ndnSIM analysis/illustration tools for collected metrics

Problem: ndnSIM tracers generate hard to read and large output files. Create a set of scripts (python, R, etc.) for the analysis and illustration (using ggplot2, numpy, etc.) of the metrics collected by tracers.

Tasks to accomplish: 1) Creation of scripts, 2) Submit the scripts to Gerrit for review (the goal is to get them merged to the master branch ASAP), 3) Demonstration of the resulted illustration to the judges

Knowledge Requirements: ndnSIM, python or R, c++

Expected Outcome: A set of scripts to automate the analysis and illustration of metrics collected using the ndnSIM tracers

Demonstration of latest NFD features in ndnSIM

Problem:

- New features have been added to NFD (e.g., packet fragmentation and reassembly, congestion control, ad-hoc faces).
- The goal is to create ndnSIM simulation scenarios and ndnSIM-specific (NS-3-based) or real world applications (if needed) to demonstrate those features.
- This project would be very useful for the ndnSIM community, since there will be ready-to-use pieces of code for the users to experiment with the new NFD features
- **Tasks to accomplish:** 1) Create a set of ndnSIS simulation scenarios, applications and make all the changes needed to ndnSIM, so that newly added NFD features can be used (e.g., packet fragmentation and reassembly, congestion control, ad-hoc faces and any other the developers might consider important and useful)
  2) Create some related documentation, so that we can add the created code on the ndnSIM website
  3) Submit the code to Gerrit for review (the goal is to get them merged to the master branch ASAP)

Knowledge Requirements: ndnSIM, NFD, ndn-cxx, c++

Expected Outcome: A set of simulation scenarios, applications and any other changes needed

Firefly (Jeff Burke, Jeff Thompson)

A scalable, fault-tolerant message bus for mobile applications.

- Native NDN communication for robust and disruption tolerant communication at the edge.
- Cloud-hosted “real-time database”, Google Firebase, as its transport.

Problem:

- Sync, repo, prefix propagation, mobility implementations still at research level – hard to show how they will all work together and what they can achieve when robust.
- At the same time, many existing cloud-based solutions have problems if edge connectivity is intermittent or over multiple interfaces, and add latency.

Our approach:

- Show how current best-in-class (NDN locally, Firebase globally) can be used *now* for app benefit.

Tasks

- Finalize mapping between Firebase and NDN
- Develop bridge process for linking a local NDN network with a Firebase store
- Implement client code on ndn-dot-net and ndn-js
- Develop sample application, inspired by IoT and AR experiments already underway by NDN team

Required Knowledge

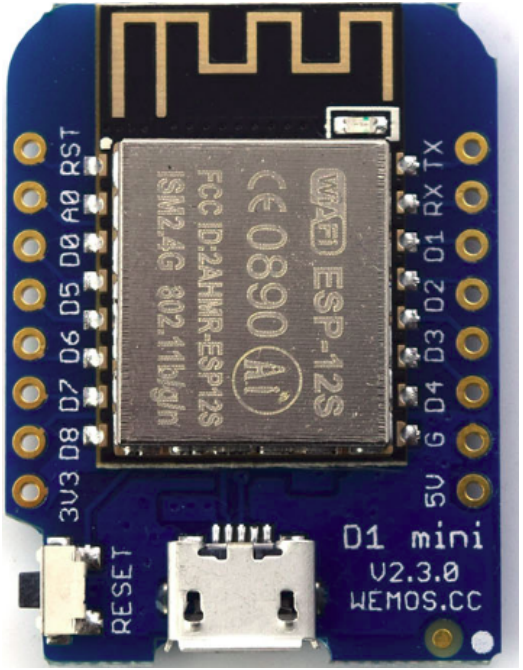
- Basic understanding of NDN including interest/data exchange, synchronization.
- Familiarity with at least one of the following languages, including how to use external libraries and write event-driven code: C#, NDN-JS, and Python.
- Helpful: Understanding of NoSQL databases

Outcomes

- Demonstration application
- Prototype library using Firebase as a transport
- Insight into new / similar / inspired functionality that could be provided via pure NDN solutions.

NDN Stack for ESP8266 Microcontroller

- Meet the ESP8266
  - 80MHz RISC CPU, 96KB data RAM
  - 802.11 b/g/n WiFi chip, full TCP/IP stack
  - GPIO pins, analog input, etc
  - small package, USD $3 each
- esp8266ndn library:
  - Arduino library to bring NDN to ESP8266
  - Face with UDP transport, Interest+Data, no dispatch
  - ndnping client and server, no prefix registration
- This project: let ESP8266 do more NDN
  - prefix registration
  - ECDSA signing and verification
  - Nack



- **Ashlesh Gawande**

NDN Stack for ESP8266 Microcontroller

- The little ESP8266 is western maker's favorite microcontroller with built-in WiFi.
  - Giving it NDN makes it more powerful, and makes NDN more appealing.
- You need:
  - knowledge about signed Interests
  - C++11, ndn-cxx or ndn-cpp-lite API
  - Arduino IDE and NFD on your computer, a full size USB 2.0 port
- Project leader will have at least 3 ESP8266 boards.
  - Recommended team size is 3. Some tasks do not require using an ESP8266 all the time.
- Project demos:
  - ndnping server with prefix registration and Nack
  - light control with ECDSA verification

- **Ashlesh Gawande**

NDN over Docker

- Still remember your first time running NFD?
  - You download this and that; compile this and that; install this and that. So let's make life easier. By packing the NDN into a Docker container, others can simply install the container and start "surfing" the NDN
- Tasks
  - Know how Docker and Docker container work
  - Redo all the steps we did for NDN setup and pack our results into a Docker container.
  - Test our container on other operating systems and let them talk in NDN.
- Required knowledge for participants
  - Once in your life time have successfully finished NDN environment setup.
- Expected outcome
  - If your OS supports Docker, then you can talk in NDN.

- **Zhiyi Zhang, Haitao Zhang**

Implementing Broadcast-based Self-learning Forwarding Strategy in NFD

- Motivation and problem statement
  - In local area networks and mobile ad-hoc networks, broadcast-based self-learning is a common mechanism to find packet delivery paths. The main benefits of this mechanism are its simplicity, adaptability, and support of mobility.
  - NFD does not have a broadcast-based self-learning forwarding strategy.
- Tasks
  - Discovery Interest Indication (#4355)
  - Prefix announcement for self-learning (#4280)
  - Measurement Table for self-learning
  - Self-learning forwarding strategy (#4290)
- Required knowledge for participants
  - C++
- Expected outcome
  - Accomplish as many tasks as possible

- **Teng Liang**

Demonstrating the Benefits of In-Network Congestion Detection

- Working Congestion Control Framework crucial for many applications
  - Currently: Ad-hoc solutions or poor performance.
- Demonstrate that Congestion Control can improve app performance
  - Build on earlier work (see Redmine)
- Tasks
  - Implement congestion detection via queue backlog or link loss detection
  - Implement congestion signaling, choose application, and evaluate on real hardware
- Required knowledge for participants
  - C++, NFD, socket programming, tc netem
- Expected Outcome: Demonstrate improved app performance

Klaus Schneider, Eric Newberry, Chavoosh Ghasemi

Overview

- This project is lead by Nicholas Gordon.
- The problem is that there are few ways to showcase NDN running on the Android platform, and no cases showing that current, popular software can be done using NDN.
- Further, there are few examples of out-of-band key exchanges for NDN, instead relying on the testbed to fully vet potential NDN users.
- The contributions to the NDN community will be two apps for Android and an exploration of using QR codes as a way to exchange information to facilitate secure sharing.

What are we going to do?

- Implement unsecure, simple file-sharing, between two Android phones using NDN.
- Develop a namespace for the photo system
- Extend the app to have “media aware” actions for exchanged photos.
- Implement the signature self-destruct feature.
- Implement QR code functionality with arbitrary text.
- Extend system to encrypt files.
- (Stretch goal) Integrate with any TPMs to improve security
- (Stretch goal) Use session keys to encrypt instead, improving cachability.
- (Stretch goal) Integrate with the testbed to enable sharing over it.

Other details

What do you need to know?

- NDN basics: namespace design, sync fundamentals, key systems
- Android app development
- For one of the stretch goals, Android system programming

What will we have done at the end?

- Delivered *two* usable, useful apps for Android that showcase NDN.
- Explore key exchange using QR codes and cameras.
- Explore challenges associated with efficent encryption and namespace design.

NFD measurements table manager

- Motivation
  - Currently, NFD's measurements table is not exposed to management clients
  - Hard to retrieve stats for measurements-based strategies such as ASF
  - Help operators and developers understand/debug the behavior of forwarding strategies
- Objective
  - Support per-prefix read-only access to the measurements table via NFD management protocol
- Requirements for participants
  - C++11
  - Understanding of how NFD management works
  - High-level understanding of how strategies and the measurements table are implemented
  - A machine capable of running NFD

- **Siham Khoussi, Davide Pesavento**

NFD measurements table manager

- Tasks
  1. Implement encoding/decoding of measurement entries, namely (in increasing order of difficulty):
     a) For a given name, represent the entry as a free-form string
        → Displayed as-is by the client
     b) For a given name, represent the entry as a map {face-id => string} or similar structure, where “string” contains face-specific measurements info
        → Strings displayed as-is by the client, indexed by face-id
     c) For a given name, represent the entry fields as structured TLVs
        → This requires the client to understand the TLV types, but is machine readable
  2. Implement an NFD manager that responds to `/localhost/nfd/measure/query/PREFIX`
  3. Implement new nfdc command: `nfdc measure query <PREFIX>`

- Expected Outcome
  - Demonstrate the new nfdc command on the ASF strategy, with ndn-traffic-generator or ndnping (can use Mini-NDN to emulate topology)

- **Siham Khoussi, Davide Pesavento**

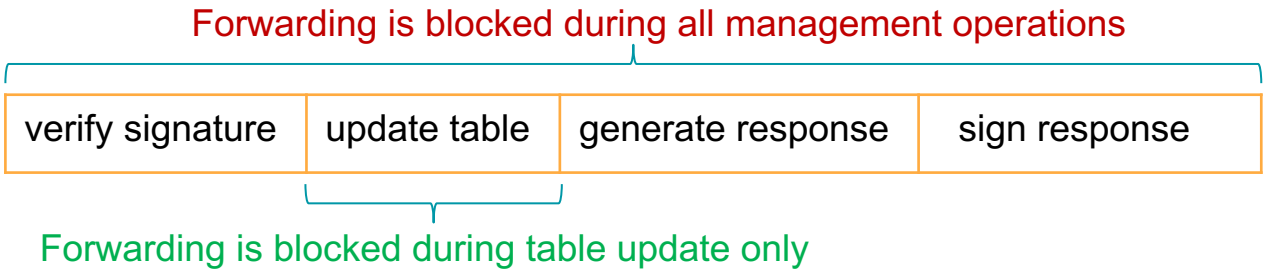
NFD Content Store Management

- Content Store: all important, little insight
  - Is in-networking caching effective? How many cache hits and misses?
  - What Data packets are stored in the Content Store?
- This project develops a Content Store (CS) management protocol:
  - Show CS hit/miss counters
  - Enumerate (a subset of) the Data packets stored in the CS
  - Provide a command to erase Data under a given prefix
- You need:
  - C++11
  - Computer or virtual machine capable of running NFD
  - Knowledge about NFD internals, especially management, is a plus

- **Junxiao Shi, Davide Pesavento**

NFD Management Thread

- Management slows down NFD!
  - NFD dispatches management Interests to the management module, in blocking mode.
  - Management module verifies command Interest signatures, generates response, and signs Data packets, all in the same thread as forwarding.
  - Packet forwarding is completely halted while management is running.



- This project moves management to a new thread
  - Heavy tasks run in the management thread, so they do not block forwarding.
  - We cannot make NFD data structures thread-safe in 12 hours, so we use a global lock: all tables are briefly locked while management accesses them.

NFD Management Thread

- Other related optimizations
  - The new management thread and the existing NFD-RIB talk to NFD forwarding (main thread) via Unix stream faces. This should be changed to an in-memory shared ring buffer to avoid socket operations and packet encoding.
  - NFD-RIB should be merged into the new management thread, so that RIB can update FIB directly and not via command Interests.
- You need:
  - C++11
  - Basic concepts of concurrency and thread synchronization
  - Knowledge about NFD internals, especially management
  - Computer or virtual machine capable of running NFD