

NDN Development Support - Common Client Libraries

NDNcomm 2015
September 28, 2015

Jeff Thompson

Overview

- What is the Common Client Library (CCL)
- Current features
- Planned features
- Questions from you on how to use CCL?
- How can you contribute?

Common Client Libraries

- Enable client applications to use NDN in C++, Python, JavaScript and Java
 - [NDN-CPP](#) – e.g. ndnrtc conference app
 - [PyNDN](#) – e.g. Building management monitoring
 - [NDN-JS](#) – e.g. Browser ChronoChat, Firefox plug-in, wiki (Ryan Bennett)
 - [jNDN](#) – e.g. Android mHealth, edge analytics (Andrew Brown, Intel - unreleased)
- Common API across languages: <http://named-data.net/doc/ndn-ccl-api>
- Includes example programs (e.g. [test-encode-decode-interest](#))

Development process

- Respond to application needs. E.g.
 - MemoryContentCache: ndnrtc needed to “push” video frames
 - NDN-CPP Lite: Arduino needed C++ without exceptions, shared_ptr
- When to port features from ndn-cxx
 - When needed, e.g. security library sign/verify
 - When NFD/test bed packets change, e.g. LP headers, certificates
 - Not needed yet, e.g. typed name components
- Language-specific features
 - Keep a minimal uniform API across languages
 - Add language-specific features as needed, e.g. Python interest.name[:-1]
 - Avoid unnecessary language use, e.g. C++ template programming

Library features: all libraries

- NDN-TLV Interest, Data, certificates
 - Soon: certificate format 2.0 (TLV)
- Generate key pair, sign/verify interest and data packets, verify policy
- ECDSA signatures (NDN-CPP and jNDN)
- MemoryContentCache
- SegmentFetcher
- ChronoSync 2013
- (removed NDNx / binary XML support)
- Soon:
 - Link objects and selected delegation in Interest
 - Support for NDNLv2 (link layer headers)
 - Group encryption protocol
 - Signature HMAC with SHA256
 - Interface to nfd-autoconfig-server

Library features: language-specific

- NDN-CPP Lite - thin C++ on top of pure C (for Arduino)
- NDN-JS: `crypto.subtle` for fast browser sign/verify
- NDN-JS: IndexedDB persistent storage for the browser
- Asynchronous I/O and thread safety:
 - NDN-CPP: Boost `asio::io_service`
 - PyNDN: `asyncio` (and `trollius`) `get_event_loop()`
 - jNDN: `nio ScheduledExecutorService` (thanks to Andrew Brown)
 - (NDN-JS: JavaScript is already single-threaded and async)

Example programs

- Encoding, signing:
 - encode-decode-interest, encode-decode-data, encode-decode-fib-entry
 - (The real file name is “test-**encode-decode-interest.js**”, etc.)
- Communication:
 - get-async, publish-async-nfd, echo-consumer
- Configure NFD (similar to `nfdc`):
 - register-route, list-rib, list-faces, etc.
- Using `repo-ng` (in `examples/repo_ng`):
 - basic-insertion, watched-insertion
- Others:
 - chrono-chat, encode-decode-benchmark, get-async-threadsafe

Discussion

Want more details on any of these? Feedback?

<http://named-data.net/codebase/platform/ndn-cc/>