Section Outline

- NDN Real-Time Communication library (NDN-RTC):
  - architecture & hands-on session

- “Low-latency media”-friendly forwarding strategies

- Congestion control

- Real-time Data Retrieval (RDR) protocol
Low-Latency Media Streaming for AR

AR Browser “under the hood”:

• NDN communication throughout

• **NDN-RTC** library for low-latency media delivery

• C++ library & tools:
  • github.com/remap/ndnrtc
Hands-on with NDN-RTC

• Prerequisites:
  • macOS >= 10.12
  • NDN platform
  • Homebrew

• Install:
  • brew update
  • brew tap remap/ndnrtc
  • brew install ndnrtc ndnrtc-stream

• Run:
  • ndnsec-keygen /hello-ndn | ndnsec-install-cert –
  • ndnrtc-stream publish /hello-ndn
  • ndnrtc-stream fetch /hello-ndn/rtc-stream
NDN-RTC C++ Library
Low-latency audio/video streaming over NDN

- First prototype in fall 2013
- Peer-to-peer approach, host-independent
- HD-capable video streaming
- Multiple bitrate streaming
- Audio streaming (echo cancellation)
- NDN-RTC-based NDN apps
  - 2013 - ndnrtec-demo, command-line
  - 2014 - ndncon, desktop conferencing app
  - 2015 - ndnrtc-client, headless client
  - 2017 - Docker containers
  - 2017 - Flume (prototype)
  - 2018 - AR Browser
Library Namespace

/hello-ndn/rtc-stream

/hello-ndn/rtc-stream/ndnrtc/3/video/camera

/hello-ndn/rtc-stream/ndnrtc/3/video/camera/720p/d/342/%00%00
Schematized Trust

Certificate Authority
/ndn/edu/ucla/remap

User

Signing Identity
/ndn/edu/ucla/remap/peter

Application Identity
/ndn/edu/ucla/remap/peter/myapp

Application

Instance Identity
/ndn/edu/ucla/remap/peter/myapp/02def3

NDN-RTC

/hello-ndn

/hello-ndn/rtc-stream
Library Architecture

- **Producer**
  - slices encoded frame into segments
  - stores segments to the memory cache
- **Consumer**
  - ensures low-latency delivery using Interest pipeline
  - re-assembles segments into frames
  - queues frames in the buffer
  - decodes & plays back frames

---


NDN-RTC Development. Experience and Goals

• Continuous development & improvements since 2013
• Multi-peer testing over the testbed
• Test out NDN with real apps
• Drive network architecture and NDN app development
• Build essentials: streaming over NDN
• Learning:  
  • how to build NDN apps  
  • how to design network  
  • how to deal with congestions  
  • how to retrieve latest data in the presence of caches
Challenges for forwarding strategy

• “A forwarding strategy decides how to forward an Interest.” *

• How to distinguish multiple consumers with retransmissions from the same consumer

• Forwarding plane measurements for applications with diverse naming schemes

* NFD Developer’s Guide
Lessons learned

• **Best route strategy**
  - Interest with same Name+Selectors coming from the same face → retransmission*
  - Interest with same Name+Selectors coming from a different face → another consumer is requesting the same Data, should be suppressed for the first time*

• Strategy measurements of forwarding plane performance:
  - Work with diverse application namespaces (not only "/prefix/segment-number")

*Redmine issues
https://redmine.named-data.net/issues/3230
https://redmine.named-data.net/issues/3219
https://redmine.named-data.net/issues/3485
Congestion control for NDN

• Traditional congestion control does not work for NDN

• Congestion control in NDN is challenging:
  • pull-based approach
  • multiple paths and endpoints
  • diverse deployment scenarios (wired, IP tunnels, wireless, etc.)
  • hard to determine “per-content fairness” with interest aggregation
Hop-by-Hop congestion control (PCON*)

• Design principles:
  • detect congestion at the bottleneck
  • signal congestion towards consumer
  • remove strong assumptions:
    • unknown link capacity & Data chunk size
    • no route-labels or data location predictions
Real-time Data Retrieval (RDR) protocol
Retrieval of Latest Real-time Data

• How to fetch the latest generated real-time data in a network with caching
• Segment numbers instead of application-level timestamps
• Need to know the exact data names to pipeline Interests to fetch multiple data segments
• Need to know the exact name of data that has not been produced yet

/real/time/data/name/0
/real/time/data/name/1
... 
/real/time/data/name/N
Why bother?

After 1 minute

Retrieved Data generated a minute ago!
Retrieve Latest Data by Making Use of Protocol Features

• Producers generate *metadata* for real-time sessions:
  • Determines for how long metadata stays fresh at each-hop CS (FreshnessPeriod)
  • Name of the latest data
  • (Optionally piggybacking) The latest generated segment

• Consumers fetch *fresh metadata* (MustBeFresh)
  • Bypass “non-fresh” cached metadata
  • Learn the exact name of the latest generated data (before \( \frac{1}{2} \) RTT)
  • Determine exact name of data to be produced in the future through naming conventions
Metadata for NDN-RTC Streaming

Name: /NDNcall/ndnrtc/user/Spyros/_metadata

MustBeFresh
CanBePrefix

Can pipeline interests to fetch delta frame 93, 94, ...

Can pipeline interests to fetch key frame 4, 5, ...

Name: /NDNcall/ndnrtc/user/Spyros/_metadata/_v=10

FreshnessPeriod: 10ms

Latest Delta Frame Name:
/NDNcall/ndnrtc/user/Spyros/.../frames/delta/92

Latest Key Frame (First Segment)
Name: /NDNcall/ndnrtc/user/Spyros/.../frames/key/3/data/0
...

Signature
FreshnessPeriod Considerations

• FreshnessPeriod is a per-hop relative metric

• Metadata may stay fresh for a longer (absolute) time further away from the producer (due to processing and propagation delays)

• FreshnessPeriod value:
  - **Short enough**: Avoid stale metadata further away from the producer
  - **Long enough**: protect producers from excessive requests

(worst possible case)
How to learn more


