NDN-RIOT Package Report

Tianyuan Yu
royu29@ucla.edu

Notes

Some parts of package are not well-written because I'm not a coding expert and still learning how to do things correctly. So any suggestions about coding or design you can provide are highly helpful! I'm keep tidying up code/comments and refining neccessary documentation. Apology again if my not so good coding style confuses you.

Package History&Overview

NDN-RIOT package is based on Wentao Shang's same named work in 2015, you can find the original paper work via NDN's publication list, or resort to link `https://named-data.net/wp-content/uploads/2015/01/design_implementation_ndn_protocol.pdf`.
Wentao's work provide NDN protocol stack on RIOT OS, but limited on basic Interest/Data exchanges. Original package creates a `ndn` thread aside from the user's main thread, serving networking things. Whereas, IoT scenario need built-in app-layer protocols (e.g., bootstrapping, service discovery) to facilitate development. The new package additionally create a `ndn-helper` thread to interact with core `ndn` thread, register faces, fib entries, etc. User can call `ndn-helper` function to retrieve issued certificate, neighbour identities and available services, and allocated access keys.

Environment Setting

Source

Use RIOT OS from `https://github.com/named-data-iot/RIOT` (not the official RIOT OS)
Use NDN-RIOT package from `https://github.com/Zhiyi-Zhang/ndn-riot`

Package Makefile

RIOT cloned from address above have already equipped with NDN-RIOT package (new old version of Wentao's work in 2015). To re-configure, go to folder `../RIOT/pkg/ndn-riot`, redirect the makefile here to a local source folder, or to remote github link (be sure of using the newest commit version number in makefile).
For example, if you'd like to replace original package, to `../RIOT/pkg/ndn-riot`, find `Makefile` and replace the package configuration with

```
PKG_SOURCE_LOCAL ?= $(RIOTBASE)/ndn-riot
PKG_BUILDDIR ?= $(PKGDIRBASE)/ndn-riot
```

Project Makefile

Each new project's which use this package should have makefile with

```
USEPKG += micro-ecc                                  #dealing with ECDSA signature
USEPKG += ndn-riot
USEMODULE += crypto                                 #dealing with crypto operation
USEMODULE += cipher_modes          #dealing with AES-128 cipher block chain mode
```

and CFLAGS to enable RIOT's crypto module
```
CFLAGS += -DCRYPTO-AES
CFLAGS += -DCRYPTO-THREEDES
```

Test and Examples

Basic APIs inherited from Wentao's original library can be found in example folder `https://github.com/Zhiyi-Zhang/ndn-riot-tests/examples`. But tests for each module are still missing. Protocols design can be found in the same repo's wiki page. Bootstrapping protocol is little complicated since we optimized it many times for speed issue.

Test-node-1 & Test-node-2

In the example folder, `test-node-1` serves as a encrypted content producer (e.g., heartbeat sensor). It first bootstraps with bootstrapping controller, fetching its identity and home prefix, then register serveral subprefixes to `ndn-helper-discovery` and broadcast. `test-node-2` serves as a encrypted content consumer. After bootstrapped and register&broadcast available services, consumer uses a ECDSA key pair to apply for the access of first listened identity's first service. Eventually consumer gets the producer's encryption key. Before running two test nodes, bootstrapping controller and access controller should be established first. Source code can be found in the same folder.

Usage

To start with `ndn-helper` related functions, call `ndn-helper-init` to create and initiate `ndn-helper` thread. To terminate, call `ndn-helper-terminate`. Most APIs mentioned can be found `helper-app.h`.

Bootstrapping

Call `ndn_helper_bootstrap_start` to passing a ECDSA key pair to `ndn-helper` to start bootstrapping thread. If success, issued certificate and parsing result will be kept in `ndn-helper`. User can call `ndn_helper_bootstrap_info` to retrieve the bootstrapping result. Thread will automatically terminate once finishing (success/timeout).

Service Discovery

Call `ndn_helper_discovery_init` to create and initiate the `ndn-helper-discovery` thread, `ndn_helper_discovery_register_prefix` is to register subprefixes for discovery. This function must be called before `ndn_helper_discovery_start`, which will broadcast available one's available services to the network. `ndn_helper_discovery_query` is used to query interested service with identity specify the interest receiver, this function (if success) will directly return the content block of data. Call `ndn_helper_discovery_terminate` to end the discovery thread.

Access Control

Call `ndn_helper_access_init` to create and initiate the `ndn-helper-access` thread.
`ndn_helper_access_producer` will use a key pair to contact access controller, trying to negotiate

a symmetric key. This function will return (if success) a pointer of negotiated symmetric key. Whereas `ndn_helper_access_consumer` requires ECDSA key pair and desired identity as inputs, and return a pointer of coresponding producer identity's encrytion (symmetric) key if application success. To terminate the `ndn-helper-access` thread, call `ndn_helper_access_terminate`.

Tips

Boards vs. Native

If you are using samr21-xpro, it can't run discovery and access control thread together for limited RAM. Currently you can try the combination bootstrap + discovery or bootstrap + access control. If you try as a native MacOS/Linux Process, RAM won't bother us.

TLV Encoding/Decoding

If having no specific explanation, identity names, service names and "subprefixes" are all encoded as Name TLV, although they are not exact "names". This is because Wentao's original library has well-supported APIs to cope with Name TLV blocks.

Debugging

Basically, most issues happen after one side receive the packet and begin processing. If one doesn't receive any packets, perhaps the reasons lie in the networking configuration or hardware modules imported.

Bootstrapping

With `ndn-helper`, native MacOS/Linux process or boards can perform a bootstrapping client role, but not the server part. Bootstrapping controller need configuration manually. Source code for bootstrapping controller can be seen at example folder. Such consideration is because we plan to re-implement the bootstrapping controller part over Android, where device are powerful enough to generate key pairs with enough security level. The similar situation also exist in access control module.

Service Discovery

Neighbour Table is only used in Service Discovery, to automatically collect available identities and services under these prefixes. Table will only be initiated once when you initiate the discovery thread. You can manually add/remove entries of the table if you need.

Access Control

Like bootstrapping, `ndn-helper` can only delegates the identity applying for access control or access keys. Access controller in the network need configuration manually. Source code in the example folder.

Key Parameters

Listed structure can be found in `helper-block.h`

ndn_bootstrap_t

1. `certificate`: hold the issue certificate from bootstrapping.
2. `home_prefix`: hold home prefix parsed from received certificate.
3. `anchor`: hold the anchor certificate (trust anchor) fetched in bootstrapping.

ndn_discovery_t

1. `identity`: used in query, indicating wanted identity
2. `service`: used in query, indicating wanted service

ndn_access_t

1. `ace`: keypair used for access control
2. `opt`: producer's optional parameters (current useless)/consumer's desired identity

ndn_keypair_t

1. `pub`: public key bits, should be 64 bytes
2. `pvt`: private key bits, should be 32 bytes

ndn_key_t

1. `key`: symmetric key bits
2. `len`: length of key bits