FRACTALIDE

19 October 2018 | www.fractalide.com

Problem

Just as TCP/IP became popular because of the HTTP browser. So, NDN will become popular if an NDN browser is created.

Fractalide is an NDN browser built from scratch

Our code is Mozilla Public License v2, therefore, it’s suitable for hackers and businesses.

We spread copyright throughout the community. So, you own your code contributions.

We’re seeking to collaborate more deeply with the Named Data Networking community and hope to provide a good platform for researchers to develop leading edge NDN ideas that can be nurtured into real world applications used by many.

Need: Why is rkt-ndn needed?

Problem being addressed

- We're building an named data networking browser and want to leverage the work of academics and engineers.
- We make extensive use of racket and thus need a racket binding to a named data networking library.

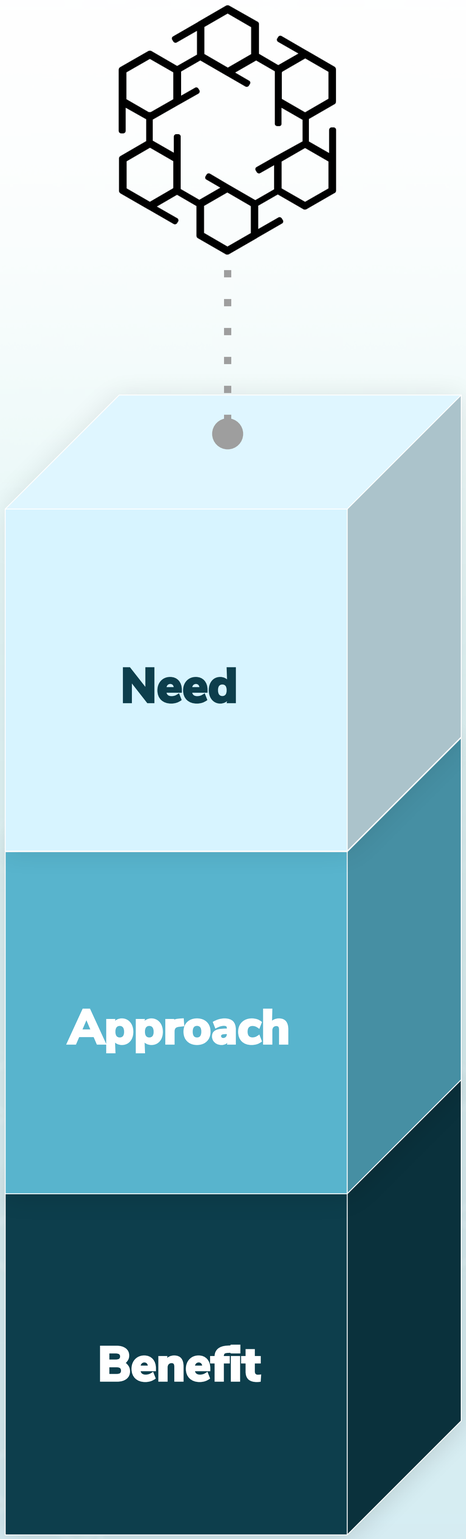
Hardpoints

- Building an undocumented **ndn-cpp** consumed 25% of our time by trying to figure obscure missing dependencies.
- The C++ layer of **ndn-cpp** and **ndn-cxx** cannot be handled by SWIG.
- C API is not all in the include directory.

Extra work

- Fixed **ndn-cpp** and got so that Continuous Integration builds it properly upon every Pull Request
- Packaged **ndn-cpp** for NixOS

Approach: Actual and expected approach taken



Technical Approach

The Fractalide platform is guided by three design principles; reusability, reproducibility and composability.

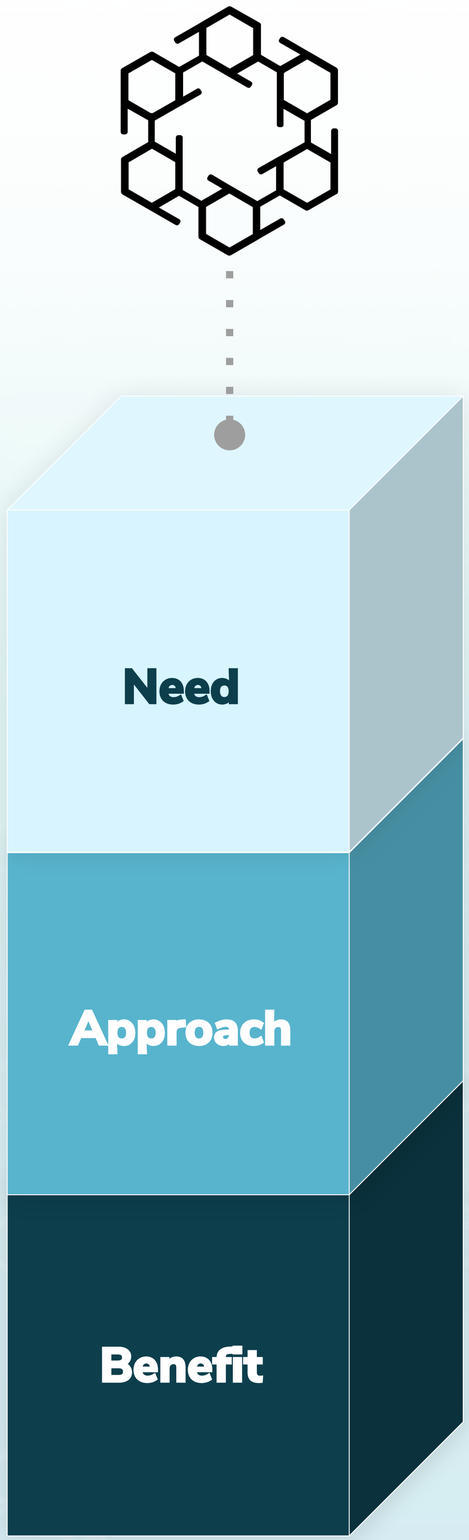
Expected Strategies

- We knew we'd start out with SWIG to generate racket bindings
- Should SWIG fail we'd have to use **racket/ffi** to talk to low level C in **ndn-cpp**. Undesirable.

Actual Strategies

- Use SWIG to generate bindings to **ndn-cxx** as the community use this the most: **Failed**
- Use SWIG to generate bindings on **ndn-cpp**, the community aren't using this so much, so not desirable: **Failed**
- Use SWIG to generate binding on **ndn-cpp** lite internal C++ layer: **Success (BUT 1** has racket issues **2** SWIG is outdated**)**
- Use **racket/ffi** to to talk to low level C API: **Success**

Surprises: no surprises



Benefit: Who benefits?

End Users

- Browser end users now have the theoretical dissemination capabilities of Google, Facebook and Twitter that's geared towards named chunks instead of setting up and reading from channels.
- Users can easily reproduce an entire researcher's app hierarchy from one top level name.

Domain Experts & Researchers

- Researchers will be able to use the flow-based (boxes and lines) interface to compose NDN applications together without looking at source code.
- Cutting edge applications can take a life of their own and actually be used by end users and remixed by programmers.

Programmers

- Programmers can improve & mix-and-match other researchers work easily.

Alternatives: What were the alternative methods?

Reimplement NDN using Rust

- The mix-and-match ease factor of **ndn-cxx** and **ndn-cpp** is extremely low, cumbersome and unfriendly. It should be easy and exciting to use. Rust offers many advantages over C++ & C. Primarily at targeting other platforms, providing solid tooling and a standard C interface which can easily be FFI'ed into from other languages.

If NDN is to become a widely adopted standard this should happen. We'd like to collaborate on this point by being a joint custodian of **github.com/named-data/ndn-rs** to ensure high standards. This solution is probably the best long term strategy but not achievable in one hackathon.Then a deliberate effort is made to deprecate **ndn-cpp** and **ndn-cxx** with a clear signal to the community this is the case.

Achieved: What was accomplished?

NixOS

- Completed a package to github.com/nixos/nixpkgs

NDN-CPP

- Submitted a patch to get **ndn-cpp** building properly on Travis-CI

RKT-NDN

- Successfully achieved a racket/ffi binding to the C APIof the **ndn-cpp** project. Further racket code must be written to recreate the higher level API needed for practical functional systems
- We only aimed to get one function to work, in this case it was the `ndn_getNowMilliseconds` function. It successfully works!

Links: the code

NixOS

- New ndn-cpp package to NixOS

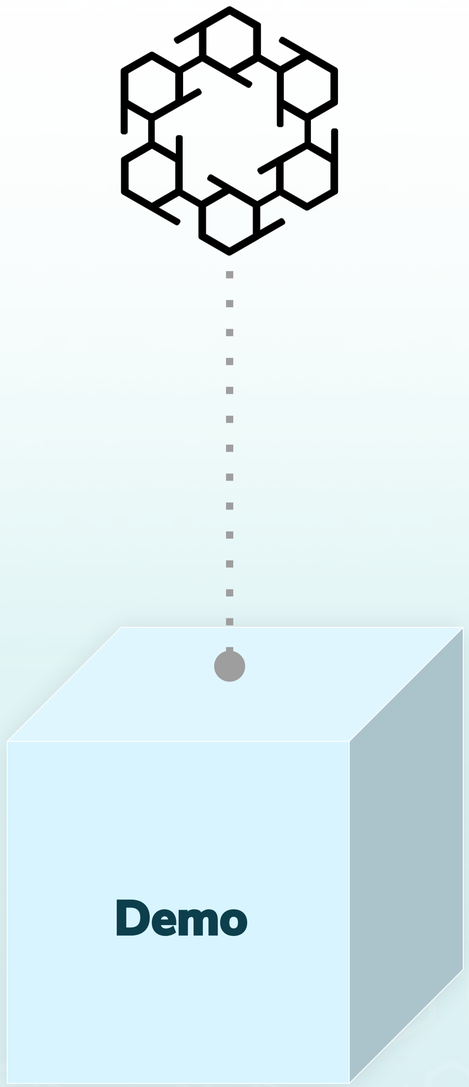
NDN-CPP

- Patch to fix CI building of ndn-cpp

RKT-NDN

- Source code to the primary hackathon objective - RKT-NDN

Demo: steps to reproduce



Reproduce

$ git clone https://github.com/7th-ndn-hackathon/rkt-ndn

$ cd rkt-ndn/ndn-cpp-c-bindings

$ nix-build

$ ./result/bin/test-ndn-cpp-c-bindings

epoch milliseconds: 1540153018518.364