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

P. Gusev

August 23, 2013

Abstract

Contents

1 Overview

1.1 Participants and institutions

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

1.2 Description

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 WebRTC engine for media encoding/decoding and C++ NDN library for transport layer. Proposed initial app design contained two main modules:

1. **Browser add-on**
   Contains NDN media-specific transport layer implementation, encoding/decoding engine (WebRTC) and provides JavaScript API for the access from the browser apps;

2. **JavaScript app**
   Provides general conference discovery between peers

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

The main goal of fetching media is to minimize the delay of receiving 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 segments namespace);

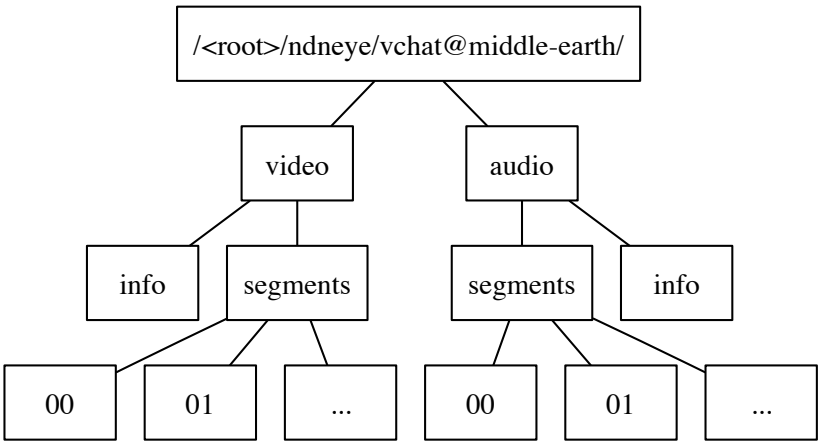

Figure 1: NDN-WebRTC Media Namespace

- Most recent media frames are delivered by pipelining media interests at a priori higher rate than producer delivers them (since the peer can obtain framerate information from the other peer by questioning "info" namespace, it can determine frequency of interest issuing);
- The presence of outstanding interests indicates retrieval of the most recent data;
- If consumer has no outstanding interests, it increases 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 "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 Consumer issues interest in index namespace and gets data about media streams parameters ($FrameRate$ and codecs).

2. Consumer issues $Index_0$ with *RigthMostChild=**true*** selector in frames namespace like this: */root/mediadata* and waits unless first segment is not received.

3. Consumer switches to **"Chase Mode"**

Chase Mode

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

2. Consumer watches two parameters: $DeliverRate$ of frames, $RoundTripTime$ for interests:

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

Fetch Mode

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

   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 ($SegNum$) is indicated using *FinalBlockID* field of each segments' DataObject. Consumer sets up *Segments interest* pipeline for each frame interest 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. Consumer periodically issues **"probe"** interest with the last known frame number and *RightMostChild=**true*** in order to detect lag from producer. If considerable lag was detected, Consumer chooses lower stream parameters and switches to the **Chase Mode**

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

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 add-on is presented on Figure 3.

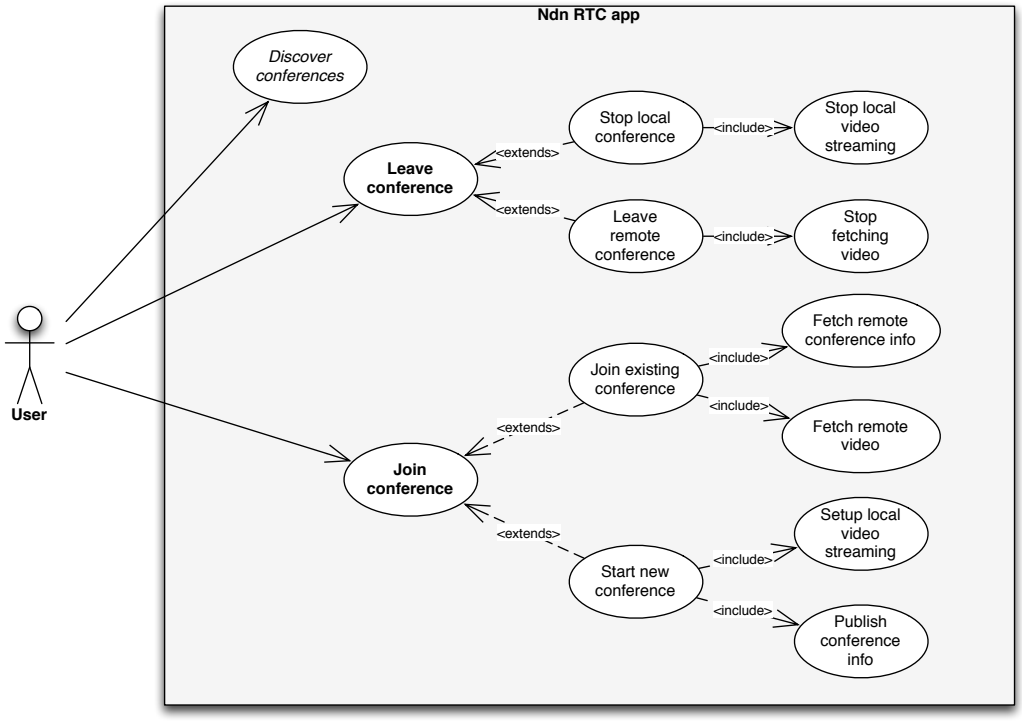

Figure 2: Add-on Use-Cases

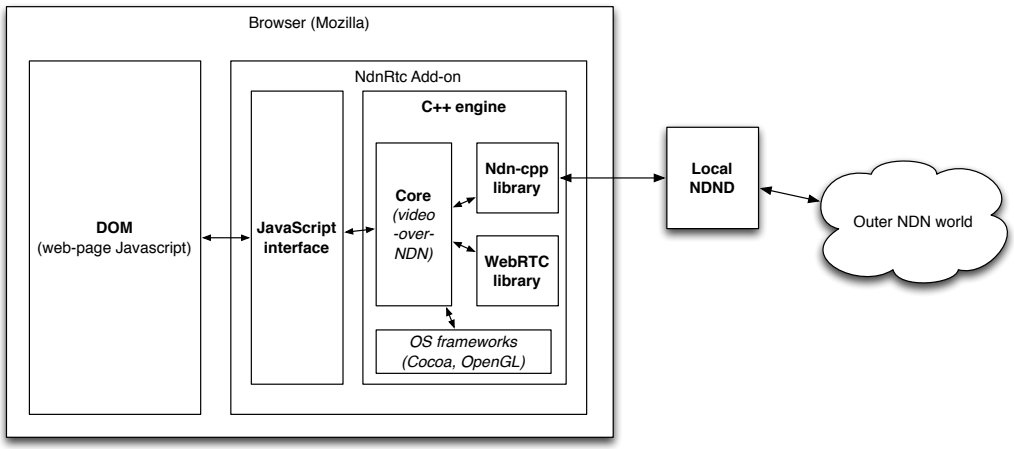

Figure 3: Add-on architecture

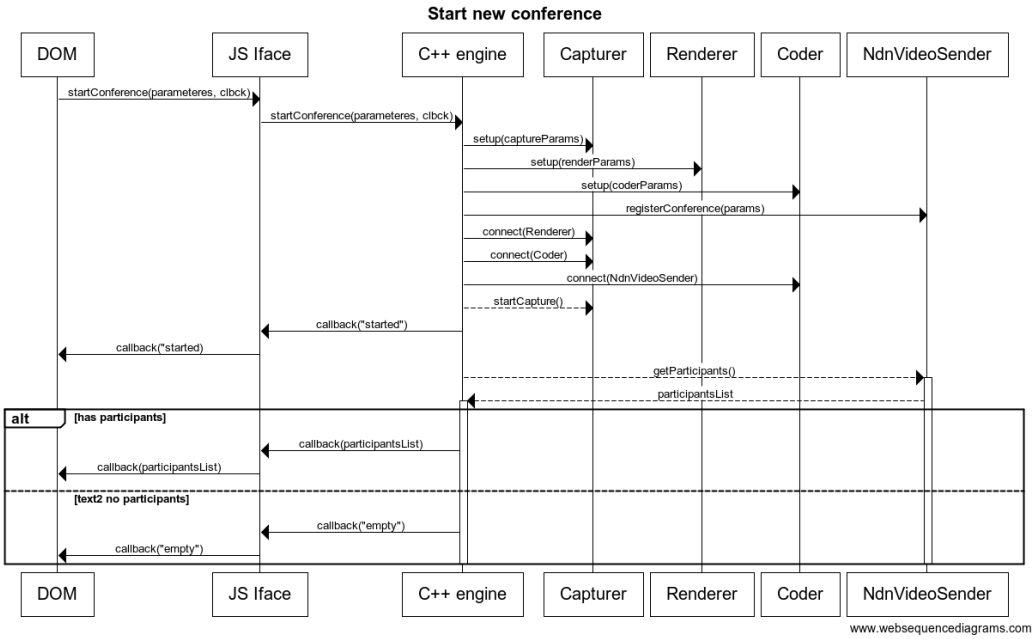

Figure 4: Sequence diagram for starting a conference

3.1 C++ XPCOM add-on

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

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

3.2 Javascript Web application

4 References

[?], [?]

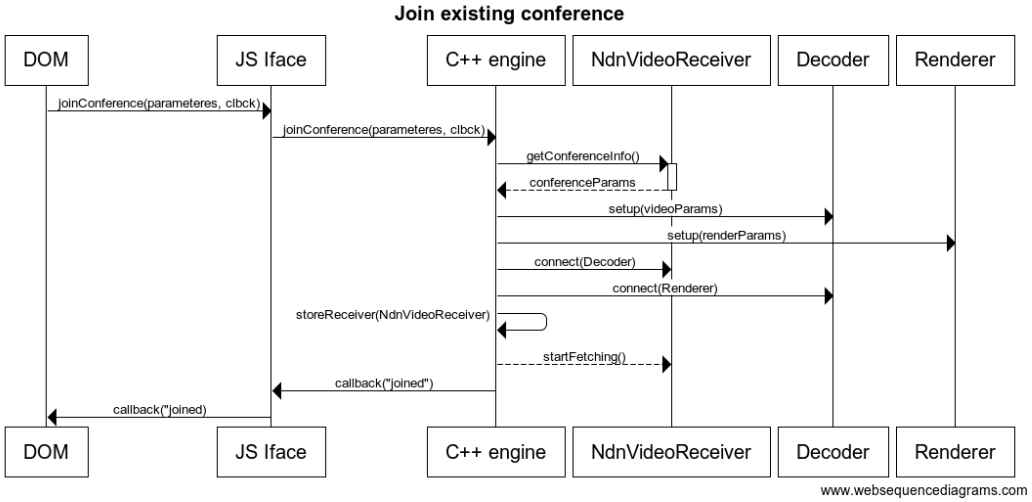

Figure 5: Sequence diagram for joining existing conference

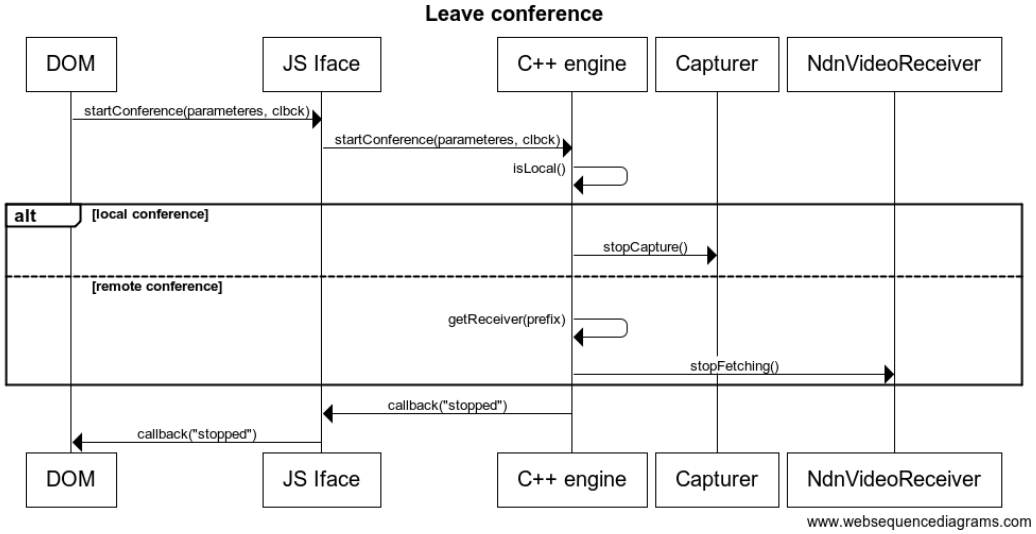

Figure 6: Sequence diagram for leaving a conference