NDN-RTC: Real-Time Videoconferencing over Named Data Networking

Peter Gusev, Jeff Burke
UCLA REMAP
NDN-RTC Project Goals

- Experimental research in low-latency, real-time multimedia communication over NDN
- Functional videoconferencing library+application:
  - Low-latency, interactive data distribution:
    - Multi-party conferences
    - Live broadcasting
  - Data-centric security: schematized trust, name-based access control
  - Wide adoption by NDN community
- Testbed traffic generation and high-load performance testing
Why use NDN for RTC?

- Demonstrate viability of low-latency streaming over NDN
- Generalized approach
  - Explore other possibilities where low-latency fetching can be used
- Inherent network capabilities
  - Mobility
    - Producer provides namespace for published data
    - Consumer needs to know only names for fetching data from the network
  - Scalability
    - No direct coordination by producer
Design Objectives

• Achieve low-latency communication
  250-750ms for audio and video
• Straightforward publishing and fetching for multi-party conferences
• Passive producer & cacheability
  • No explicit coordination between producer and consumer
  • Enable exploration of network-supported scaling to high producer-consumer ratios
• Multiple bitrate streams
  • Supported by namespace
  • Enable near-future work on Adaptive Rate Control
• Data verification using existing NDN features
Architecture Overview

- Pull-based approach: complexity shifts to the consumer
- Producer
  - media acquisition & encoding
  - segmenting & naming according to namespace
  - adding segments the the network-aware cache
- Consumer
  - maintain Interests pipeline according to current network conditions
  - track data arrival and buffer level for Interests retransmissions
  - re-assemble segments, buffer, decode & playback
Typical Producer Setup
Application Namespace

Root:
- User prefix (username)

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

Encoding media 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
Segmentation & Metadata

- Encoded frames (1Mbps):
  - Key: ~30KB (30 segments)
  - Delta: ~1-6KB (~5 segments)
- Producer stores segments in app cache
  - Segment size - 1000 bytes
  - NDN overhead - ~330-450 bytes
- Metadata
  - Frame-level: encoding info, timestamps
  - Segment-level: generation delay, total segments number, etc.
Frame Fetching

- **Generation delay** $d_{\text{gen}}$ – time interval between interest receipt and data generation (*producer-side*)
- **Assembling time** $d_{\text{asm}}$ – time needed to fetch all frame segments (*consumer side*)
- **$RTT'$** – consumer-measured round trip time for the interest (*consumer side*)
Interests Pipeline and Retransmissions

- reserved slot - no segments fetched yet
- frame being assembled (some segments fetched)
- fully fetched frame

Initial interests

Data segments

Interests for missing segments

Missing data segments

Playback pointer

Reserved size

Estimated buffer size

Playable size

RTX checkpoint
Interest Expression Control

- **Consumer challenge:** ensure acquisition of the latest data without resorting to **direct communication** with the producer and given the presence of **network cache**

- **Observation:** fresh data arrives at producer rate, cached data mimics Interest expression pattern

- **Consumer goal:** receive data at a consistent rate, not reach producer directly

Bursty arrival of stale data copies Interests expression pattern

Periodic arrival of fresh data reflects publishing sample rate
Interest Demand

- **Outstanding Interests** ensure latest data delivery
- The **minimal number** of outstanding Interests that ensures latest data retrieval defines “Interest Demand”

- **Interest Demand** ($\lambda$) driven by:
  - DRD (Data Retrieval Delay) – generalized RTT
  - Data inter-arrival delay (producer publishing delay observed by consumer)

  $$\text{Interest Demand} = \frac{\text{DRD}}{\text{D}_{\text{arr}}}$$

- Consumer changes **Interest Demand** value in order to adjust fetching **aggressiveness**

- Data-driven Interest expression:
  - Quicker response to new network and publishing conditions
  - Faster and more robust bootstrapping
Bootstrapping

- **Bootstrapping mode**: seek through cached data quickly until *freshest data* begin to arrive
- Main indicator: packet inter-arrival delay $D_{arr}$
- Interest Demand adjustment:
  1. Initialize $\lambda_D$ and initiate Interest expression
  2. If no fresh data in allocated time – *increase demand*: $\lambda = \lambda + 0.5\lambda_D$; $\lambda_D = \lambda_D + 0.5\lambda_D$
  3. If cache exhausted – *decrease demand*: $\lambda = \lambda - 0.5\lambda_D$; $\lambda_D = \lambda_D - 0.5\lambda_D$ and wait for one of two outcomes:
     a) $RTT'$ decreases and freshest data continues to arrive – repeat step 3
     b) Cached data starts to arrive – restore to the previous $\lambda_D$, bootstrapping ended.

30 FPS; 100ms RTT;
Bootstrapping

Conditions:
- FPS: 30
- GOP: 30
- RTT ~100ms
- $\lambda_{\text{start}} = 30$

Results:
- $RTT' \sim 110$ms,
- Unstable phase $\sim 700$ms, 1600ms
- $\lambda_{\text{final}} = 4$
Iterative Design Improvements

• Consumer-producer synchronization
  • *Interest demand* allows to adjust fetching aggressiveness of the consumer
  • Consumer reduces number of pending Interests in PIT, thus achieving better synchronization with the Producer

• Streaming performance:
  • Namespace separation of *Key* and *Delta* frames
  • Audio packet bundling

• NFD: early retransmissions strategy
  • App retransmission was suppressed until Interest times out in PIT
  • Varying Interest lifetime is risky when data is not produced yet or network conditions change
  • BestRoute2 strategy allows early app retransmission without giving up Interest lifetimes
Implementation Details

• C++ library (OS X, Ubuntu)
• WebRTC for audio pipeline
• VP8/9 video encoder
• Forward error correction with OpenFEC
• Open source
  github.com/remap/ndnrtc
• GUI OS X conferencing app on top of NDN-RTC – ndncon
  github.com/remap/ndncon
Future Work

• Adaptive Rate Control (*in progress*)
• Linux compatible version (*in progress*)
• Ubuntu headless app (*in progress*)
• Further tests
  • multi-party uni- and bi-directional tests (*ongoing*)
  • NFD performance stress tests (*ongoing*)
  • large-scale tests using headless Ubuntu app
• Data authentication and encryption with multi-party support
• Scalable video coding
Challenges

• How to reduce consumer reaction delay?
  • No direct producer-consumer communication
  • Robust freshest data detection
  • Faster reaction to network conditions

• How to efficiently encrypt media without losing NDN advantages?
  • Depends on application objectives – Reformulate conferencing?
  • Leverage broadcast encryption and other schemes

• How to achieve inter-consumer synchronization?
  • While preserving no direct communication
  • Consider varying network conditions
Opportunities for Collaboration

• NDN project team plans to use and improve *ndncon*. Help welcome!
• Others can use NDN-RTC library for creating more applications.
  • NDN-RTC repo: [github.com/remap/ndnrtc](http://github.com/remap/ndnrtc)
  • *ndncon* repo: [github.com/remap/ndncon](http://github.com/remap/ndncon)
• Deeper research into rate control, interest expression algorithms needed.
• Need to do simulations to look at algorithm performance under various caching conditions, topologies, and use cases.
Thank you!

Q&A
Additional Slides
Example NDN-RTC-driven Improvements

• NFD: Revised retransmissions strategy
  • App retransmission was suppressed until Interest times out in PIT
  • Varying Interest lifetime is risky when data is not produced yet or network conditions change
  • BestRoute2 strategy allows early app retransmission without giving up Interest lifetimes

• NDN-CCL: Library support for app-level PIT
  • Common low-latency case: handle Interests that arrive before data is ready
  • Need to store Interests in producer-side PIT
  • Same approached used in OpenPTrack real-time person-tracking

• Testbed/NFD: Performance stress-tests (ongoing)
  • 3-9Mbit/sec data streams per producer
  • 9Mbit/sec: ~1000 Interest/sec, ~900 data segments/sec
  • Traffic generator for the testbed
Design & Development Progress

- **Design**
  - “Interest Demand” concept introduction
  - Audio packet bundling
- **Implementation**
  - Desktop GUI application *ndncon*
    - group chats (ChronoChat2013)
    - automatic user discovery (ChronoSync2013)
    - screen sharing
  - Thread optimization
    - single-threaded architecture, decreased CPU
  - Asynchronous logging
  - Automated test environment (local testbed, NDN testbed)
- **Ported to Ubuntu**
  - special thanks to Luca Muscariello (Orange), Zhehao Wang (UCLA)
Adaptive Rate Control

- Collaboration with Panasonic R&D department
- Established development plan:
  - NDN-RTC modifications, REMAP - **October 2015**
  - ARC implementation\(^1\), Panasonic - **November 2015**
  - Early tests - **December 2015**
  - Full tests – **January-February 2016**
  - Completion – **March 2016**
- Implementation details\(^2\)
  - Gapless stream switching
  - Challenging Interests for bandwidth probing
  - Ongoing monitoring of intrinsic network parameters (DRD, D_{arr}, etc.)
