NDN-RTC

Peter Gusev
UCLA REMAP
9/5/2014
NDNComm 2014 Demo

- **Producer 1:** Live NDNComm HD streaming (1080p 30fps, 1.5Mbps)
- **Producer 2:** REMAP office webcam producer (SD, 30fps, 500Kbps)
- **Demo 1:**
  - **Consumer for 3 streams:** NDNComm, REMAP and Demo-2
  - **Producer:** webcam producer (SD, 25fps, 500Kbps)
- **Demo 2:**
  - **Consumer for 3 streams:** NDNComm, REMAP and Demo-1
  - **Producer:** webcam producer (SD, 25fps, 500Kbps)
NDN Real Time Conferencing Library

Goals:

– Real-time audio/video/text chat library which allows many-to-many conferencing over the NDN network and requires no direct communication between peers
– Starting point for NDN traffic congestion control algorithm research
– Test NDN-CPP library and NFD
– Traffic generator for the testbed

Initial gains over IP:

– No load on a publisher (network does content distribution)
– Intrinsic multicast (one-to-many and many-to-many scenarios)
– On track for peer-to-peer with no STUN, TURN, etc.
NDN-RTC library

- C++ code
- Linked against NDN-CPP and WebRTC libraries
- Interfaces:
  - Publish media (audio/video) streams
  - Fetch media (audio/video) streams from multiple producers
- Demo app is provided
  - Publishing audio/video stream
  - Fetching audio/video streams (multiple)
Publisher

Camera

RAW frames stream

Encoder1

Cache

NDN

interests data
Publisher. Multiple encoder threads

Camera → RAW frames stream

Encoder1 → Encoder2 → EncoderN

Cache

interests data

NDN
Publisher. Multiple media streams
Segmentation

- **Encoded frames (1Mbps):**
  - Key: ~30KB (20 segments)
  - Delta: ~1-6KB (~4 segments)
- **Producer stores segments in app cache**
  - Segment size - 1000 bytes
  - NDN overhead - ~330-450 bytes
  - Complete segment less than MTU
User namespace

- **Root:**
  - User prefix (username)

- **Media streams:**
  - Media streams (audio/video)
    - Streams meta info

- **Encoding threads:**
  - Individual encoding parameters

- **Frame type:**
  - Key and Delta frames in separate branches

- **Packet:**
  - Individual media packets (audio samples, encoded video frames)

- **Data type:**
  - Data and Parity segments in separate branches

- **Segments:**
  - Actual NDN-data objects
Consuming

Interest pipeliner

NDN

Segments
%00%N1
%00%N2
...
%00%NN

Buffer

Renderer

Raw frame

Decoder

Encoded frame

Playout

NDN Interest pipeliner Interests

Interests

Segments

Buffer

Playhead
Frame fetching

- **Generation delay** $d_{gen}^n$ – time interval between receiving an interest and satisfying it with data (*producer-side*)
- **Assembling time** $d_{asm}^n$ – time needed to fetch all frame segments (*consumer side*)
- **RTT$_n$** – consumer-measured round trip time for the interest (*consumer side*)
Interest pipeline and retransmission

B₁ \geq RTT, B₂ \geq RTT

Minimal buffer size \geq 2 \times RTT milliseconds
Chase mode

- There is no direct coordination b/w consumers and producers
- Producer generates data at high rate (~20-30FPS) and this data becomes outdated fast
- **Start-up time:** consumer is aware that stale data is present in the network and tries to avoid playing it back
- **Chasing mechanism:**
  - Cache exhaustion:
    - Latest data can not arrive faster than it’s being produced – it arrives at producer’s rate
    - Cached data arrives with the same frequency it was requested
  - Chase mode:
    - issue interest for the RIGHTMOST segment
    - upon receiving first segment – start issuing interests for the next frames with interval $d^{int} < Producer rate$
    - Monitor $d^{arr}$ – frame inter-arrival interval:
      - If $d^{arr}$ is increasing – continue fetching
      - If $d^{arr}$ is stable – switch to “Fetch“ mode
Future improvement (suggested by Dave Oran):
1. piggyback video sync data on audio stream
2. use audio stream for chasing instead of video
Forward Error Correction

- OpenFEC library
- Producer publishes parity data under separate namespace:
  - `<frame prefix>/<frame#>/parity/<segments>`
- Consumer **may** additionally fetch parity data for enabling FEC
- If by the playback time frame is missing any segments – FEC is applied as the “last resort”
- Amount of parity data is configurable (currently 20%)
- Collaborated with Daisuke Ando (Exchange student from Japan)
- **Future improvement** (suggested by Dave Oran): use frame-level parity data rather than segment-level
Demo app

• Console app
  – MacOS X 10.9 and up
  – Buildable from sources
    github.com/remap/ndnrtc
  – Redmine
    redmine.named-
    data.net/projects/ndnrtc

• Functionality:
  – Publish audio/video stream
  – Fetch multiple audio/video streams
Future steps

• Real-time Adaptive Rate Control:
  – In collaboration with Panasonic R&D department (Muramoto-san, Yoneda-san)
  – Keep low-latency transmission & best throughput
  – Maintain RTT fairness (self-fairness)
  – Consumer-driven
  – NW bandwidth estimation based on RTT and timeouts
  – Control interest rate according to bandwidth estimation

• Conference discovery (Zhehao Wang)
• Text chat (Zhehao Wang)
• Browser integration (Zhehao Wang)
• Security
• Desktop conference tool
  – Adding modularity to the existing code
• Compare to existing solutions
  – Can be RTC over NDN better than IP?
• Scalability tests
Areas for future research

• Interests pipelining
  – Express just enough interests to fetch needed frames and meet the deadline, but keep low latency

• Alternatives to cache exhaustion
  – How consumer can be sure that it’s getting the latest data from the network without explicit producer-consumer signaling?

• Security
  – Trust model; signing and verification; encryption approach?

• Scalability
  – How many conference peers can there be?
  – What are the requirements for the forwarder?
  – What are the requirements for the peers?

• Relationship between forwarder strategy and application
  – Best route strategy 2
Links

• Source code
  – https://github.com/remap/ndnrtc
  – branches:
    • master – current released version (v0.9.alpha4)
    • dev – current development branch (v0.9.alpha5)

• MacOS binaries (library, demo, supporting files)
  – https://github.com/peetonn/ndnrtc-archive
  – Special branch for demo events:
    • demo/ndncomm2014

• Redmine
  – http://redmine.named-data.net/projects/ndnrtc/issues
Thanks

Q&A

Peter Gusev
peter@remap.ucla.edu