NdnRTC app design and protocol specification (DRAFT v0.1)

P. Gusev

September 3, 2013

Abstract

Contents

1 Overview

1.1 Participants and institutions

- Jeff Burke
- Jeff Thompson
- Peter Gusev
- Qiuhan Ding

1.2 Description

The video conferencing tool is one of those apps which is needed first of all for internal use and experimenting with real-time communication over NDN. As we are interested in increasing NDN popularity among non-research peers as well, we are trying to provide easy-to-setup NDN apps by employing web browser APIs. NDN-WebRTC is a JavaScript application which utilizes the WebRTC engine for media encoding/decoding and C++ NDN library for the transport layer. The proposed initial app design contains two main modules:

1. **Browser add-on**
   Contains the NDN media-specific transport layer implementation, the encoding/decoding engine (WebRTC) and provides a JavaScript API for access from browser apps;
2. **JavaScript app**
   Provides general conference discovery between peers

The app borrows some ideas from previous work [?, ?]. It maintains synchronization of a digest tree in order to keep track of current chat participants in the same manner as ChronosChat [?]. Given that information, a new participant can start fetching media data objects from other peers (according to the media namespace presented on Figure 1).

The main goal of fetching media is to minimize the delay of receiving the latest frames. Having that in mind, we came up with the following design assumptions:

- Consumers are in full control of the speed of issuing media interests (i.e. interests in a segment namespace);

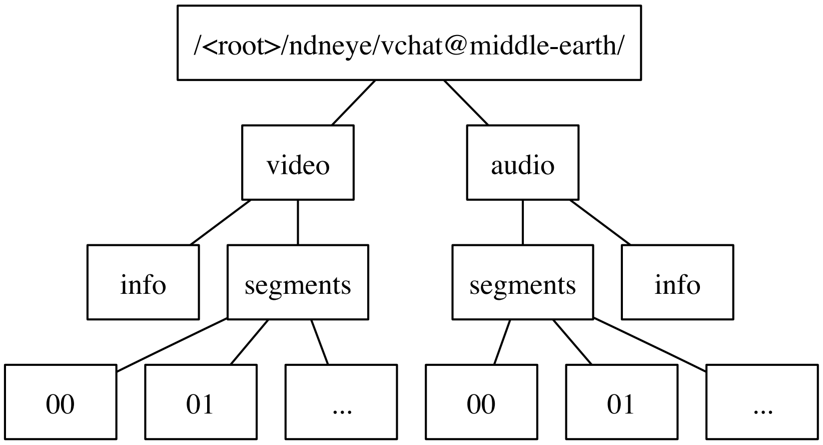

Figure 1: NDN-WebRTC Media Namespace

- Most recent media frames are delivered by pipelining media interests at an a priori higher rate than the producer delivers them (since the peer can obtain framerate information from the other peer by questioning the "info" namespace, it can determine the frequency of interest issuing);
- The presence of outstanding interests indicates retrieval of the most recent data;
- If the consumer has no outstanding interests, she increases the interest rate or, depending on the average rendering time, switches to a lower media quality.

Since no synchronization problems need to be solved in such a "consumer" approach, we hope that NDN-WebRTC can show better results in scalability for many-to-many scenarios.

1.3 Next steps

- Provide user authentication in video conferences
- Implement secure media transfer
- Scalable video encoding

2 Protocol specification

2.1 Intro

2.2 Discovery

TBD

2.3 Negotiating

1. Upon successful discovery of a video producer URI, the consumer issues an interest in the index namespace and gets data about the media stream's parameters ($FrameRate$ and codecs).

2. The consumer issues $Index_0$ with the *RigthMostChild=**true*** selector in the namespace for the frames like this: */root/mediadata* and waits until the first segment is received.

3. The consumer switches to **"Chase Mode"**

Chase Mode

1. THe consumer extracts the frame number - FN from the obtained *DataObject* and pipelines interests at a rate of $2 * FrameRate$ (twice the producer data rate) with *RightMostChild=**true*** in the namespace for segments like this: */root/mediadata/FN/0.* i.e. the pipeline contains interests $Index_{FN}$, $Index_{FN+1}$, $Index_{FN+2}$,...

2. The consumer watches two parameters: $DeliveryRate$ of frames, $RoundTripTime$ for interests:

   - If $DeliveryRate$ is the same as $FrameRate$ and $RoundTripTime$ is not growing steadily, the consumer switches to **Fetch Mode** (see below)
   - If either $DeliveryRate$ and/or $FrameRate$ are growing, the consumer chooses a lower quality (by modifying the prefix) and re-enables **Chase Mode**

Fetch Mode

1. The consumer extracts the latest frame number - LFN and sets up a *Frame interest* pipeline at a frequency of $2 * FrameRate$:

   for a 24 fps video:

   - **t = 0 sec**
     Interest for */root/video/LFN/0*, timeout = 1.5 sec
   - **t = 1/48s**
     Interest for */root/video/LFN+1/0*, timeout = 1.5 sec
   - **t = 2/48s**
     Interest for */root/video/LFN+2/0*, timeout = 1.5 sec
   - ...

   The number of segments per frame ($NumSegs$) is indicated using the *FinalBlockID* field of each segment's DataObject. The consumer sets up a *Segments interest* pipeline for each frame like this:

   - Interest for */root/video/LFN/0*
   - Interest for */root/video/LFN/1*
   - Interest for */root/video/LFN/2*
   - ...
   - Interest for */root/video/LFN/$SegNum$*

2. The consumer periodically issues a **"probe"** interest with the last known frame number and *RightMostChild=**true*** in order to detect a lag from the producer. If a considerable lag was detected, the consumer chooses stream parameters for a lower rate and switches to the **Chase Mode**

3. The consumer watches the value of *DeliveryRate* and *RoundTripTime* for the interests. If either of these values starts to grow, the consumer should choose stream parameters for a lower rate and switch to the **Chase Mode** in order to minimize the lag from the producer.

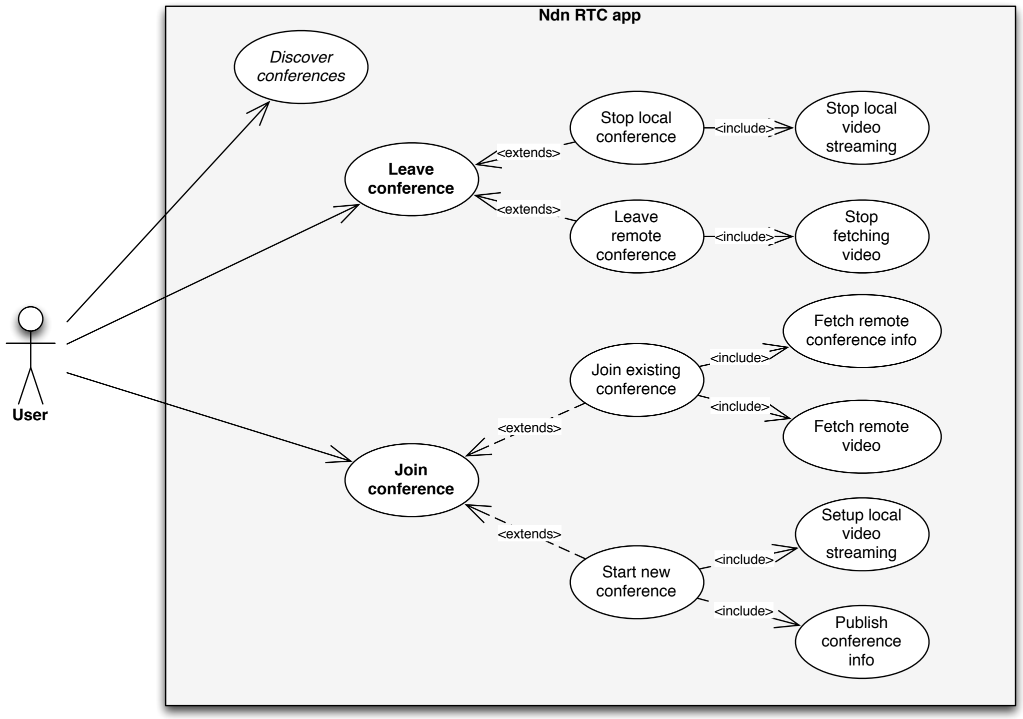

Figure 2: Add-on Use-Cases

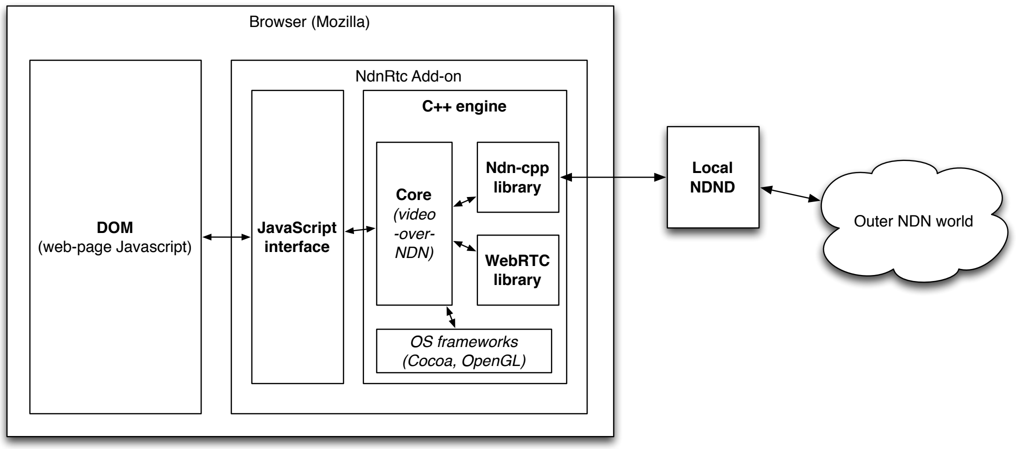

Figure 3: Add-on architecture

3 App design

Figure 2 represents common use cases for the add-on. Currently, use case *Discover conferences* is left for future development.

Top-view architecture of the add-on is presented on Figure 3.

3.1 C++ XPCOM add-on

This sections describes the internal architectural approach for the C++ part of the add-on.

One can start learning how the add-on works by looking at the sequence diagrams of the main use-cases: Starting a conference (Figure 4), Joining an existing conference (Figure 5) and Leaving a conference (Figure 6).

3.2 Javascript Web application

4 References

[?], [?]

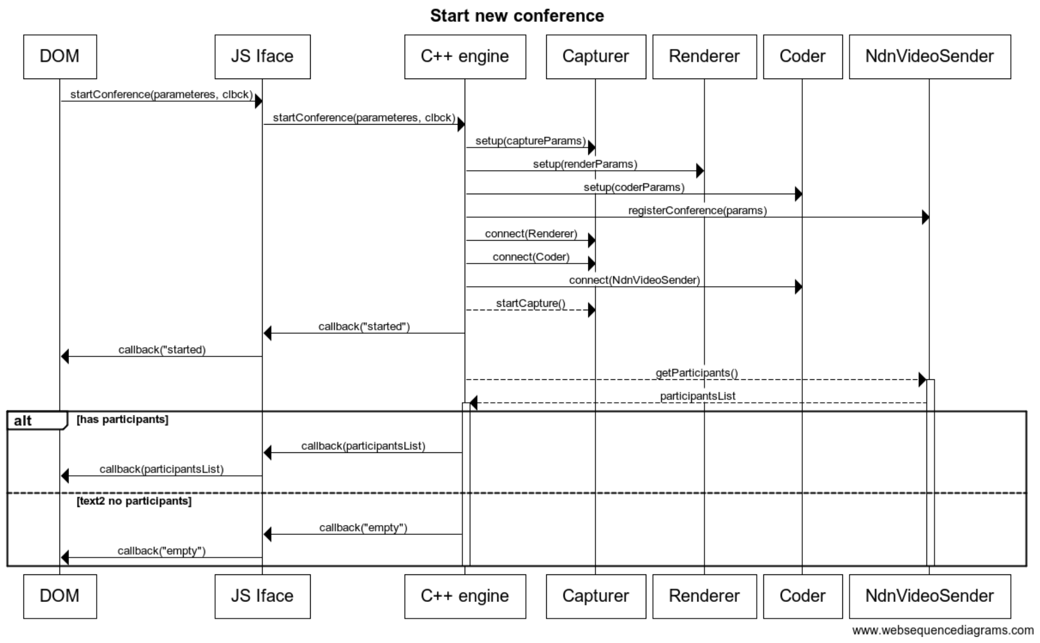

Figure 4: Sequence diagram for starting a conference

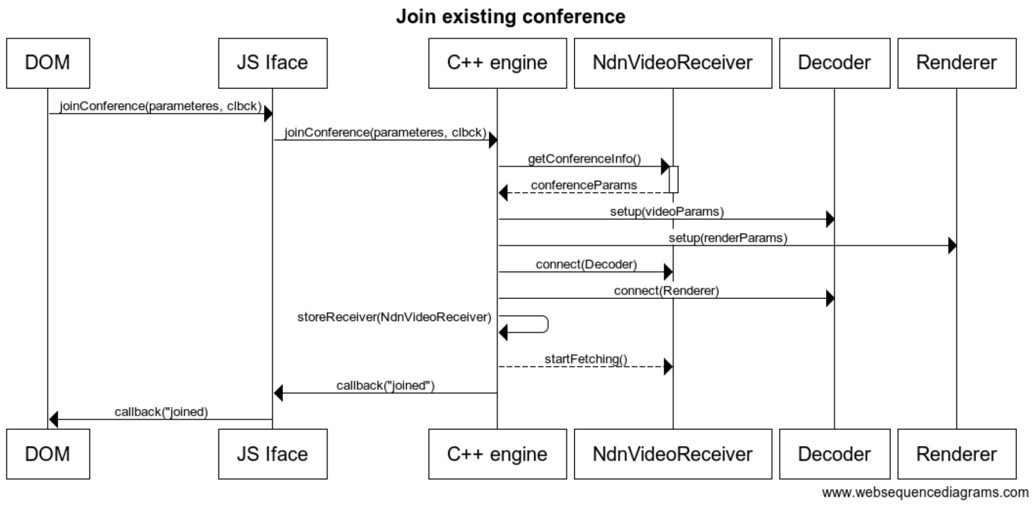

Figure 5: Sequence diagram for joining an existing conference

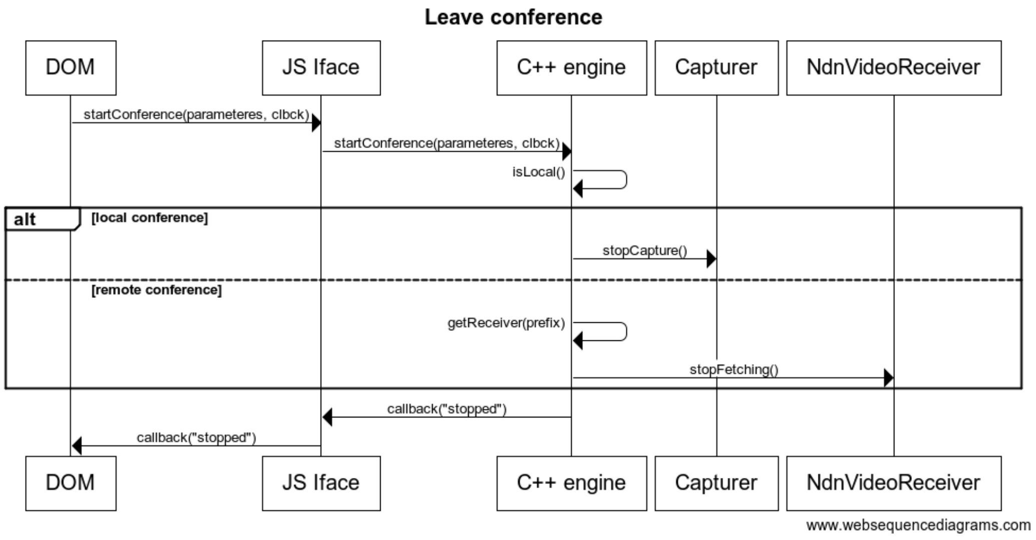

Figure 6: Sequence diagram for leaving a conference