Motivation

- Named Data Networking (NDN) has shown great potential in supporting network applications in the IoT environments [1].
- The goal of this project is to bring NDN protocol support to the constrained IoT devices with 100s of KB memory and low-power CPU.
- We build on top of a popular IoT software platform called RIOT-OS [2].

Riot-OS Features

- Common OS abstraction across multiple platforms (ARM, Arduino, MSP430)
- Multi-threading + IPC
- Custom network stack
- C/C++ programming environment
- Standard build tools (gcc, make)
- Simulator for testing on Linux PCs

References


Application Interface

The NDN code on RIOT-OS is C99-compatible.

```
Object          Interface
Name             ndn_name_from_uri, ndn_name_append, ndn_name_print, ndn_name_compare_block, ndn_name_get_component_from_block
Interest         ndn Interest create, ndn Interest get name, ndn Interest get piv, ndn Interest get node
Data             ndn Data create, ndn Data get name, ndn Data get content, ndn Data get metainfo, ndn Data verify signature
APP Handle       ndn app create, ndn app run, ndn app destroy, ndn app schedule, ndn app express interest, ndn app register prefix, ndn app put data
```

List of API for NDN APP on RIOT-OS

```
static ndn_app_t handle = NULL;
static int on_data(ndn block_t *interest, ndn block_t *data)
{
    ndn block_t name;
    ndn data get name(data, &name);
    ndn name print(name);
    ndn block_t content;
    ndn data get content(&name, &content);
    // Do something with content...
    return ndn_APP_STOP;
}
```

Dem Application: ndn-ping

This demo application shows two RIOT-OS nodes running NDN-Ping client and servers respectively in a emulated network environment on a Ubuntu 15.10 machine. NDN packets are sent over Ethernet directly.

System Design

- Threads: APP, IPC, NDN, Net Device Driver
- OS core: Sched, IPC, SoftIRQ
- Hardware: CPU, Timer, Net Device
- Software architecture of NDN on RIOT-OS

The NDN protocol is implemented as a kernel thread. The IPC channel is used for:

- Passing NDN packets from & to APP and network device driver threads
- Sending configuration commands (e.g., add faces, register prefixes)

Currently implemented features:

- Basic packet forwarding logic (PIT, FIB, CS)
- Support for Ethernet and 802.15.4
- Memory efficient packet encoding & decoding
- HMAC-SHA256 data signing and verification

Limitations & Future Work

- Currently the code is only tested in emulated environments. The next step is to try it out on a real IoT device.
- The current implementation does not have routing support or FIB/RIB management. An interesting research direction is to provide routing functionality for constrained NDN-IoT networks.
- The current implementation does not include advanced NDN features such as forwarding strategies or cache management policies. It is yet unclear whether it is necessary to support those features on constrained devices.

Source Code

The source code of this work is available at https://github.com/wentaoshang/RIOT/tree/ndn/. It is currently released under LGPL v2.1, the same license used by RIOT-OS itself.

Acknowledgment

This work has been supported by the National Science Foundation under award CNS-1345318, CNS-1345142, CNS-1455794, and CNS-1455850.

Simple NDN consumer on RIOT-OS