NDN Libraries
Progress and Plans

March 24, 2019
Jeff Thompson, NDN Team
jefft0@remap.ucla.edu
Overview

• Common Client Libraries (CCL)
• PSync
• Common Name Library (CNL)
• NDN-RTC
• Quick summary of recent research progress
What are the Common Client Libraries (CCL)?

• Enable client applications to use NDN in C++, Python, JavaScript, Java, .NET
• Common API across languages: [http://named-data.net/doc/ndn-ccl-api](http://named-data.net/doc/ndn-ccl-api)
• Interest/Data, signatures, encryption, transports, app utilities, unit tests, examples
• Track ndn-cxx research (security, NAC, NDN protocols, NFD interaction)
• Backwards compatibility, platform flexibility for development stability
• Used in NDN-RTC, BMS, mHealth, neighborhood network, web page apps, ICE-AR
• Specialized libraries: NDN-CPP Lite (Arduino), Imp, Android, browser speedups
• Stats (total): 10,771 commits, 277 closed issues, 79 pull requests, 80 forks
Example

```python
face = Face("memoria.ndn.ucla.edu")
name = Name("/ndn/edu/ucla/remap/demo/ndn-js-test/hello.txt/%FDU%8D%9DM")
def onData(interest, data):
    print(data.content.toRawStr())
face.expressInterest(name, onData)
```
CCL Features

• Certificate signing/validating – RSA, ECDSA, HMAC
• Configurable cert chain policies, regex name matching
• Flexible public/private key database API
• Signed Interests – verify with same API as certs
• Name-base access control (AES encryption, RSA key protection)
• MemoryContentCache, SegmentFetcher
• Optional thread-safe network I/O
• Configurable wire format (see below)
• ChronoSync, PSync (see below)
• Unit tests, example programs
CCL wire format abstraction

- API is not hard-wired to one wire format
- Enable backwards compatibility if running with old forwarders
  
  ```java
  WireFormat.setDefaultWireFormat(Tlv0_1WireFormat.get())
  ```
- Can specify on ad hoc basis if sending to a various networks
  
  ```java
  face.expressInterest(name, onData, Tlv0_1WireFormat.get())
  ```
- Was used for transition from CCN 0.x
- Plans to support other ICN wire formats
CCL – Next steps

• NDN wire format v0.3 (with backwards compatibility)
  • Typed name components
  • Removed (most) Interest selectors
  • Interest hop count
  • Interest defaults to exact name (optional CanBePrefix)
  • Extra application parameters in the Interest
  • Explicit fields for signed interests (instead of using name components)

• New wire formats

• Support new network autoconfig protocols
What is PSync?

- Developed as improvement to ChronoSync
- Used in NLSR to sync routes on the NDN test bed
- Part of the CCL
- Invertible Bloom filter of a set of hashed names
  - Send interest with my IBF, receive interests with others’ IBF
  - Stable state: Everyone sends the same IBF – Interest aggregation, no Data
  - Update: I receive a different IBF with missing names and provide in reply Data
  - IBF efficiently updates a set difference of ~275 names
- Eventual consistency from pairwise updates – broadcast not needed
- Option to subscribe to partial namespace updates
Example PSync app

```python
face = Face()
def onNamesUpdate(names):
    print("Got names, starting with " + names[0].toUri())

updateSize = 80
pSync = FullPSync2017(updateSize, face, Name("/sync"), onNamesUpdate)
pSync.publishName(Name("/edu/ucla/jefft/paper.txt"))
```
PSync – Next steps

• Implement in Python, JavaScript, Java (currently in C++)
• Use as native sync for the Common Name Library (see below)
• Stress test “eventual consistency” without broadcast
• Support partial PSync (waiting for use case)
• NDN Project: A Quick Summary of Recent Progress
What is the Common Name Library (CNL)?

• Library enabling applications to work with hierarchical, named data collections.
  • Namespace object (root and child nodes)
  • Application interacts with a Namespace node (attach handlers, receive notifications)

• Provides a lightweight way to integrate various:
  • Sync mechanisms (i.e., PSync, vector sync)
  • Data access patterns (i.e., Consumer/Producer API, fetch latest),
  • Publishing models (i.e., publish/subscribe, in-memory content cache),
  • Complex namespace queries / pattern matching (i.e., regexp, wildcards),
  • Triggered data generation (supporting security)

• Currently using in ICE-AR mobile client application
  (No interest-data exchange exposed to developers of that app.)

• Segmented content with a Meta packet and versioning

• Built-in encode/decode, encrypt/decrypt, sign/verify as part of the pipeline

• New names added to the Namespace tree through PSync, app is notified
CNL Motivation

• Provide tools for working with namespaces as they represent collections, in an information-focused rather than communication-oriented way

• Assume asynchronous network operations will be used to sync the namespace and consume/publish objects in the collection

• Insulate non-networking developers from communication details

• Make progress towards NDN as a middleware-replacement in terms of high-level, application-facing features, but try to stay as general as possible

• Work with aggregate application-specific objects, not (segmented) blobs in packets

• As a result, support namespace synchronization the way that is conceived / described at a high-level, and promote it as an application-level concept to explore
Example segmented content consumer app

```python
face = Face("memoria.ndn.ucla.edu")
page = Namespace("/ndn/edu/ucla/remap/demo/ndn-js-test/named-data.net/project/ndn-ar2011.html/%FDX%DC5B")
page.setFace(face)

def onSegmentedObject(namespace):
    print("Got segmented object size " + str(namespace.obj.size()))

page.setHandler(SegmentedObjectHandler(onSegmentedObject)).objectNeeded()
```
CNL Handlers

• Support extensibility
• Set Namespace for special fetching, publishing, object representation
• Unified API for developers too
  • https://github.com/named-data/PyCNL/blob/master/python/pycnl/segmented_object_handler.py
Unified publisher/consumer

- **objectNeeded()** – From application (producer) or network (consumer)
- **Producer**
  - CNL receives Interest, adds to PIT, calls OnObjectNeeded (if not already in cache).
  - Handler’s OnObjectNeeded answers True.
  - CNL waits for application to produce data asynchronously.
  - Application calls setObject().
  - CNL does serialize/encrypt/sign and satisfies PIT.
- **Consumer**
  - Application calls OnObjectNeeded for a Namespace node.
  - (All handlers answer False.)
  - CNL does Face.expressInterest and waits for Data.
  - CNL receives Data, does verify/decrypt/deserialize and OnStateChanged(OBJECT_READY)
NDN-CNL: Name node state diagram

Integrating Interest/Data and Packet-/Prefix-level objects
CNL – Next steps

• High-performance persistent storage
• Port to Java and JavaScript
• More applications
  • Currently used in augmented reality mobile client application
What is NDN-RTC?

- NDN C++ video (HD) streaming library for macOS, Ubuntu, Android
- Sub-second (~150ms) latency
- Based on VP9 video encoder
- Pipelining, retransmission, FEC
- Unified consumer for live and stored video
- Recent updates
  - RDR protocol for getting latest data
  - Remove dependency on WebRTC (though no echo cancellation pipeline for now)
  - New “Fast Repo” rocksdb-based storage
  - Moving towards generalized approach for objects, metadata, etc.
Applications

Current
• ICE-AR (AR browser as part of Intel/NSF ICN-WEN)
  • NDN-RTC streams phone POV video for edge processing (object, face, pose recognition)
  • processed information delivered back to the phone to enrich phone’s environmental understanding (deep context)
• TouchNDN (integration with Derivative’s Touchdesigner)
  • aiming to replace NDI for live video production
  • leverage NDN to efficiently disseminate live video over L2 (or L3) to multiple nodes for simultaneous processing & storage
  • nodes integrate “historical” playback from repo data seamlessly with live streaming, for scrubbing real-time streams
• Assorted command line tools

Previous
• ndncon/flume – pure P2P conferencing app
  • not up to speed with latest library
Future Plans

- Scalable video coding (VP9)
- Region-of-Interest-based fetching (360° video use case)
- Volumetric video streaming
- Congestion control, when apps need it (based on Schneider 2016)

=> Looking for app users and codebase collaborators
Recent NDN Code Release Updates

• NFD and ndn-cxx version 0.6.5
• ndn-tools version 0.6.3
• NDN Android version 0.6.5-3
  • Based on the latest version of NFD (0.6.5)
  • Including updated GUI based on work at NDN hackathon
• ndnSIM 2.7
  • Based on the latest released versions of NS-3 (version 3.29) and NFD (version 0.6.5)
    https://ndnsim.net/2.7/RELEASE_NOTES.html
• Mini-NDN 0.40
  https://github.com/named-data/mini-ndn/releases/tag/v0.4.0
• Named-data Link State Routing Protocol (NLSR) version 0.5.0
  https://named-data.net/doc/NLSR/0.5.0/RELEASE-NOTES.html
Recent New Code Releases / In progress

• pSync, a synchronization protocol for NDN
  https://named-data.net/doc/PSync/0.1.0/RELEASE-NOTES.html

• NDN IoT Package

• Mini-NDN-WiFi
NDN IoT Package

• An NDN-based IoT framework with two goals
  • Localized trust and automated security management
  • Ease-of-use IoT software development kit

• Features
  • Lightweight NDN software stack and forwarder, specifically tuned for constrained devices
  • Seamless integration of heterogenous link layer protocols (BLE, WiFi, IEEE 802.15.4, etc.)
  • Ease-of-use high-level APIs for bootstrapping, service discovery, access control, and schematized trust management
  • Easy adaptation to new IoT hardware/software platforms

• Ongoing efforts
  • Further memory-saving NDN forwarder design
  • Demonstrative application to illustrate how to build an IoT system in a fundamentally different way from today’s IP-based solutions

• More detailed to be reported @ next IETF
8th NDN Hackathon (March 8-10, 2019)
http://8th-ndn-hackathon.named-data.net/hacks.html

• **First Prize**: *NFD-Android Enhancements*
  • Alex Afanasyev, Ju Pan, Sanjeev Kaushik Ramani, Davide Pesavento

• **Second Prize**: *Sigcomm Tutorial App (NDN-IoT demo)*
  • Zhiyi Zhang, Xinyu Ma, Edward Lu, Yu Guan, Erynn-Marie Phan, Laqin Fan

• **Third Prize**
  • *Self-Learning for Ad Hoc Wireless Networks*
    • Md Ashiqur Rahman, Davide Pesavento
  • *Sync in MANET Library + Demo*
    • Tianxiang, Zhaoning, Spyros
  • *Addressing ndncatchunks Performance Issues*
    • Klaus Schneider, Saurab Dulal
NDN Project at Google Summer of Code

https://summerofcode.withgoogle.com/organizations/6559809451589632/
Publications/Presentations/Tech Reports

• “A Note on Naming and Forwarding Scalability in Named Data Networking” Yu Zhang et al, ICC 2019 Workshop, May 2019

• “The Role of Data Repositories in Named Data Networking” Lixia Zhang et al, ICC 2019 Workshop, May 2019

• “Proof of Authentication for Private Distributed Ledger” Zhiyi Zhang et al, NDSS 2019 workshop, Feb 2019


• “Packet Forwarding in Named Data Networking Requirements and Survey of Solutions” Zhuo Li et al, to appear in IEEE Communications Surveys & Tutorials 2019

How to learn more

• Common Client Library (CCL)
  • C++: [https://github.com/named-data/ndn-cpp](https://github.com/named-data/ndn-cpp)
  • Python: [https://github.com/named-data/PyNDN2](https://github.com/named-data/PyNDN2)
  • JavaScript: [https://github.com/named-data/ndn-js](https://github.com/named-data/ndn-js)
  • Java: [https://github.com/named-data/jndn](https://github.com/named-data/jndn)

• PSync: Scalable Name-based Data Synchronization for Named Data Networking
  • [https://named-data.net/publications/scalable_name-based_data_synchronization/](https://named-data.net/publications/scalable_name-based_data_synchronization/)

• Common Name Library (CNL)
  • C++: [https://github.com/named-data/cnl-cpp](https://github.com/named-data/cnl-cpp)
  • Python: [https://github.com/named-data/PyCNL](https://github.com/named-data/PyCNL)

• NDN-RTC: [https://github.com/remap/ndnrtc](https://github.com/remap/ndnrtc)