Play $audio$
**end function**
```

## 5.2 NDNtube

Although the namespace of NDNtube might look very similar to the namespace of NDNlive, the patterns of the data production and retrieval are quite different.

### *5.2.1 Publisher*

Publisher's application has three producers: dynamic playlist producer, video content producer and audio content producer. Figure 6 shows the locations of the producers in the NDNtube namespace.

Playlist producer $P_1$ is responsible for generating the latest playlist every time a video file is added or removed from the collection of media resources. Producer $P_1$ runs as long as the whole publisher application.

Video content producer $P_2$ is responsible for publishing video frames and the stream information object for a each particular media resource. Since producer $P_2$ is configured with *LOCAL_REPO* option, all packets are written to the repo running on the same local host. After all video frames as well as stream information object are successfully inserted in the repo, producer $P_2$ terminates its execution.[3]

Audio content producer $P_3$ is responsible for publishing audio frames and the stream information object for a each particular media resource. Since producer $P_3$ is configured with *LOCAL_REPO* option, all packets are written to the repo running on the same local host. After all audio frames as well as stream information object are successfully inserted in the repo, producer $P_3$ terminates its execution (Preudocode 3).

### *5.2.2 Player*

The application has five consumers: playlist consumer $C_1$, video content consumer $C_2$, video stream information consumer $C_3$, audio content consumer $C_4$ and audio stream information consumer $C_5$. Figure 6 shows the locations of the consumers in the NDNtube namespace.

*Data retrieval.*

In this section we describe how NDNtube consumer application uses SDR and RDR protocols. The pseudocode of the NDNtube consumer application is provided in the Algorithm 4.

[3]Publisher process continues to run.

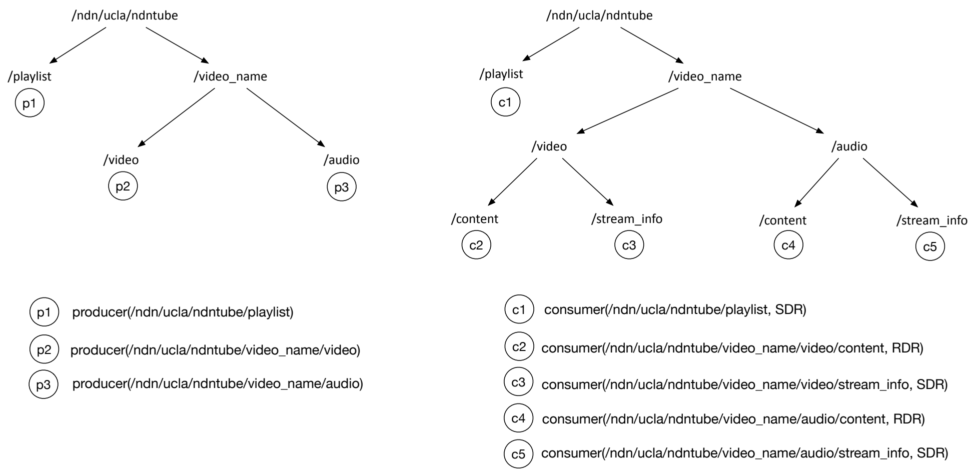
/ndn/ucla/ndntube
/playlist
p1
/video_name
/video
p2
/audio
p3
/ndn/ucla/ndntube
/playlist
c1
/video_name
/video
/audio
/content
c2
/stream_info
c3
/content
c4
/stream_info
c5
p1 producer(/ndn/ucla/ndntube/playlist)
p2 producer(/ndn/ucla/ndntube/video_name/video)
p3 producer(/ndn/ucla/ndntube/video_name/audio)
c1 consumer(/ndn/ucla/ndntube/playlist, SDR)
c2 consumer(/ndn/ucla/ndntube/video_name/video/content, RDR)
c3 consumer(/ndn/ucla/ndntube/video_name/video/stream_info, SDR)
c4 consumer(/ndn/ucla/ndntube/video_name/audio/content, RDR)
c5 consumer(/ndn/ucla/ndntube/video_name/audio/stream_info, SDR)

**Figure 6: Locations of producers and consumers in the NDNtube namespace.**

**Algorithm 3** NDNtube publisher

```
$h_v \leftarrow$ producer(/ndn/ucla/NDNtube/video-1234/video)
setcontextopt($h_v$, local_repo, TRUE)
while NOT Final_Frame do
    $Name\ suffix_v \leftarrow$ video frame number
    $content_v \leftarrow$ video frame
    produce($h_v$, $Name\ suffix_v$, $content_v$)
end while

$h_a \leftarrow$ producer(/ndn/ucla/NDNtube/video-1234/audio)
setcontextopt($h_a$, local_repo, TRUE)
while NOT Final_Frame do
    $Name\ suffix_a \leftarrow$ audio frame number
    $content_a \leftarrow$ audio frame
    produce($h_a$, $Name\ suffix_a$, $content_a$)
end while
```

1. *Content Retrieval* All video and audio frames as well as stream information objects are retrieved by **RDR** (*Reliable Data Retrieval*) protocol, which provides ordered and reliable fetching of Data segments. NDNtube video player does not consume a live streaming media, and consequently can afford much larger buffering delays in order to preserve the original quality of the video and audio resources. By default, NDNtube buffers for at least two seconds of video and audio frames of real playback time before it begins (or resumes) its playback. Buffering allows to soften the delays of frame retrieval due to possible Interest retransmissions done by the RDR protocol.

   An expected but nevertheless interesting effect of frame-by-frame reliable delivery shows itself in rare cases when a particular video or audio frame cannot be retrieved within a reasonable amount of time (e.g. Interest retransmissions) and application faces the choice whether it wants to skip the frame or try to consume() it again. Since our goal was to prototype a Youtube-like user experience, in this situation, NDNtube consumer will try to retrieve the same frame again.

2. *Playlist Retrieval*

   Playlist is periodically updated by the video publisher, which essentially means creation of a new Data packet with a unique name (e.g. new timestamp name component). The consumer that is trying to obtain the names of available media resources does not know the unique name of the latest playlist, and therefore cannot use UDR or RDR protocols which assume such knowledge. A simple solution of this problem is to use **SDR** (*Simple Data Retrieval*) protocol with *Right_Most_Child* option set as TRUE. The protocol generates a single Interest packet with *RightmostChildSelector* which is capable of fetching the latest playlist.

*Frame-to-frame interval.*

Since all the content and stream information already exists in the Repo for a long time, consumer can be quite aggressive with fetching video and audio frames. By default, NDNtube player starts fetching (via consume()) the next frame right after the current was successfully retrieved, which corresponds to the frame-to-frame interval of 0 milliseconds. Having below 0 ms. frame-to-frame interval is also a reality, because it is possible to fetch multiple frames in parallel.

The NDNtube consumer's Pseudocode is shown as Algorithm 4.

## 6. PRIOR WORK

NDNVideo provides live and pre-recorded video streaming over NDN using Gstreamer library for media processing

**Algorithm 4** NDNtube consumer

```
    $h_v \leftarrow$ **consumer**(/ndn/ucla/NDNtube/video-1234/
     video, *RDR*)
    **setcontextopt**($h_v$, **new_content**, *ProcessVideo*)
 4: **while** *NOT Final_Frame* **do**
        $Name\ suffix_v \leftarrow$ video frame number
        **consume**($h_v$, $Name\ suffix_v$)
        $framenumber++$
 8: **end while**
    **function** PROCESSVIDEO(byte[] **content**)
        $video \leftarrow$ decode **content**
        Play $video$
12: **end function**
    $h_a \leftarrow$ **consumer**(ndn/ucla/NDNtube/video-1234/
     audio, *RDR*)
    **setcontextopt**($h_a$, **new_content**, *ProcessAudio*)
16: **while** *NOT Final_Frame* **do**
        $Name\ suffix_a \leftarrow$ audio frame number
        **consume**($h_a$, $Name\ suffix_a$)
        $framenumber++$
20: **end while**
    **function** PROCESSAUDIO(byte[] **content**)
        $audio \leftarrow$ decode **content**
        Play $audio$
24: **end function**
```

and Repo for permanent storage [10]. Since the new NFD does not support the previous packet format, this implementation of video streaming is now obsolete.

The major difference between NDNlive / NDNtube and NDNVideo is the organization of the video and audio content. In the NDNVideo, video or audio streams are chopped into fixed-size segments, which requires a special mapping between real playback time and segment numbers to keep video and audio synced, and support "seek" functionality (Figure 7). Fixed-size segmentation breaks the boundaries of frames (e.g. Application Data Units), which causes the interdependency between different frames leading to the problems inherent to TCP/IP such as Head-Of-Line (HOL) blocking.

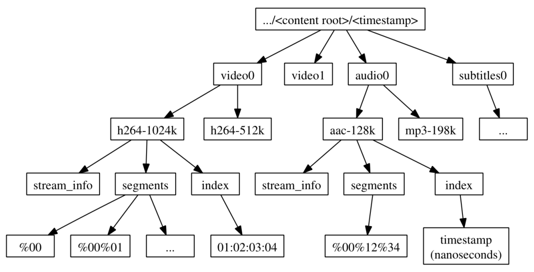

**Figure 7: Prior NDNVideo Naming Space**

In NDNlive and NDNtube, video and audio streams are chopped into frames. One frame may contain several segments. Since segmentation is provided by Consumer / Producer API, it is transparent for the application, which has to focus only at frame level. Since every frame has a unique name and is independent from any other frames, any missing frames do not affect other frames, which is highly beneficial for video streaming applications, especially for live streaming applications. For example, in NDNlive, if the previous frame can't be retrieved on time, the consumer can just skips it and continues with the next frame to keep the video streaming.

Table 1 illustrates other significant differences such as library dependencies and versions, and programming languages.

| | NDNlive & NDNtube | NDNVideo |
|---|---|---|
| Dependencies | ndn-cxx / NFD<br>Consumer / Producer API | CCNx / CCNR<br>pyccn |
| Gstreamer | 1.x | 0.1 |
| Framing | video & audio frames | fixed segments |
| Language | c++ | python |

**Table 1: Comparison with NDNVideo**

## 7. CONCLUSION

This technical report provides a detailed view on two video streaming applications: NDNlive and NDNtube. NDNlive is capable of streaming live video captured with the camera, and NDNtube is capable of streaming video from the mp4 encoded video files. Both applications organize video and audio streams as sequences of frames, which results in a big amount of flexibility in decision making process at the consumer side of the application. Due to the increased flexibility, NDNlive consumer is able to drop the frames which have not been reassembled by the playback deadline in order to keep up with the live stream.

NDNlive and NDNtube have different requirements for the user experience and, therefore, use different data retrieval protocols — UDR and RDR, respectively. Both applications serve as good and relatively complete examples of using Consumer / Producer API for people who are interested in developing applications with the current implementation of the library, as well as Consumer / Producer model in general.

## 8. REFERENCES

[1] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking Named Content," in *Proc. of CoNEXT*, 2009.

[2] L. Zhang et al., "Named Data Networking (NDN) Project," Tech. Rep. NDN-0001, October 2010.

[3] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, K. Claffy, P. Crowley, C. Papadopoulos, L. Wang, and B. Zhang, "Named Data Networking," *ACM*

*SIGCOMM Computer Communication Review*, July 2014.

[4] D. D. Clark and D. L. Tennenhouse, "Architectural Considerations for a New Generation of Protocols," *SIGCOMM Comput. Commun. Rev.*, vol. 20, no. 4, pp. 200–208, Aug. 1990.

[5] T. Stockhammer, "Dynamic adaptive streaming over HTTP: standards and design principles," in *Proceedings of the second annual ACM conference on Multimedia systems*. ACM, 2011, pp. 133–144.

[6] I. Moiseenko and L. Zhang, "Consumer-producer api consumer-producer api for named data networking," Technical Report NDN-0017, NDN, Tech. Rep. NDN-0017, August 2014.

[7] [Online]. Available: http://gstreamer.freedesktop.org/

[8] S. Chen, W. Shi, J. Cao, A. Afanasyev, and L. Zhang, "NDN Repo: An NDN Persistent Storage Model," 2014. [Online]. Available: http://learn.tsinghua.edu.cn:8080/2011011088/WeiqiShi/content/NDNRepo.pdf

[9] [Online]. Available: http://redmine.named-data.net/projects/repo-ng/wiki/Repo_Protocol_Specification

[10] D. Kulinski and J. Burke, "NDN Video: Live and Prerecorded Streaming over NDN," UCLA, Tech. Rep., 2012.

[11] A. Afanasyev, J. Shi, B. Zhang, L. Zhang, I. Moiseenko, Y. Yu, W. Shang, Y. Huang, J. P. Abraham, S. DiBenedetto *et al.*, "NFD developer's guide," Technical Report NDN-0021, NDN, Tech. Rep., 2014.

[12] [Online]. Available: http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer/html/GstBuffer.html