#1. Adopt Mini-NDN for testing any project based on NFD

- Motivation
  - Currently, to test an application that runs over NFD, a user may need to configure multiple machines for the NDN platform. Users may currently use multiple virtual machines, Docker instances, or with physical machines. But, the process of configuring an NDN network for application testing can be simplified with the use of Mini-NDN, a Mininet based project that emulates a network on a single machine.
- Tasks
  - Modify Mini-NDN such that it can easily run NFD integration tests
    - Currently, real machines are used to run integration tests. Using real machines can make configuration tough and changing topologies difficult. If these tests are instead run in Mini-NDN, speed and effectiveness of development (#2940) will greatly improve.
  - Implement a system such that integration tests can be triggered from the upload of a patch to Gerrit. Mini-NDN can then be used to run the tests and confirm correct code behavior (#3053)
- Requirements
  - Linux with Mini-NDN installed, Python, Bash
- Metrics
  - Improvement can be shown by comparing the time and resources it took a team to setup and test their application before and after using Mini-NDN. They can compare the ease of setup and use on a single machine versus multiple physical machines.
- Contributions
  - Allow integration tests to run for every NFD, ndn-cxx, … code submission

Ashlesh Gawande, **Vince Lehman**, Lan Wang

#2. World Wide Web over NDN

- Motivation
  - World Wide Web (or the Web) is the primary tool billions use to interact on the Internet. NDN needs to provide World Wide Web service
- Tasks
  - Develop a system that allows a user to interact with a simple website over NDN. This website must contain the following features: (a) static content, such as pictures; (b) dynamic content, such as current timestamp; (c) user submission, such as posting an anonymous comment on a message board.
    - no cookies, no javascript
    - Web application should not be tailored for NDN. The website can be served from a FastCGI server, an HTTP server, a NodeJS http.Server-like object, or a Python WSGI webapp. The web server developed in this project should make the same website available over both HTTP and NDN.
    - ndn-protocol.xpi from NDN-JS can be used as the browser.
    - Alternatively, develop a JavaScript web application that fetches packets with NDN-JS in browser, and render them in an <iframe>; this application itself can be delivered via HTTP, but subsequent communication should be performed via NDN over WebSockets.
- Requirements
  - Knowledge of HTTP
- Metrics
  - Demonstrate static content, dynamic content, user submission features of the website works over both HTTP and NDN
- Contributions
  - Illustrate NDN is capable of serving World Wide Web

Junxiao Shi, **Teng Liang**

#3. NFD on Windows

NDN Forwarding Daemon (NFD) is an essential component of NDN platform

NFD is available on POSIX systems

Linux OS X FreeBSD Android

Raspberry Pi OpenWRT Odroid Intel Galileo

1.25 billion computers are running

and NFD isn't available to them

How can you help?
Cross compile NFD to run on Windows.

Junxiao Shi

#3. NFD on Windows

In this project:

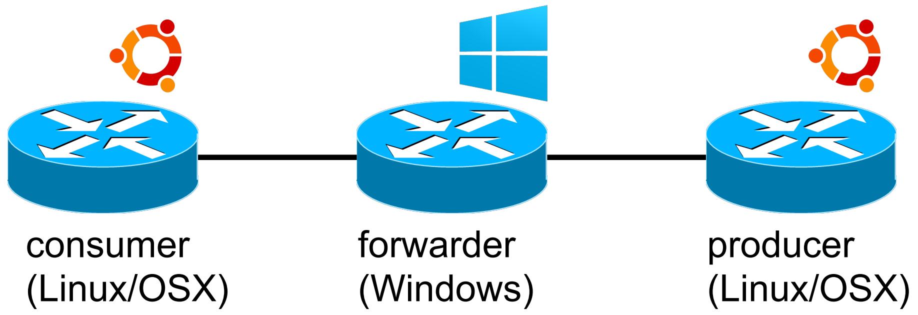
- Cross-compile NFD to run on Windows.
- You may do cross-compilation on either Linux or Windows. You may use Cygwin, MinGW, Subsystem for UNIX-based Applications (SUA).
- It's okay to disable certain NFD features (such as UnixStreamTransport).

You need:

- good knowledge about ndn-cxx and NFD
- a Windows 7/8/10 computer or virtual machine

Project demo:

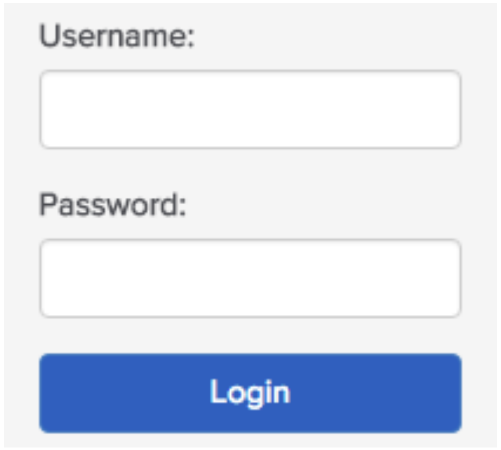
- NFD on Windows can forward packets between other nodes.



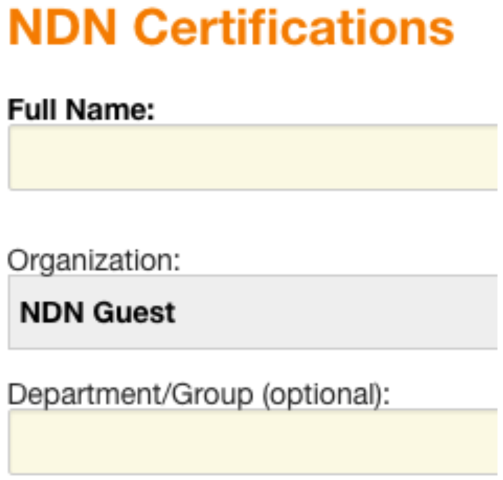
Junxiao Shi

#4. Certificate issuer for traditional web applications

Traditional web applications authenticates users with username+password.



NDN requires the user to have a certificate in order to produce contents. User must carry a personal device that holds the certificate.


NDN Certifications

Full Name:

Organization:
NDN Guest

Department/Group (optional):

Can we allow traditional web applications to use NDN communication while hiding the certificate issuance from the user?

Junxiao Shi

#4. Certificate issuer for traditional web applications

Workflow:

1. The web application is delivered over HTTPS, and is completely trusted by the user.
2. The user is authenticated with username+password, or through a third-party authentication service (e.g. Twitter OAuth).
3. For first time user, the web server generates and publishes a long-lived NDN key pair: /webapp-name/username. The private key is kept on the server.
4. During each login, the web server generates and publishes a short-lived NDN key pair: /webapp-name/username/session-id. The private key is sent to the browser.

#4. Certificate issuer for traditional web applications

Project demo:

- Demonstrate the system with a simple web application.
- Illustrate the keys are issued correctly, and others can verify Data packet when the user is online or has gone offline.

You need:

- frontend: JavaScript, NDN-JS
- backend: any server-side technology (e.g. PHP, Python)
- knowledge about NDN trust model and certificates

Junxiao Shi

#5. New Cache Decision Policies for NFD

- Motivation
  - NFD currently admits all incoming Data packets into the Content Store.
  - Therefore, the content inside multiple content stores is often similar, which decreases the cache hit ratio. We propose to randomize the admission decision to increase the cache diversity inside the network and thus improve caching efficiency.
- Tasks
  - Change the Content Store admission policy to a randomized policy.Options include:
    - (a) uniform random caching;
    - (b) leave copy down (LCD)
    - (c) more likely to cache a Data packet when it's closer to producer;
    - (d) more likely to cache a Data packet when it's closer to consumer;
    - (e) parse the name and attempt to cache all segments of a content version.
    - (f) other ideas
  - Evaluate the cache decision policy on a realistic topology (grid, fat-tree, etc.)
- Requirements
  - ndnSIM, C++, NFD
- Metrics
  - A report should show the results of this investigation: overall cache hit ratio; average distance from each node to the nearest cache replica, etc.
- Contributions
  - Improving the performance of in-network caching by increasing cache diversity

Junxiao Shi, **Klaus Schneider**

#6. NFD web control panel

Currently, NFD must be configured using nfdc on the command line.

It would be helpful to have a web interface through which we can configure and monitor NFD.

In this task, your goal should be to develop a web interface that performs many of the functions of nfdc and nfd-status, including:

- Displaying the current NFD version and global counters
- Displaying a list of NFD faces and their counters
- Creating a TCP or UDP unicast face
- Deleting an existing face
- Displaying a list of RIB routes
- Adding a route
- Removing a route

Junxiao Shi, **Eric Newberry**

#6. NFD web control panel

Evaluation Metric:

- Ease of use of the web interface

Technologies:

- HTML
- JavaScript
- NDN-JS
- NFD Management commands

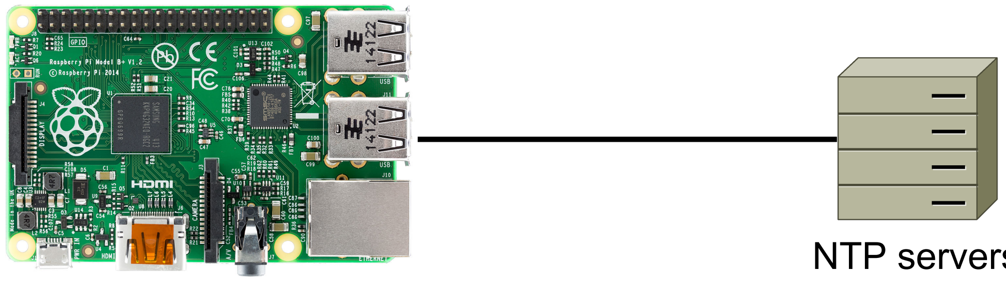
This task will help make configuring and managing NFD more user-friendly and will help in the further deployment of NDN infrastructure.

Junxiao Shi, **Eric Newberry**

#7. Clock setup for devices without hardware RTC

Certain cheap devices do not have a hardware Real Time Clock (RTC). When they are powered on, they don't know the current time.

Raspberry Pi relies on Network Time Protocol (NTP) to setup its clock during system boot.



NTP is an IP-based protocol, which will not work in a pure NDN network.
NTP is insecure: a malicious NTP server can return a fake timestamp.

We need a secure clock setup protocol in NDN.

#7. Clock setup for devices without hardware RTC

Protocol: allow a device without hardware RTC to obtain a timestamp from a time service over NDN network.

- The timestamp must be fresh, and accurate to the order of 100 milliseconds.
- The signature from the time service must be verified.
- The time service must serve the general public. It's unfeasible to manage one symmetric key per device.
- The key used by the time service must be replaceable, and every data signing key directly used by the time service has a validity period of at most one year. Key signing keys can have longer validity period.
- Network transmission, signing, and signature verification consume non-zero time.

Protocol design should be submitted as a document or slides.

#7. Clock setup for devices without hardware RTC

Project demo

- The prototype contains the time service and a client.
- The client code should not access system clock.
- Demonstrate the system over 0~500ms network delay between time service and client, and what accuracy it achieves.

You need

- entry-level knowledge about any NDN library (how to write consumer and producer)

Contribution / Challenge

- Explore trust model without relying on accurate clock

Junxiao Shi

#8. Implement Demo App (Headless or GUI) on Ubuntu Linux

- Motivation
  - NDN-RTC is a library for real-time streaming over NDN that is compatible with Linux. *ndncon* is a cocoa app on OS X which uses NDN-RTC to provide video and audio conferencing. As an initial effort towards creating *ndncon* for Linux, we want to build headless NDN audio-video conferencing application which will employ NDN-RTC for media streaming and be fully interoperable with OS X version of *ndncon.* GUI may be also developed if there will be an interest.
- Tasks
  - Setting up development environment: building dependencies (including WebRTC) and NDN-RTC library on Ubuntu Linux (15.04),
  - Creating a **headless application** using available source code and materials:
    - Code: https://github.com/remap/ndnrtc, https://github.com/remap/ndncon
    - NDN-RTC Background: http://named-data.net/publications/techreports/ndntr-0033-ndnrtc/
  - If resources allow, implement GUI version in parallel
- Requirements
  - C++ app experience with Ubuntu Linux, NDN-RTC, NDN-CPP
- Metrics
  - Complete an NDN-RTC headless app on Ubuntu that publishes and consumes media streams (audio and video) that are interoperable with existing OS X version.
  - Complete extension to include a GUI for the developed app.
- Contributions
  - Primarily a cross-platform implementation contribution, but one that will enlarge the ability to use NDN-RTC for project communication as well as evaluations

Jiachen Wang, Peter Gusev, Jeff Burke

#9. Create pilot energy dashboard using NDN-JS

- Motivation
  - The UCLA NDN team has bridged a subset of UCLA campus building monitoring data to the NDN testbed for research on NDN-based building management. This project creates a prototype energy dashboard using this data and provides us the opportunity to evaluate development ease for this type of application.
- Tasks
  - Create a consumption-only application that will use NDN-JS to access, verify, and (possibly) decrypt data arriving from the NDN testbed.
- Requirements
  - JavaScript, NDN-JS, data visualization (e.g., D3)
- Metrics
  - 1) Data retrieval based on mapping user choices to namespace selections;
  - 2) Verification of trust using library functions for this;
  - 3) Decryption (if necessary);
  - 4) Visualization. First with a single data type, then perhaps for multiple data types or aggregates.
- Contributions
  - It will provide experience in working with time-series data over NDN.
  - Creates an end-user facing sample application for the enterprise building management network environment; it verifies the applicability of the namespace and other aspects of the publishing design for a dashboard-style application. Feedback from developers will also provide opportunity to evaluate NDN ease of use for this scenario.

Zhehao Wang, Jeff Burke

#10. Define API for local prefix discovery

- Motivation
  - Some applications need to discover prefix(es) under which they can publish data that they can expect to be reached by Interests expressed on the global Internet. How to do so is defined here: http://named-data.net/doc/NFD/current/misc/local-prefix-discovery.html This project prototypes library support for retrieving this message according to the protocol.
- Tasks
  - Design API for applications to subscribe for reachable (routable) prefix updates
  - Implement desing in one of NDN libraries
- Requirements
  - NDN-CPP/PyNDN2/NDN-JS/jNDN
- Metrics
  - Demo app that demonstrates how it works
    - use ndn-autoconfig-server to locally announce different routable prefixes
    - [if NDN-JS] Use ndn-autoconfig-server from a test bed node from the browser.
- Contributions
  - Discovery of allowed local publishing prefixes is critical for applications in which a publisher is mobile, obtains local connectivity, or otherwise doesn't know forwarded prefixes in advance. This project provides an experimental base for continuing development of auto configuration capability.
- Project Wiki
  - https://github.com/jefft0/ccl-prefix/wiki
  - discovery API, resources

Jeff Thompson, Jeff Burke

#11. NDN over BTLE on the Arduino (1/2)

- Motivation
  - NDN does not require IP to communicate and may provide many benefits for IoT applications in the future; this project continues exploratory research on NDN support on the Arduino. It reports data from an analog sensor interface (or other demo input source on the Arduino as decided by the team).
- Tasks
  - This project starts with the existing NDN arduino sensor example here, https://github.com/named-data/ndn-cpp/tree/master/examples/arduino and extends it to communicate using Bluetooth LE on an Rfduino or similar device. It could require building a bridge application to accept data via BTLE on a compatible device and pass it to NFD via a socket (or an extension to NFD itself, but this may be too time-consuming), as well as a demo consumer app using an NDN library.
- Requirements
  - jNDN and/or ndn-cpp (lite), knowledge (or willingness to learn) Bluetooth API for both Arduino and the base station device.
- Metrics
  - 1) Send NDN packets over BTLE from Arduino.
  - 2) Associate with and communicate to a bridge application and pass packets to local NFD.

Jeff Thompson, Jeff Burke

#11. NDN over BTLE on the Arduino (2/2)

- Specific Requirements
  - Create a network of up to 10 RFduinos that sense or generate values and publish them via NDN, communicating using native NDN (TLV format) packets over BTLE.
  - They should gather and report their “readings” in Data packets returned in response to Interests issued over BTLE from a more capable device (either a phone or a laptop).
  - Use NDN-CPP Lite on the Arduino, and any NDN library along with the most current version of NFD on the base station device.
  - Little or no configuration should be required. (Discovery can be explored, or out-of-band exchange of BTLE addresses potentially used as names).
- Options
  - Any application scenario for these devices that meets the above requirements is acceptable. Try to create something fun (and clear) as a demo.
  - The more capable device can ideally act as a forwarder for other regular NDN devices, allowing them to access the RFduino network.
  - Names of the RFduinos can be implicit - e.g., their bluetooth address.
  - Non-default packet sizes can be used if needed or a fragmentation protocol can be implemented. (We only recommend the former, if needed.)
- Project Wiki
  - https://github.com/jefft0/ndn-btle/wiki

#12. Hack in NDNS code to be simulated using the latest ndnSIM

- Motivation
  - The lastest release of ndnSIM (http://ndnsim.net/2.1/) provides ability for applications (libraries) written using ndn-cxx library to drive simulation scenarios. At the same time, to allow simulation, there are several requirements that code has to meet (http://ndnsim.net/2.1/guide-to-simulate-real-apps.html) .
- Tasks
  - Create simulation environment to run experiments with NDNS codebase
    - Fork NDNS codebase and attach as a submodule to simulation repository
    - Enable compilation of relevant parts of NDNS code (not everything can be simulated)
    - Go through the code and (if necessary) fix issues that hinder ndnSIM support (refer to ndnSIM guide)
  - Design and write NDNS server and NDNS client NS-3 applications to drive experimentation
- Requirements
  - C++, ndnSIM, ndn-cxx
- Metrics
  - Using the developed scenario, demonstrate basic functionality and analyze results.
- Contributions
  - Enable simulation-based experimentation with NDNS (https://github.com/named-data/ndns) package

Alex Afanasyev

#13. Synchronize NLSR using Chronosync

- Motivation
  - Currently, NLSR uses its own forked and modified version of Chronosync, nsync, to synchronize a router's Link-State Database (LSDB). Thus, any updates, improvements, or bug fixes to Chronosync cannot be incorporated easily. This project proposes that NLSR should use the most up-to-date version of Chronosync directly and not maintain its own version.
- Tasks
  - Remove nsync from repository and use latest ChronoSync library
  - Convert code to publish new adjacency, coordinate, and name prefix information under distinct name prefixes
    - Currently, each information LSA is published under the same prefix with certain parts of the sequence number corresponding to each type
- Requirements
  - C++, ndn-cxx
- Metrics
  - Pass unit tests without embedded nsync
  - Run updated NLSR successfully on emulated testbed (e.g., with mini-ndn)
- Contributions
  - Using the latest version of Chronosync would greatly benefit NLSR and, in turn, improve the entire NDN testbed. NLSR would be able to take advantage of any improvements or bug fixes to Chronosync. Using 3 distinct prefixes would simplify logic and could make diagnosing issues in the log easier, and removing nsync from the NLSR repository would

Vince Lehman, Lan Wang

#14. nTorrent: BitTorrent in Named Data Networking

- Motivation
  - The goal of BitTorrent is to achieve efficient data dissemination by enabling data retrieval from multiple peers. BitTorrent is an application layer overlay. As a result, it has to both implement a data-centric paradigm on top of the point-to-point TCP/IP networks and infer information available at the network layer (e.g., peer distance, routing policies), so that to select the peers from whom to retrieve data.
  - NDN has also the goal of achieving efficient data dissemination. However, it offers the data-centric communication model directly at the network layer. Thus, it enables applications to care exclusively about data, as the network layer can retrieve data from multiple close locations in a way seamless to the application, and unburdens applications from peer discovery and selection.
- Tasks
  - Implement design described in TR
    - .Torrent file generator, .Torrent file publisher, .Torrent file downloader, .Torrent file parser
    - Hierarchical manifest generator, Hierarchical manifest publisher, Hierarchical manifest downloader, Hierarchical manifest parser
    - Torrent data segmentation, Torrent data publisher
    - Data integrity checker, Data fetching module
- Requirements
  - C++, ndn-cxx
- Metrics
  - ?
- Contributions
  - Demonstrates NDN ability to simplify a TCP/IP-based application design
  - Implemented prototype can become a library to be used in other NDN applications

Spyros Mastorakis, Alex Afanasyev, Yingdi Yu

#15. NDN-RTC Trust Verification and Encrypted Media

- Motivation
  - NDN-RTC is a library for real-time streaming over NDN (ala Skype or Google Hangouts) which provides basic signing and verification mechanisms with predefined keys. However, currently, there is no support for NDN trust model (trust schema and testbed certified keys) and media encryption is not provided, which makes communications insecure.
- Tasks
  - Implement signing & verification by authorized testbed keys in ndncon/NDN-RTC
    - define trust model, write trust schema, implement
  - Implement symmetric encryption of media data to enable secure RTC communications
    - design, implementation
  - Application (ndncon) should provide clear UI for distinguishing between secure/insecure media transfer as well as UI to allow user select certified key to sign published media streams. This will require some modifications/expansions of NDN-RTC library and ndncon application.
- Requirements
  - Advanced C++ and Objective-C (Cocoa) applications on OS X
- Metrics
  - Expand existing NDN-RTC library and ndncon application with functions to support secure media transmission.
- Contributions
  - Implementation contribution; more experience with using schematized trust. There is no secure key exchange mechanism currently implemented in deployed NDN applications.

Peter Gusev, Jeff Burke

#16. Access control with NFN

- Motivation
  - Today, cloud computing is a very important topic for IT system, while ICN solution mainly focus on delivering data. Named Function Networking (NFN) is an extension to Named Data Networking where a user can request complex results instead of single data items. NFN's task is to orchestrate the execution of such queries and to reuse results of (sub-) computations wherever possible. Beyond activating single functions (e.g. "/avg (/weather/ca/2015/08)") for consumption, users can also publish new services by binding a name to the corresponding computations.
- Tasks
  - Revisit existing code (NFN implemented in CCN-lite, Python bindings for NDN and Scala NFN-runtime) and either extend or replace it. Building and integrating a bridge to Hadoop systems could also be envisaged
  - Another important NFN application is access control that can be interposed between a data consumer and the objects and services residing in the NDN network. The project will start with an existing access control solution that runs on top of our named-function network. It enables fine-grained access control and auditing as well as (access controlled) chaining of data processing and filtering services.
  - The goal is to demonstrate how data can be filtered and made accessible to authorized parties only. We will provide GPS running data that should be anonymized and delocalized (shifting the coordinates to an arbitrary origin). The access control and filtering mechanism can be ported to other applications. We will provide a demo network for experiments and teach people how NFN can be used.
- Requirements
  - C, Python, Scala
- Metrics
  - The number and the complexity of the developed NFN applications
- Contributions
  - NFN is an extension to NDN, which enables user to request complex results and to perform operations similar to today’s cloud computing. NFN combines advantages of NDN with the idea of cloud computing and introduces a simple way to plug in additional

Christopher Scherb, Claudio Marxer,
Christian Tschudin